Lenguaje de programación R

Lara Moreno Jesús Alejandro.
Negrete Montalvo Raúl Omar.
Enrique Sánchez Paredes

Temas.

- ¿Qué es R?
- Un poco de historia del lenguaje R.
- Características importantes de R.
- Ejemplos de aplicaciones en la actualidad.
- Introducción al lenguaje.
- Integración con un IDE.
- Los datos y sus tipos.
- Estructuras de control y manejo de datos.
- R y SQL.
- Graficación en R.
- Creación de GUI's en R.
- Bibliografía.

¿QUÉ ES R?.

R es un lenguaje y entorno de programación para análisis estadístico y gráfico

Historia del lenguaje R.

R fue creado en **1992 en Nueva Zelanda por Ross Ihaka y Robert Gentleman.**

La intención inicial con R, era hacer un lenguaje didáctico, para ser utilizado en el curso de Introducción a la Estadística de la Universidad de Nueva Zelanda.

Para ello decidieron adoptar la **sintaxis del lenguaje S** desarrollado por **Bell Laboratories.** Como consecuencia la sintaxis es similar a la del lenguaje S pero la semántica es sensiblemente diferente.

En 1993 se da un primer anuncio público del software R. En el **año de 1995 Martin Mächler**, de la **Escuela Politécnica Federal de Zúrich,** convence a **Ross y Robert** a usar la **Licencia GNU** para hacer de **R un software libre**. Como consecuencia, a partir de **1997, R forma parte del proyecto GNU**.

En 1996 se crea una lista pública de correos; sin embargo debido al gran éxito de R, los creadores fueron rebasados por la continua llegada de correos. **Y en 1997 crearon dos listas de correos**, **R-help** y **R-devel**, que son las que actualmente funcionan para responder las diversas dudas que los usuarios proponen.

Fue hasta **febrero de 29 del 2000**, que se considera al software completo y lo suficientemente **estable, para liberar la versión 1.0.**

Versión	Descripción
0.16	Es la última versión alfa desarrollada esencialmente por Ihaka y Gentleman, que incluye gran parte de las características descritas en el "White Book".
0.49 (23 de abril de 1997)	(23 de abril de 1997) Es la versión más antigua de la que se conserva el código (que todavía compila en algunas plataformas UNIX). En esta fecha arrancó también CRAN con tres espejos que albergaban 12 paquetes. Poco después aparecieron las versiones alfa para Windows y Mac OS.
0.60 (5 de diciembre de 1997)	R se integra oficialmente en el Proyecto GNU. El código se versiona a través de CVS.
1.0.0 (29 de febrero de 2000)	Los desarrolladores lo consideran suficientemente estable para su uso en producción.
1.4.0	Se introducen los métodos S4 y aparece la primera versión para Mac OS X.

Versión	Descripción
2.0.0 (4 de octubre de 2004)	Introduce el lazy loading, que permite una carga rápida de datos con un coste de memoria mínimo.
2.1.0	Aparece el soporte para UTF-8 y comienzan los esfuerzos de internacionalización para distintos idiomas.
2.9.0	El paquete 'Matrix' se incluye en la distribución básica de R.
2.11.0 (22 de abril de 2010)	Soporte para sistemas Windows de 64 bits.
2.13.0 (14 de abril de 2011)	Añadida una nueva función al compilador que permite acelerar las funciones convirtiéndolas a byte-code.
2.14.0 (31 de octubre de 2011)	Añadidos espacios de nombres obligatorios para los paquetes. Añadido un nuevo paquete de paralización.
2.15.0 (30 de marzo de 2012)	Nuevas funciones de balanceo de cargas. Mejorada la velocidad de serialización para grandes vectores.
3.0.0 (3 de abril de 2013)	Mejoras en GUI, funciones gráficas, gestión de memoria, rendimiento e internacionalización.

Datos sobre el lenguaje S.

- **S es un lenguaje** que fue desarrollado por **John Chambers** y colaboradores en Laboratorios Bell (AT&T), actualmente L**ucent Technologies, en 1976**.

- Este lenguaje, **originalmente fue codificado e implementado como unas bibliotecas de FORTRAN.** Por razones de eficiencia, en **1988 S fue reescrito**.

- En 1998 se liberó la versión 4 que es prácticamente la versión actual.

- **En 2004 inghstful** decide comprar el lenguaje a Lucent (Bell Laboratories) por la suma de **dos millones de dólares hasta la fecha es el dueño.**

- En el año **2008 TIBCO compra inghstful** por 25 millones de dólares.

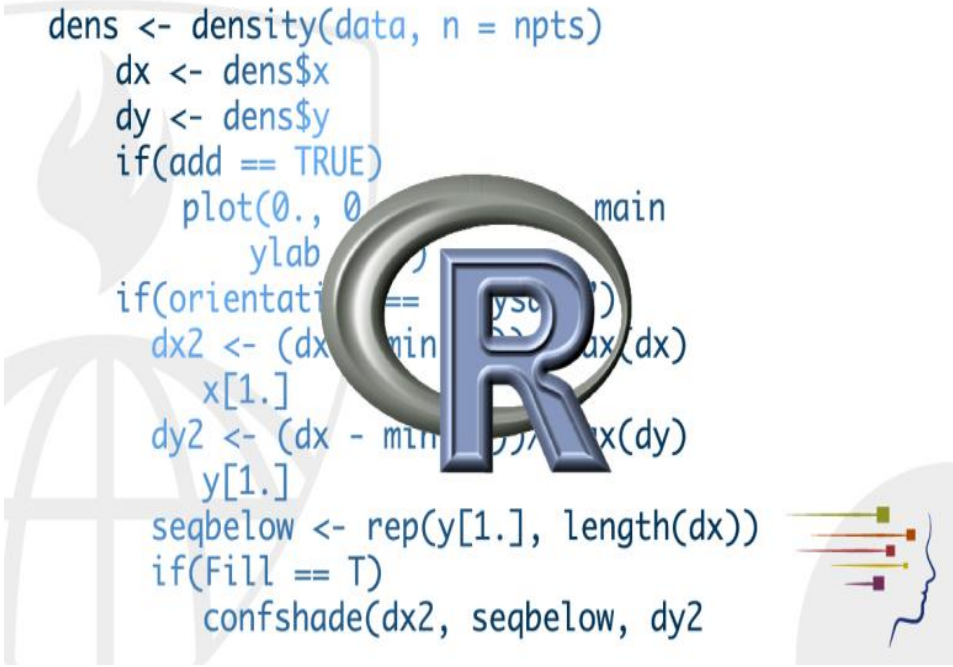
Características importantes de R.

Características de R.

- El sistema R está dividido en **dos partes conceptuales:**
    1- El sistema base de R, que es el que puedes bajar de CRAN.
    2- En todo lo demás.
- La capacidad de **gráficos de R es muy sofisticada** y mejor que la de la mayoría de los paquetes estadísticos.
- R es muy útil para **el trabajo interactivo,** pero también es un poderoso lenguaje de programación para **el desarrollo de nuevas herramientas**, por ejemplo rclimdex, cliMTA-R, etc.
- Tiene una **comunidad muy activa**, por lo que, **haciendo las preguntas correctas** rápidamente **encontrarás la solución a los problemas** que se te presenten en el ámbito de la programación con R.

- Al ser **software libre** lo hace un lenguaje atractivo, debido a que no hay que preocuparse **por licencias** y cuenta con la libertad que garantiza GNU.
- Debido a su estructura, **R consume mucho recurso de memoria**, por lo tanto si se utilizan datos de tamaño enorme, **el programa se alentaría** o, en el peor de los casos, **no podría procesarlos**.

Ejemplos de aplicaciones con R

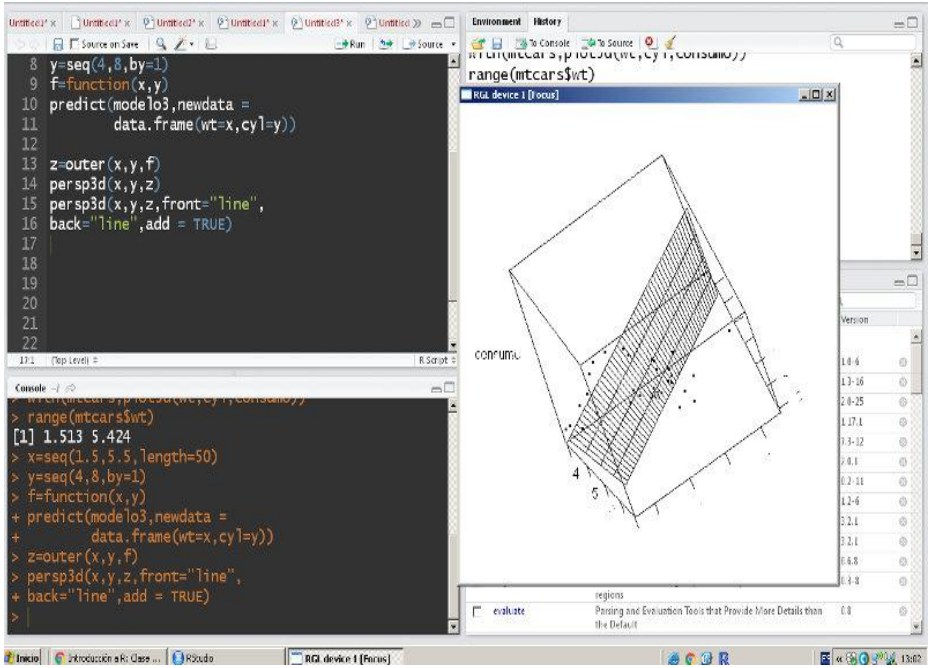


Ejemplos de aplicaciones de R.

- Aplicaciones forestales y medioambientales con SIG y Teledetección. Información.
- Aplicaciones de R para el análisis de datos en ecosistemas forestales. Información.
- Modelos generalizados aplicados a la Economía en lenguaje R. Información.
- Econometría: sus usos y aplicaciones en R. Información.
- Estadística y programación para oceanógrafos Información.

INTRODUCCIÓN AL LENGUAJE.

- Instalación y configuración.
- Sintaxis básica de R.
- Nuestro primer programa (“hola mundo”).
- Extensiones y paquetes.



Instalación y configuración.

La instalación de R es es muy simple, R se encuentra disponible para los **3 sistemas operativos más usados Windows, Linux y Mac**.

La página oficial donde se puede obtener es https://cran.rstudio.com

Para el caso de Windows y Mac la instalación no necesita una configuración propia como tal.

En linux para poner instalarlo se necesita hacer unos ajustes adiciones a la distribución que se esté utilizando, toda la documentación se encuentra en la página de descarga.

Instalación en Windows.

Como se mencionó anteriormente la instalación en windows es la más sencilla solo basta con ejecutar el instalador y este se encarga de la instalación y configuración de R.

Instalación Visual.

Sintaxis Básica de R.

La sintaxis de R es muy simple e intuitiva, por ejemplo. Un objeto puede ser creado con el operador “asignar” el cual se denota como una flecha con el signo menos y el símbolo “>” o “<” dependiendo de la dirección en que asigna el objeto, por ejemplo

$> ***n <- 15*** , $> ***5 -> n*** .

El nombre de un objeto debe **comenzar con una letra** (A-Z and a-z) y puede **incluir letras, dígitos** (0-9), y **puntos** (.). **R discrimina entre letras mayúsculas y minúsculas** para el nombre de un objeto, de tal manera que x y X se refiere a objetos diferentes.

Para definir una función en R se sigue una simple sintaxis la cual se define como:

```
nombreFuncion <- function(parametros, ..., ....) {
        ..............
        ..............
        return(datos)
}
```

Para usar esta función simplemente se llama por su nombre ejemplo:

$> ***x <- nombreFuncion(x1, 2,)*** o $> ***nombreFuncion(x1).***

Nuestro primer programa (“Hola mundo”).

Ya que conocemos la sintaxis básica de R ahora si podemos crear nuestro primer programa el famoso “Hola mundo”, para eso vamos a hacer uso de la función print(x) donde x es una una cadena de caracteres. Solo con ejecutar:

$> ***print(“Hola mundo :D”)***

Obtenemos nuestro mensaje, ahora, ¿Como llamamos esta función dentro de otra función?

Ejemplo visual.

Extensiones y paquetes.

R consta de un sistema base y de paquetes adicionales que extienden su funcionalidad.

Tipos de paquetes:

1. Los que forman **parte del sistema base** (ej. ctest).
2. Los que **no son parte del sistema base**, pero son recommended (ej., survival, nlme) En GNU/Linux y Windows ya forman parte de la distribución estándar.
3. **Otros paquetes**; ej., UsingR, foreing, los paquetes de Bioconductor (como multtest, etc.). **Estos se han de seleccionar e instalar individualmente.**

Instalación de un paquete.

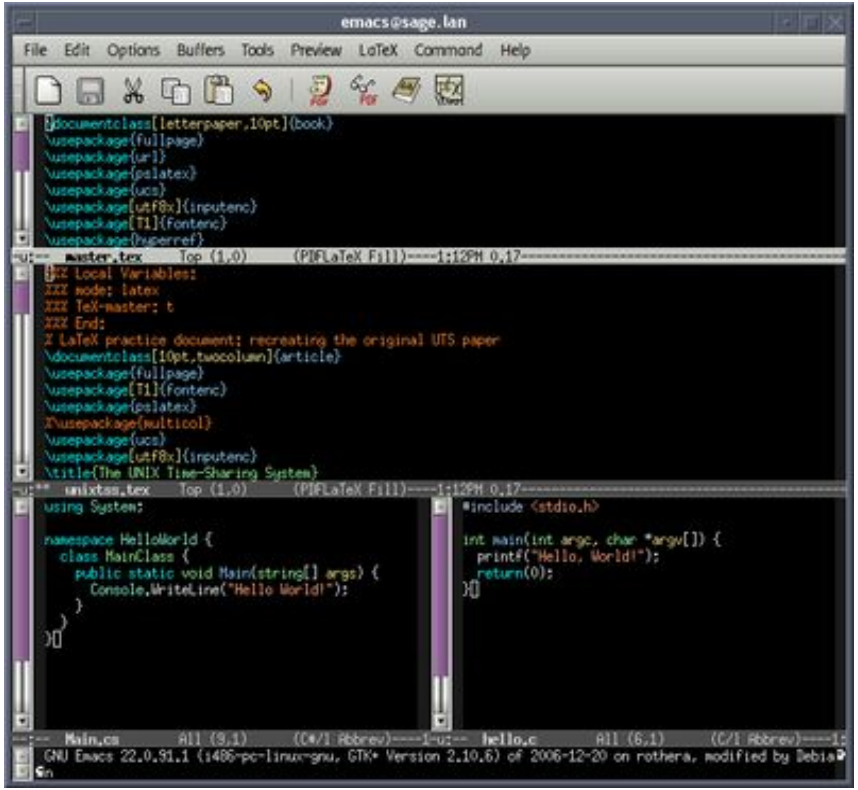
Para instalar un paquete en R existen varias maneras pero la mas rapida y simple que funciona en los 3 sistemas operativos es mediante línea de comandos.

install.packages(Nombre paquete).

Ejemplo visual instalación del paquete RSQLite

INTEGRACIÓN CON UN IDE.

- ¿Qué es un IDE?
- IDEs compatibles con R.
- Instalación e integración con RStudio.



¿Qué es un IDE? (Integrated Development Environment).

Un IDE es una **herramienta** que nos ayuda a **desarrollar de una manera amigable nuestras aplicaciones**, brindándonos ayudas visuales en la sintaxis, plantillas, wizards, plugins y sencillas opciones para probar y hacer un debug.

Muchas veces existe la confusión entre el lenguaje y el IDE, por poner un ejemplo hay personas que creen que sin tener instalado un IDE de Java (Como Netbeans o Eclipse) no se puede desarrollar en el lenguaje. Esto no es cierto, nosotros podemos desarrollar sin ningún IDE conociendo la documentación del lenguaje y del sistema operativo.

IDEs compatibles con R.

Instalación e integración con 

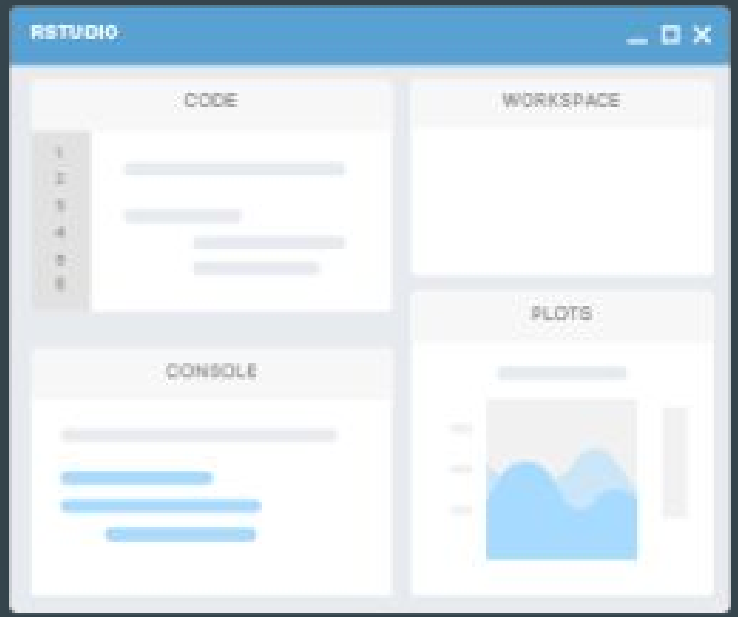
RStudio es (IDE) para R. Incluye una consola, un editor de resaltado de sintaxis que admite la ejecución directa de código, así como herramientas para trazar, historial, depuración y gestión del espacio de trabajo.

RStudio está disponible en código libre y comercialmente, se ejecuta en el escritorios (Windows, Mac y Linux) o en un navegador conectado a RStudio Server o RStudio Server Pro.

Para su instalación solo basta con ir a su página oficial https://www.rstudio.com y descargar la versión adecuada a su sistema operativo, el único requisito es tener instalado R.

Interface a RStudio.

RStudio consta de una interfaz sencilla y amigable a los nuevos usuarios.



Code: Es el área donde se edita y visualiza todo nuestro código de R.

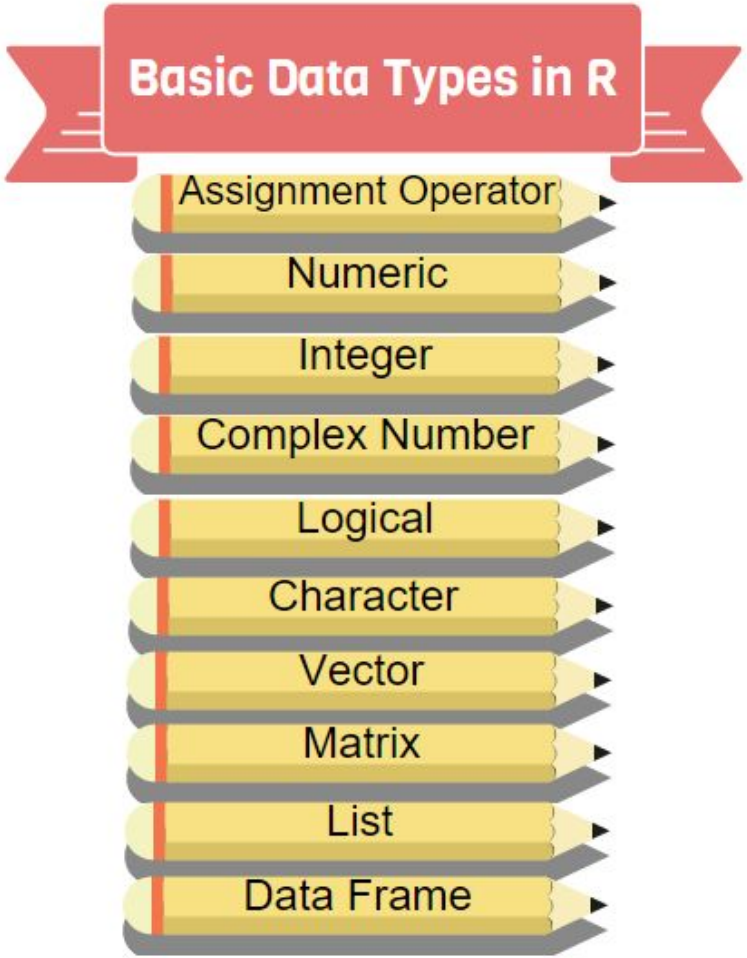
Workspace: En esta área se encuentran todas nuestras variables y paquetes que utilizamos.

Console: Es el área donde se ejecutan las instrucciones de R.

Plots: Se visualizan todos los gráficos.

LOS DATOS Y SUS TIPOS.

- Datos numéricos.
- Vectores.
- Matrices.
- Listas.
- Data frames.
- Funciones.



Los datos y sus tipos.

Todas las cosas que manipula R se **llaman objetos.** En general, éstos se **construyen a partir de objetos más simples**. De esta manera, se llega a los objetos más simples que son de **cinco clases** a las que se **denomina atómicas** y que son las siguientes:

1. **character** (cadenas de caracteres)
2. **numeric** (números reales)
3. **integer** (números enteros)
4. **complex** (números complejos)
5. **logical** (lógicos o booleanos, que sólo toman los valores True o False)

Datos numéricos.

Probablemente el principal uso de R es la **manipulación de datos numéricos.** El lenguaje agrupa estos datos en tres categorías:

1. **numeric**
2. **integer**
3. **complex**

Cuando se introduce algo que puede interpretar como un número, su inclinación es tratarlo como un dato de tipo numeric, es decir, un número de tipo real, a no ser que explícitamente se indique otra cosa. ejemplo visual.

Tipos de objetos

objeto	tipos	varios tipos posibles en el mismo objeto?
vector	numérico, caracter, complejo *o* lógico	No
factor	numérico *o* caracter	No
arreglo	numérico, caracter, complejo *o* lógico	No
matriz	numérico, caracter, complejo *o* lógico	No
data.frame	numérico, caracter, complejo *o* lógico	Si
ts	numérico, caracter, complejo *o* lógico	Si
lista	numérico, caracter, complejo, lógico función, expresión, . . .	Si

Vectores.

Se ha dicho con anterioridad que las **clases atómicas de datos no se manejan de manera individual.** En efecto, en **todos los ejemplos anteriores**, **el lenguaje ha creado implícitamente vectores de longitud 1**, y son esos los que se han asignado a las variables.

La primer manera de crear vectores es a partir de los elementos individuales que compondrán el vector. Para esto se utiliza la función ***c()***.

La función c() sirve para **concatenar varios elementos del mismo tipo**.

```r
c(4,2,-8)  # Creación de un vector sin asignarlo a una variable

## [1]  4  2 -8

## ----------------
## Distintas formas de asignar un vector a una variable
u <- c(4,2,-8) # Usando el operador <-
c(4, 2, -8) -> v # Usando el operador ->
# Usando la función assign:
assign("w", c(4, 2, -8))
p = c(4, 2, -8) # Usando el operador =
print(u); print(v); print(w); print(p)

## [1]  4  2 -8
## [1]  4  2 -8
## [1]  4  2 -8
## [1]  4  2 -8
```

Operaciones aritméticas con vectores.

```
v <- 2 + 3  # Resulta en un vector de longitud 1
v

## [1] 5

v <- c(2, 3) - c(5, 1)  # Resulta en un vector de longitud 2
v

## [1] -3  2

v <- c(2, 3, 4) * c(2, 1, 3)  # Resulta en un vector de longitud 3
v

## [1]  4  3 12

v <- c(2, 3, 4)^(3:1)  # Eleva a potencias 3,2,1
v

## [1] 8 9 4
```

Matrices.

Desde el punto de vista del lenguaje, una matriz es un vector con un atributo adicional: ***dim***.

Para el caso de las matrices, **este atributo es un vector entero de dos elementos**, el número de renglones y el número de columnas que componen a la matriz.

Una de las formas de construir una matriz es a partir de un vector, como sigue:

```r
(m <- 11:30)  # Un vector con 20 números
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
## [19] 29 30
```

```r
# Para convertirla en matriz simplemente se especifica el
# atributo dim
dim(m) <- c(4, 5)  # 4 renglones y 5 columnas
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   15   19   23   27
## [2,]   12   16   20   24   28
## [3,]   13   17   21   25   29
## [4,]   14   18   22   26   30
```

```r
class(m)
```

```
## [1] "matrix"
```

Las matrices también se pueden crear de manera flexible por **medio de la función primitiva matrix()**, que permite alterar la secuencia por default de armado de la matriz; esto es, ahora, si se quiere, se puede armar la matriz por renglones en vez de columnas:

```
(m <- matrix(11:30, nrow = 5, ncol = 4, byrow = TRUE))

##      [,1] [,2] [,3] [,4]
## [1,]   11   12   13   14
## [2,]   15   16   17   18
## [3,]   19   20   21   22
## [4,]   23   24   25   26
## [5,]   27   28   29   30
```

Listas.

Una lista, de la clase list, es una clase de datos que puede contener cero o más elementos, cada uno de los cuales puede ser de una clase distinta. Por ejemplo, se puede concebir una lista para representar una familia: la mamá,

el papá, los años de casados, los hijos, y las edades de los hijos, de la manera

siguiente:

```
familia <- list("Maria", "Juan", 10, c("Hugo", "Petra"), c(8,
    6))
familia

## [[1]]
## [1] "Maria"
##
## [[2]]
## [1] "Juan"
##
## [[3]]
## [1] 10
##
## [[4]]
## [1] "Hugo"  "Petra"
##
## [[5]]
## [1] 8 6
```

Al igual que en el caso de los vectores, los elementos de las listas pueden ser nombrados, lo que añade

mayor claridad a su significado dentro de la lista. La forma de hacer esto se muestra a continuación:

```
familia <- list(madre="Maria", padre="Juan", casados=10,
                hijos=c("Hugo", "Petra"), edades=c(8, 6))
familia

## $madre
## [1] "Maria"
##
## $padre
## [1] "Juan"
##
## $casados
## [1] 10
##
## $hijos
## [1] "Hugo"  "Petra"
##
## $edades
## [1] 8 6
```

Data Frames.

```
(m <- cbind(ord=1:3, edad=c(30L, 26L, 9L)) )

##      ord edad
## [1,]   1   30
## [2,]   2   26
## [3,]   3    9

(v <- c(1.80, 1.72, 1.05) )

## [1] 1.80 1.72 1.05

ff <- data.frame(familia=c("Padre", "Madre", "Hijo"),
                 m, estatura=v)
ff

##   familia ord edad estatura
## 1   Padre   1   30     1.80
## 2   Madre   2   26     1.72
## 3    Hijo   3    9     1.05
```

Un data frame es una lista, cuyos componentes pueden ser vectores, matrices o factores, con la única salvedad de que las longitudes, o número de renglones, en el caso de matrices, deben coincidir en todos los componentes.

La apariencia de un data frame es la de una tabla y una forma de crearlos es mediante la función data.frame().

Funciones.

A diferencia de otros lenguajes de programación procedurales, como C, Java, y PHP, en R las funciones constituyen una

clase. Por ejemplo, los objetos de esa clase pueden ser asignados a variables; podría darse el caso, incluso, de

armar una lista cuyos elementos fueran funciones.

La sintaxis para la creación de una función es como sigue:

variable <- **function**(*arg_1, arg_2, ..., arg_n*) *expresion*

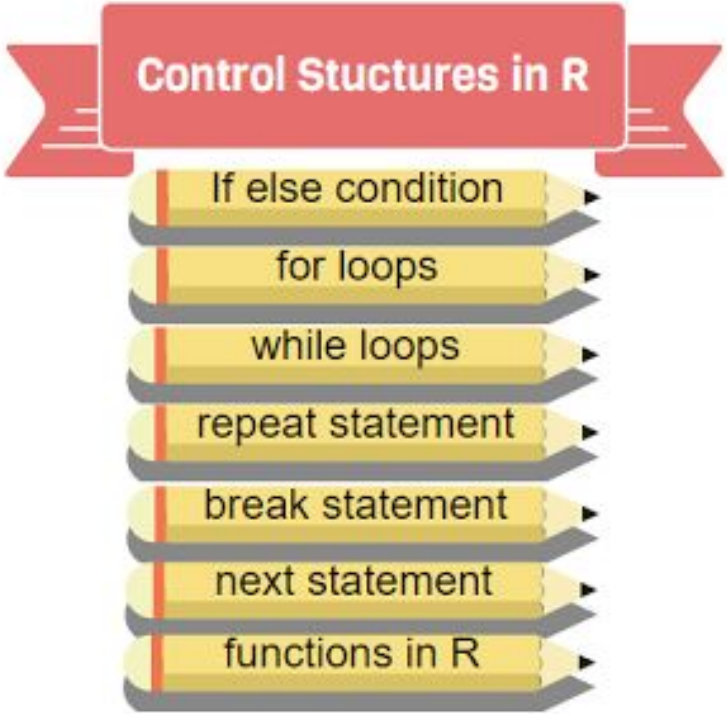
Como se puede ver, se trata de una asignación de un valor: la función, a una variable. A partir de esa definición, la variable se puede utilizar como el nombre de la función. En R, toda expresión tiene un valor, así que el valor de

la expresión será lo que la función regresará cuando se aplique. En seguida se muestra un ejemplo.

```
hipotenusa <- function(x, y) {
    sqrt(x^2 + y^2)
}

class(hipotenusa)

## [1] "function"
```

ESTRUCTURAS DE CONTROL Y MANEJO DE DATOS.

- Construcciones IF-ELSE.
- Ciclos.
- Leyendo datos de archivos.
- Manipulación de objetos.
- Clasificación, transformación y agregación de datos



Operadores

Operadores					
Aritméticos		Comparativos		Lógicos	
+	adición	<	menor que	! x	NO lógico
-	substracción	>	mayor que	x & y	Y lógico
*	multiplicación	<=	menor o igual que	x && y	id.
/	división	>=	mayor o igual que	x | y	O lógico
^	potencia	==	igual	x || y	id.
%%	módulo	!=	diferente de	xor(x, y)	O exclusivo
%/%	división de enteros				

La construcciones IF-ELSE

Estas construcciones son semejantes a las de otros lenguajes de programación, con una salvedad que puede ser capitalizada por los usuarios del lenguaje: la construcción en sí misma regresa un valor, que puede, si se quiere, ser asignado a una variable o utilizado de otras maneras.

```
aa <- 15
if (aa > 14) # if sin else
  print("SI MAYOR")

## [1] "SI MAYOR"

if (aa > 14) print ("SI MAYOR")

## [1] "SI MAYOR"

if (aa > 14) { # Instrucción compuesta

  print ("PRIMER RENGLON")
  print ("SI MAYOR")
}

## [1] "PRIMER RENGLON"
## [1] "SI MAYOR"

# Usando el valor que regresa el if
y <- 10
y <- if (aa > 14) 50
y

## [1] 50
```

La construcción IF admite una sola expresión, pero ésta puede ser la expresión compuesta, que se construye mediante los paréntesis de llave { }, y las expresiones en su interior, separadas ya sea por el cambio de renglón o por ';'. En los casos anteriores, la expresión señalada por el if se ejecuta u omite dependiendo del valor de la condición, TRUE o FALSE, respectivamente.

```
if (10 > aa) { # 1er. bloque
    print("RANGO MENOR")
} else if ( 10 <= aa && aa <= 20) { # 2o. bloque
    print("primer renglon"); print("RANGO MEDIO")
} else { # 3er. bloque
    print("RANGO MAYOR")
}

## [1] "primer renglon"
## [1] "RANGO MEDIO"
```

Bucles

El lenguaje cuenta con varios tipos de ciclos o repeticiones, a saber: repeticiones por un número determinado de veces, repeticiones mientras se cumple una condición y repeticiones infinitas. En seguida se discutirá cada uno de estos casos.

Bucles

Existen dos tipos de bucles

- **For** – Si sabemos el numero de iteraciones
- **While** – Si el numero de iteraciones depende de cálculos que se efectúen en el propio bucle.

Ejemplo con For – Sacar el cuadrado de los primeros diez números naturales

```
for (i in 1:10) print(i^2)
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

Ejemplo con While - Escribir todos los números naturales cuyo cubo sea inferior a 100.

```
i <- 1
i3 <- i^3
While (i3 < 100){
    cat(i, i3, "\n")
    i <- i + 1
    i3 <- i^3
```

Repeticiones por un número determinado de veces

La construcción que habilita esta operación es la instrucción for. El número de veces que se repite la expresión o expresiones englobadas en la instrucción, puede estar explícita en ella misma, como se muestra a continuación:

```
letras <- c("c", "l", "i", "M", "T", "A")
for (i in 1:6) {
    print(letras[i])
}
```

```
for (letra in letras) {
    print(letra)
}
```

```
for (i in seq_along(letras)) {
    print(letras[i])
}
```

```
## [1] "c"
## [1] "l"
## [1] "i"
## [1] "M"
## [1] "T"
## [1] "A"
```

En el tercer caso se llamó a la función seq_along(), que genera una secuencia de enteros de acuerdo con el número de elementos que tenga el objeto que se le da como argumento. El segundo caso, tipifica la esencia de la construcción for, ya que se trata de ir tomando uno a uno los elementos del objeto consignado después de la partícula in del for.

Repeticiones mientras se cumple una condición

La construcción que habilita esta operación es la instrucción while, que se puede ejemplificar como sigue:

```
# Para la misma tarea de los ejemplos anteriores
i <- 1
while (i <= 6) {
    print(letras[i])
    i <- i + 1
}
```

```
## [1] "c"
## [1] "l"
## [1] "i"
## [1] "M"
## [1] "T"
## [1] "A"
```

En este caso, si no se tiene cuidado en el manejo del índice i, involucrado en la condición, se puede dar lugar a un ciclo sin salida.

Repeticiones infinitas

La construcción que habilita esta operación es la instrucción repeat.

```
i <- 1
repeat {
    print(letras[i])
    i <- i + 1
    if (i > 6)
        break
}
```

El flujo normal de todos los ciclos se puede interrumpir básicamente por medio de tres instrucciones diferentes: break, next y return.

Leyendo datos de archivos.

Una manera de crear un vector es a partir de un archivo de texto. Sea, por ejemplo, el caso del archivo UnVec.txt, que se contiene la siguiente información: 12 15.5 3.1 -2.2 0 0.0007

Supóngase ahora que a partir de esos datos se quiere crear un vector. Para eso se usa la función scan(), como se muestra a continuación:

```
vec <- scan("UnVec.txt")
print(vec)

## [1] 12.0000 15.5000  3.1000 -2.2000  0.0000  0.0007
```

La función scan() ofrece la posibilidad de indicarle explícitamente el tipo de vector que se quiere crear. Así por ejemplo, la creación de un vector de enteros se hace de la siguiente manera:

```
vec <- scan("IntVec.txt", integer())
print(vec); class(vec) # El vector y su clase

## [1]   4   3  -2   1   0 200  -8  20
## [1] "integer"
```

La función inversa, en este caso, de la función scan(), es la función write. Así, un vector cualquiera fácilmente se puede escribir en un archivo de texto, como se muestra a continuación:

```
vv <- c(5, 6.6, -7.7)
write(vv, "OtroArchivo.txt")
```

R y SQL.

R y SQL

El lenguaje R proporciona muchas características para seleccionar datos desde data frames.

Con el paquete de sqldf, sólo debes asumir que tu data frame es una como una tabla de una base de datos y usar SQL directamente, sí, directamente! Maravilloso no?.

El paquete **sqldf** utiliza su **propio motor de base de datos interna**, por lo que **no hay que hacer una configuración especial para una base de datos**. Sólo tienes que introducir la siguiente sentencia en R para instalarlo y directamente utilizarlo:

```
$> install.packages("sqldf")
```

```
$> library(sqldf)
```

GRAFICACIÓN EN R.

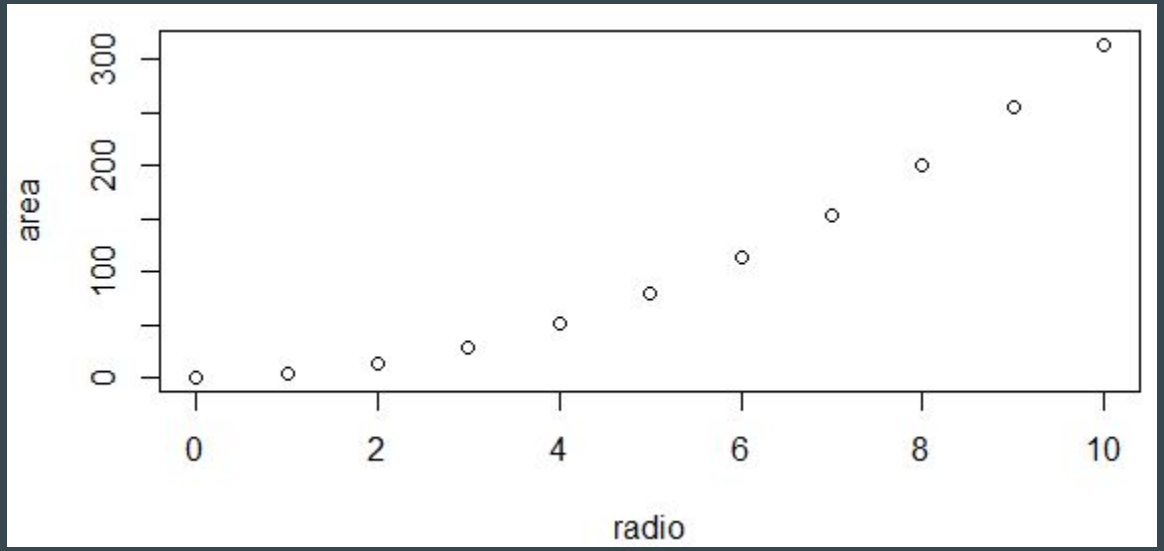
Una de las grandes virtudes del lenguaje R, es la facilidad que ofrece para presentar la información correspondiente a los datos que maneja o a los cálculos que desarrolla, de una manera gráfica. El lenguaje cuenta, no con uno, sino con varios sistemas, en general separados, para organizar o especificar visualizaciones gráficas.

Funciones Gráficas

`plot(x)`	graficar los valores de `x` (en el eje *y*) ordenados en el eje *x*
`plot(x, y)`	gráfico bivariado de `x` (en el eje *x*) y `y` (en el eje *y*)
`sunflowerplot(x, y)`	igual a `plot()` pero los puntos con coordenadas similares se dibujan como flores con el número de pétalos igual al número de puntos
`piechart(x)`	gráfico circular tipo 'pie'
`boxplot(x)`	gráfico tipo 'box-and-whiskers'
`stripplot(x)`	gráfico de los valores de `x` en una línea (como alternativa a `boxplot()` para pequeños tamaños de muestra)
`coplot(x~y | z)`	gráfico bivariado de `x` y `y` para cada valor o intervalo de valores de `z`
`interaction.plot (f1, f2, y)`	si `f1` y `f2` son factores, grafica el promedio de `y` (en el eje *y*) con respecto a los valores de `f1` (en el eje *x*) y de `f2` (curvas diferentes); la opción `fun` permite escoger un estadístico de `y` (por defecto, el promedio: `fun=mean`)
`matplot(x,y)`	gráfica bivariada de la primera columna de `x` *vs.* la primera columna de `y`, la segunda columna de `x` *vs.* la segunda columna de `y`, etc.
`dotplot(x)`	si `x` es un marco de datos, hace un gráfico de puntos tipo Cleveland (gráficos apilados fila por fila y columna por columna)

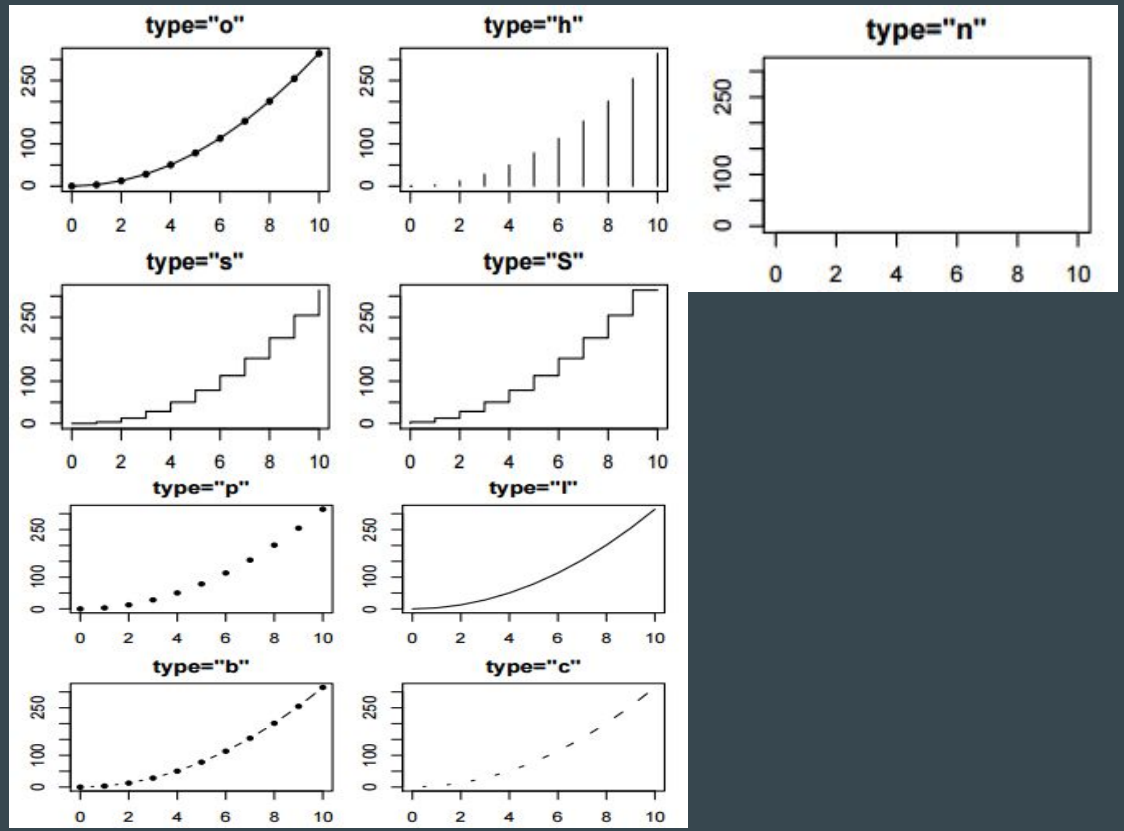
Ejemplo Usando plot()

```
1  radio <- 0:10 # Vector de radios
2  area <- pi*radio^2 # Vector de áreas
3
4  plot(radio, area) # x=radio y=area
```



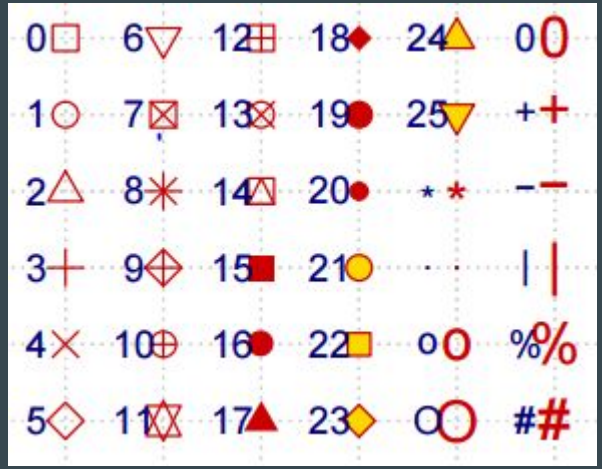
Parámetros para cambiar la apariencia del gráfico (type).

type=	Tipo de gráfico
"p"	puntos (gráfico de dispersión)
"l"	líneas
"b"	ambos (puntos y líneas)
"c"	sólo las líneas de type="b"
"o"	puntos y líneas, sobre-graficados
"h"	agujas (tipo histograma)
"s"	función escalera (horizontal a vertical)
"S"	función escalera (vertical a horizontal)
"n"	no-grafica (sólo se grafican referencias)



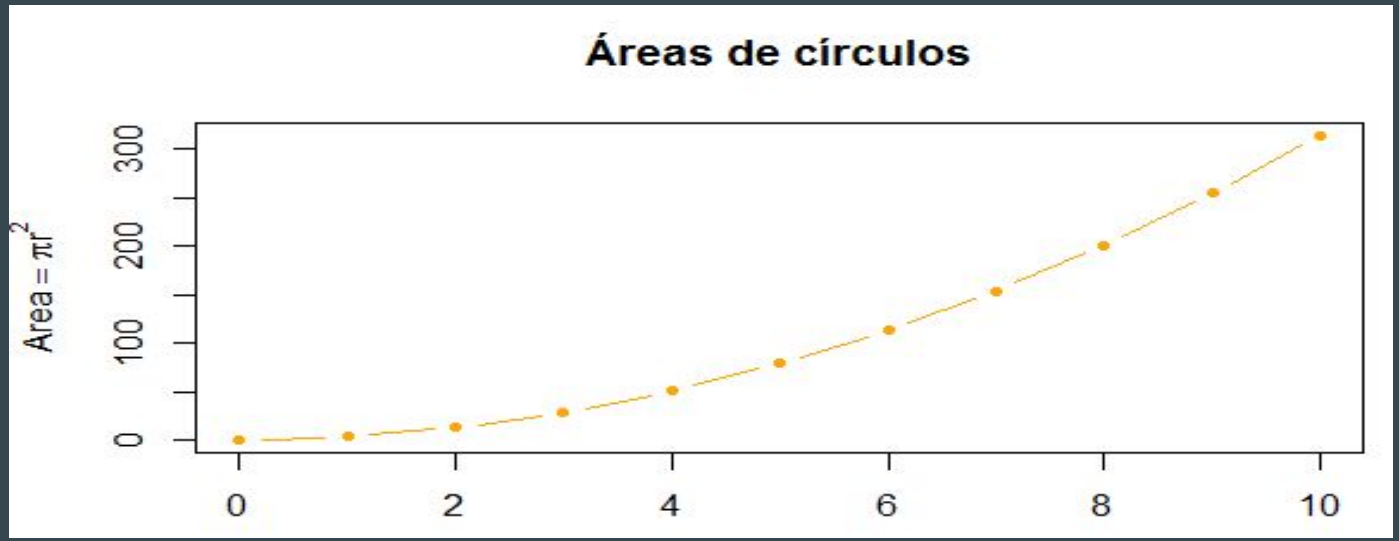
Parámetros para cambiar la apariencia del gráfico.

Parámetro	Uso/Descripción
`main`	Título *principal* del gráfico
`sub`	Subtítulo del gráfico
`xlab`	Etiqueta del eje de las abscisas
`ylab`	Etiqueta del eje de las ordenadas
`asp`	Relación de aspecto del área gráfica (y/x)
`xlim, ylim`	Vectores que indican el rango de valores en los ejes x,y
`log`	Los ejes que serán logarítmicos, p.ej., "", "x", "y", "yx"
`col`	Color
`bg`	Color del *fondo*
`pch`	Símbolos para puntos
`cex`	Para escalado de símbolos de puntos
`lty`	Tipos de línea
`lwd`	Grosor de líneas



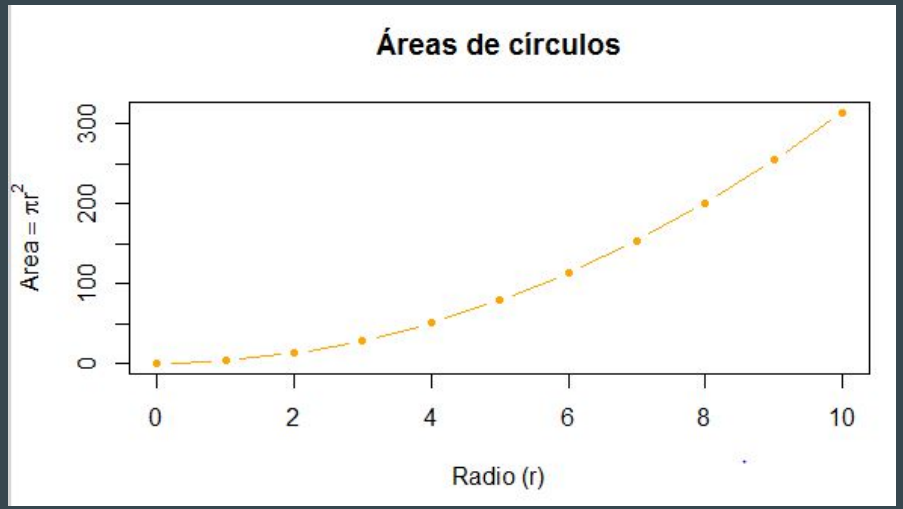
Continuación de Ejemplo Usando plot()

```r
radio <- 0:10 # Vector de radios
area <- pi*radio^2 # Vector de áreas

plot(radio, area, # x=radio y=area
     type="b", # "both", puntos y líneas
     main="Áreas de círculos",
     xlab="Radio (r)",
     ylab=expression(Area == pi*r^2), # Una expresión
     col="orange", # color (naranja)
     pch=20) # tipo de símbolo para punto
```

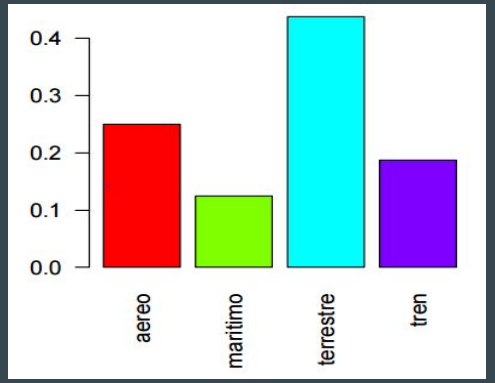


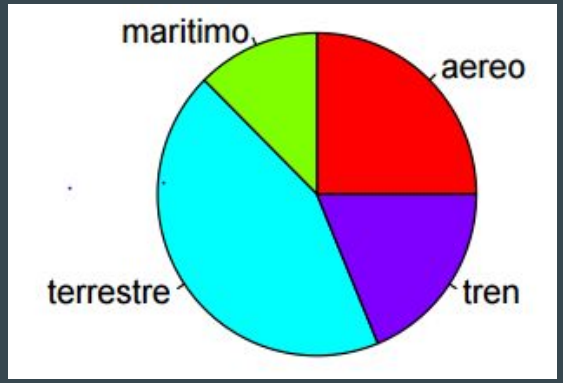
Continuación de Ejemplo Usando plot()

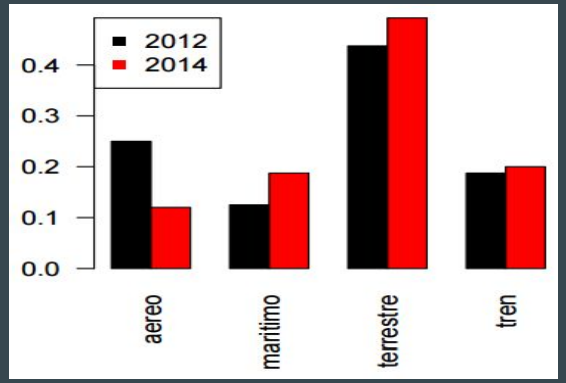
```r
radio <- 0:10 # Vector de radios
area <- pi*radio^2 # Vector de áreas

plot(radio, area, # x=radio y=area
     type="b", # "both", puntos y líneas
     main="Áreas de círculos",
     xlab="Radio (r)",
     ylab=expression(Area == pi*r^2), # Una expresión
     col="orange", # color (naranja)
     pch=20) # tipo de símbolo para punto
```

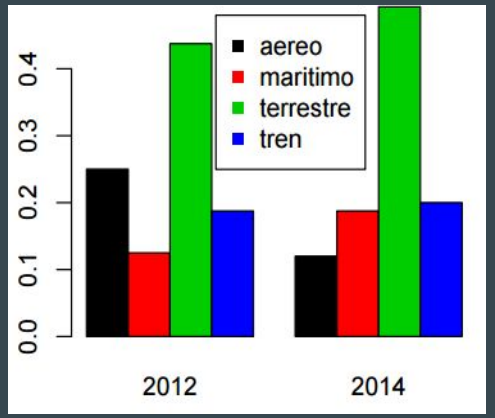


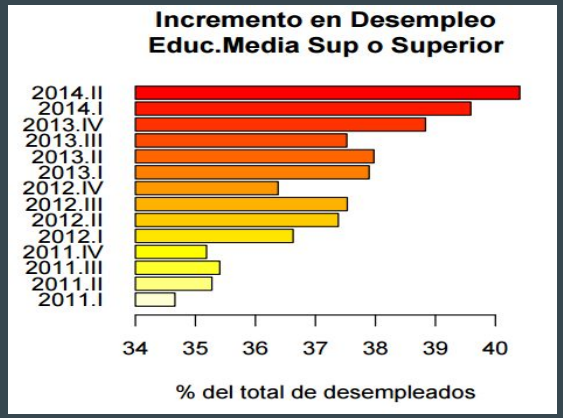
Ejemplos de Gráficas

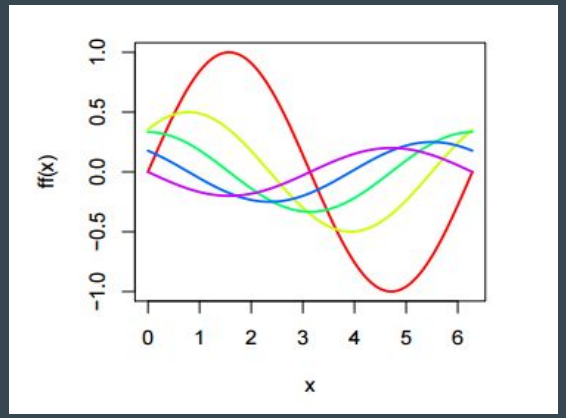




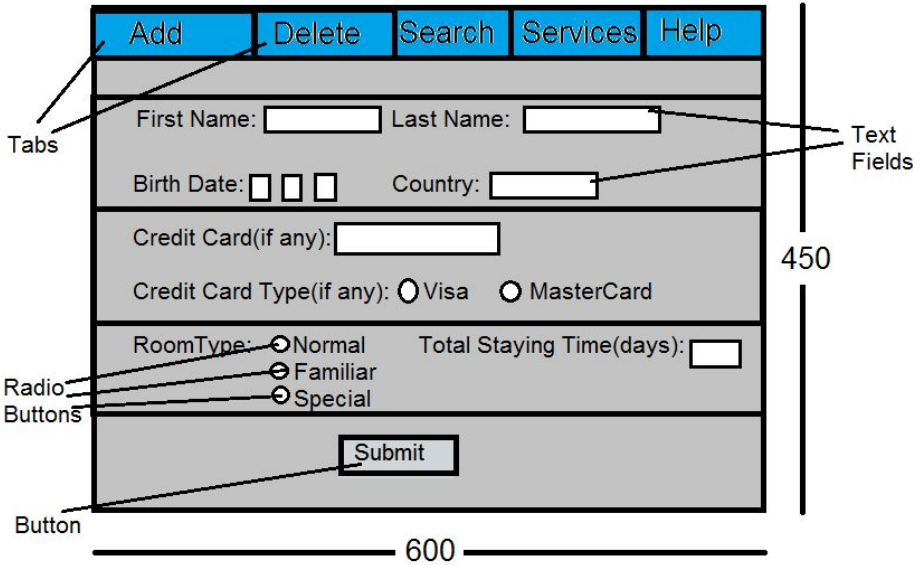








CREACIÓN DE GUIs EN R.



https://cran.r-project.org/web/packages/gWidgets/vignettes/gWidgets.pdf