Análisis de datos en salud

Clase 1. Introducción a R

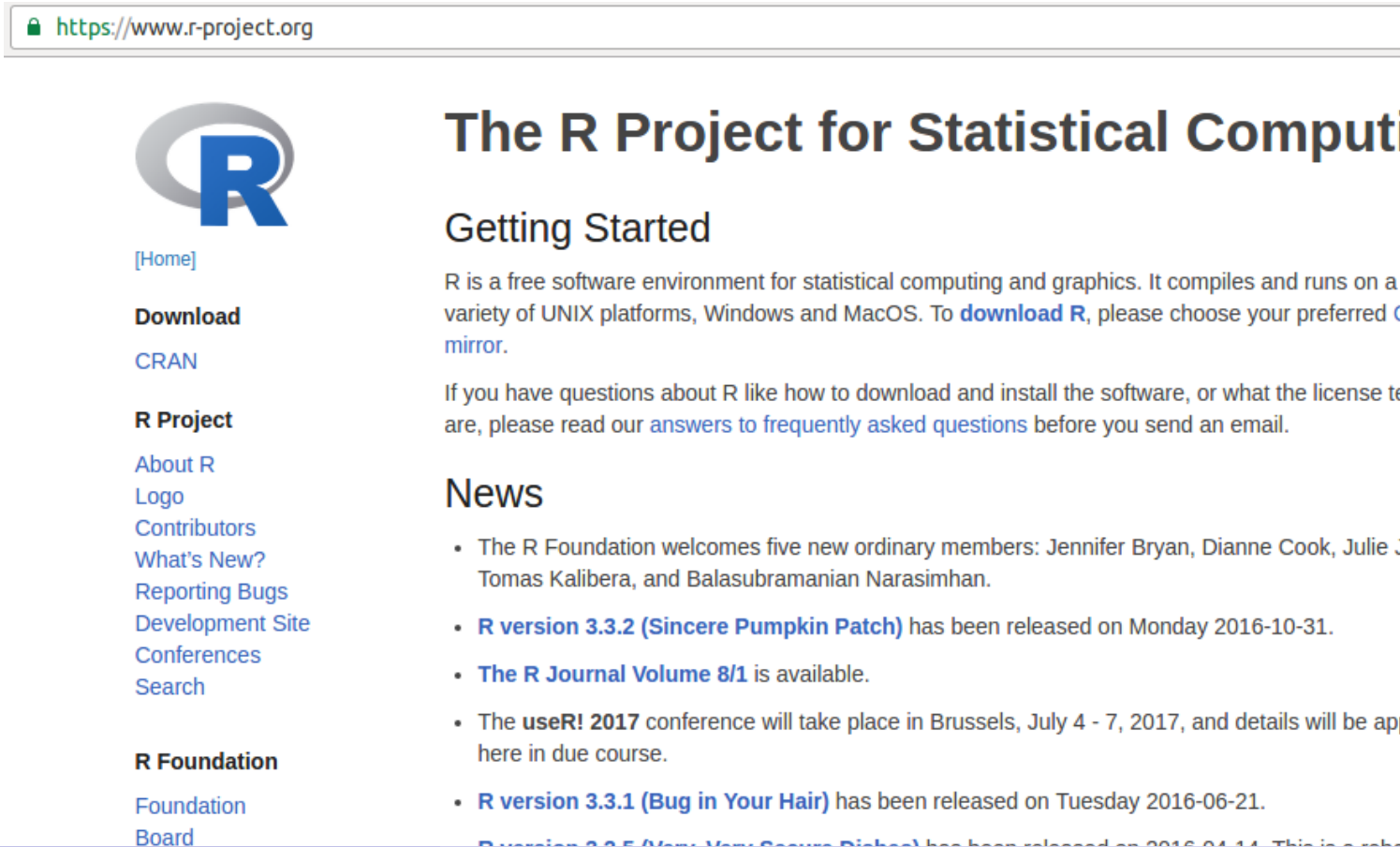
Lic. Estefania Mancini
emancini@leloir.org.ar
mancini.estefania@gmail.com

Buenos Aires, 5 diciembre de 2016

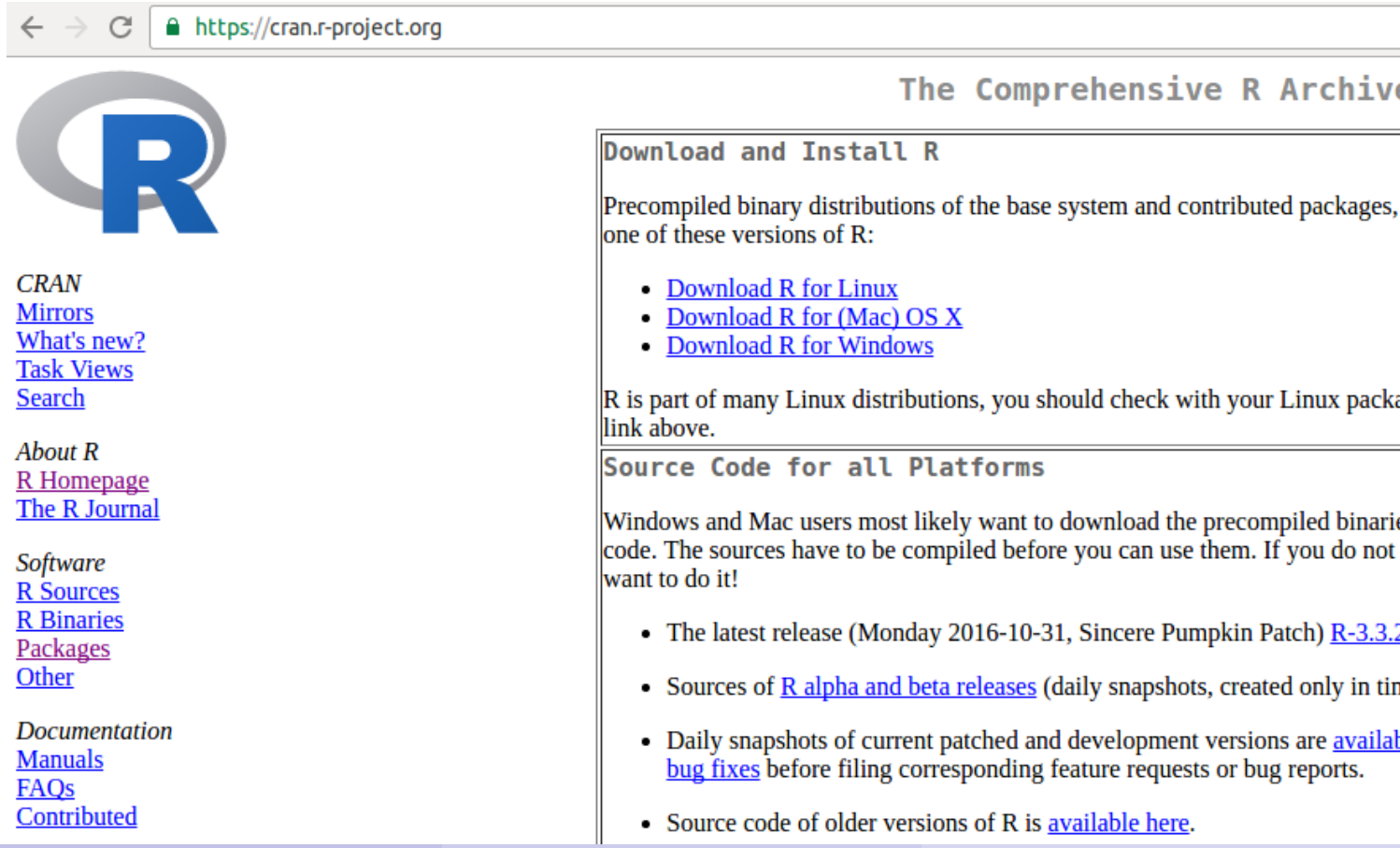
Qué es R?

- **Software de análisis de datos**: análisis estadísticos, visualización, modelado
- **Lenguaje de programación**: de tipo scripting (completo), interactivo, que soporta diferentes paradigmas: estructurado, funcional y orientado a objetos.
- La estructura de datos están diseñadas para permitir la transferencia sencilla de datos entre funciones
- **Entorno para análisis estadístico**: están disponibles funciones para simulaciones, modelos estadísticos, gráficos
- Permite una visualización simple

El proyecto R



El proyecto R: CRAN cran.r-project.org

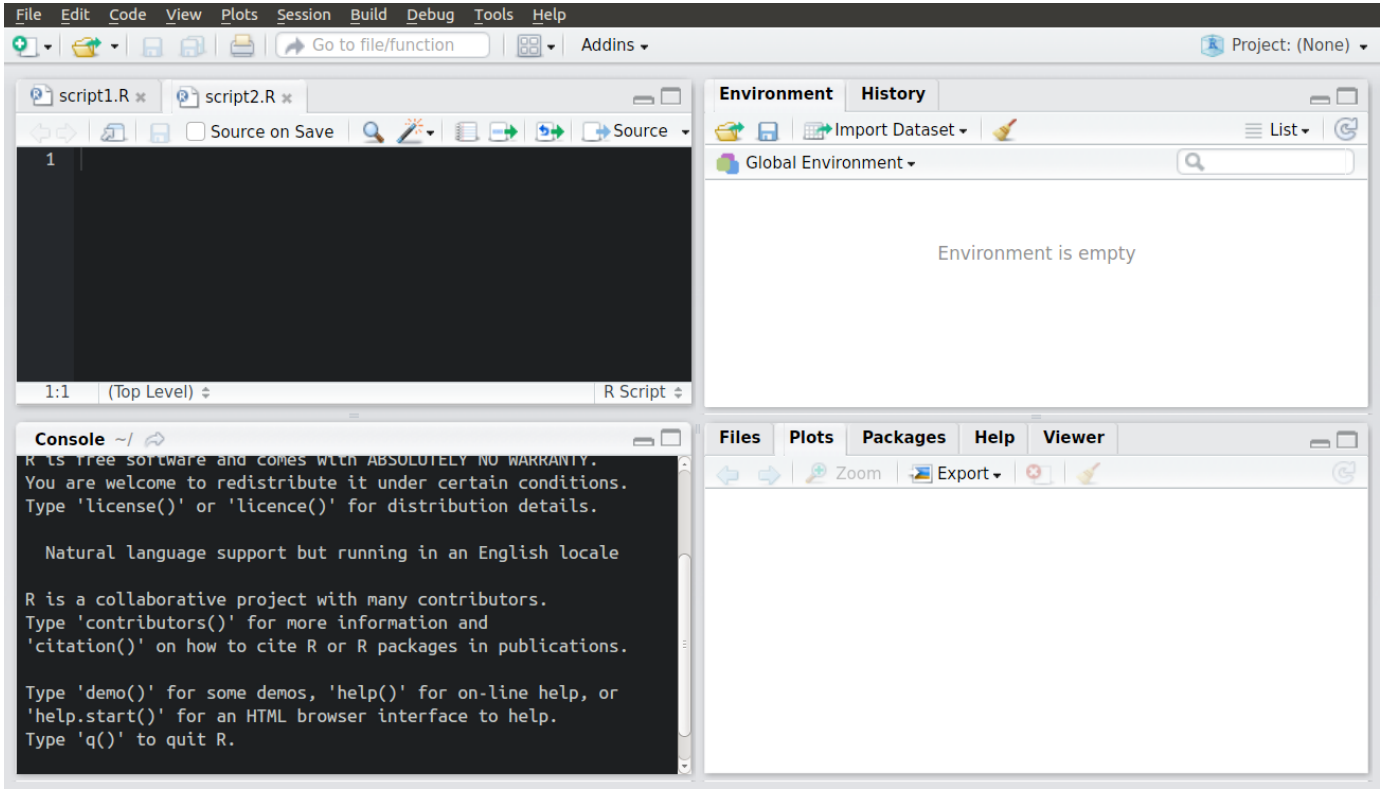


El proyecto R: PROS

- Es flexible:
  - Existen más de 2000 paquetes para extender R en cada dominio de conocimiento
  - Se instala una versión **core** con los paquetes indispensables
  - Luego se puede customizar la instalación con paquetes acordes a nuestras necesidades
- Es GRATUITO
- Multiplataforma: versiones para WINDOWS, LINUX, MAC
- Puede usarse con GUI, terminal (consola), interfaces API
- Es poderoso para I/O data, incluyendo bases de datos
- Buena documentación online
- Colaboradores activos alrededor del mundo
- Numerosos foros de ayuda y discusión

Muchas de estas razones justifican el auge en la comunidad científica

R STUDIO


File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Project: (None)
script1.R
script2.R
Source on Save
Source
1
1:1
(Top Level)
R Script
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Console ~/
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Files
Plots
Packages
Help
Viewer
Zoom
Export

Ingresando comandos

- La interfaz principal permite ingresar texto y es el modo primario de interacción con el software.
- La línea de comandos se simboliza con >
- Luego de ingresar el comando apretar ENTER
- Tiene historial: podemos navegar los comandos anteriores (y guardarlo en un archivo)

```
savehistory()
```

- Tiene capacidad de autocompletar: usar la tecla TAB
- Es sensible a mayúsculas y minúsculas
- Podemos escribir 2 operaciones seguidas separadas de ;

```
obj1<-"a"; obj2<-"b";
```

- para ver el contenido de un objeto, basta con tipearlo.

```
obj1
```

Ayuda

Cuando tengo un problema es fundamental pedir ayuda dando los detalles de la version de R y de los paquetes que estoy usando: Cómo puedo saberlo? Si ya estoy en la sesión de R:

```
sessionInfo()
```

Para saber si existe ayuda de una función (ejemplo plot):

```
?plot, help(plot)
```

Vamos ahí...

sessionInfo

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> sessionInfo()
R version 3.2.3 (2015-12-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=es_AR.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=es_AR.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=es_AR.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=es_AR.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
>
```

Estructura de R

- R-base: núcleo de R formado por `utils, stats, datasets, graphics ...`
- R-contrib: el resto de los paquetes. Se organizan en repositorios:
  - CRAN, Comprehensive R Archive Network
  - Bioconductor (bioconductor.org)
- Instalar paquete de CRAN

  ```
  chooseCRANmirror()
  install.packages(<nombre del paquete>)
  ```

  Cada uno instalará paquetes de acuerdo a sus necesidades

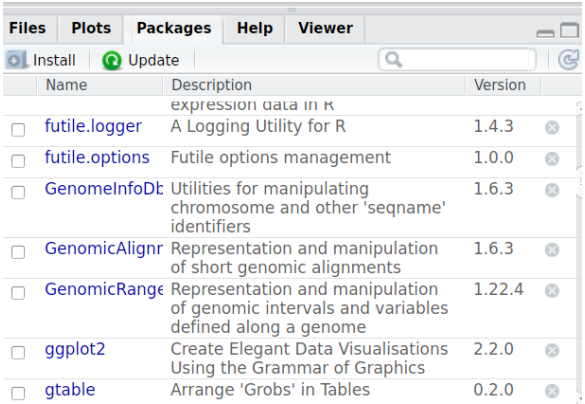
Estructura de R

- Instalar paquetes

  `install.packages(ggplot2)`

- cargar paquetes:

  `library(<nombre del paquete>)`
  `library(ggplot2)`

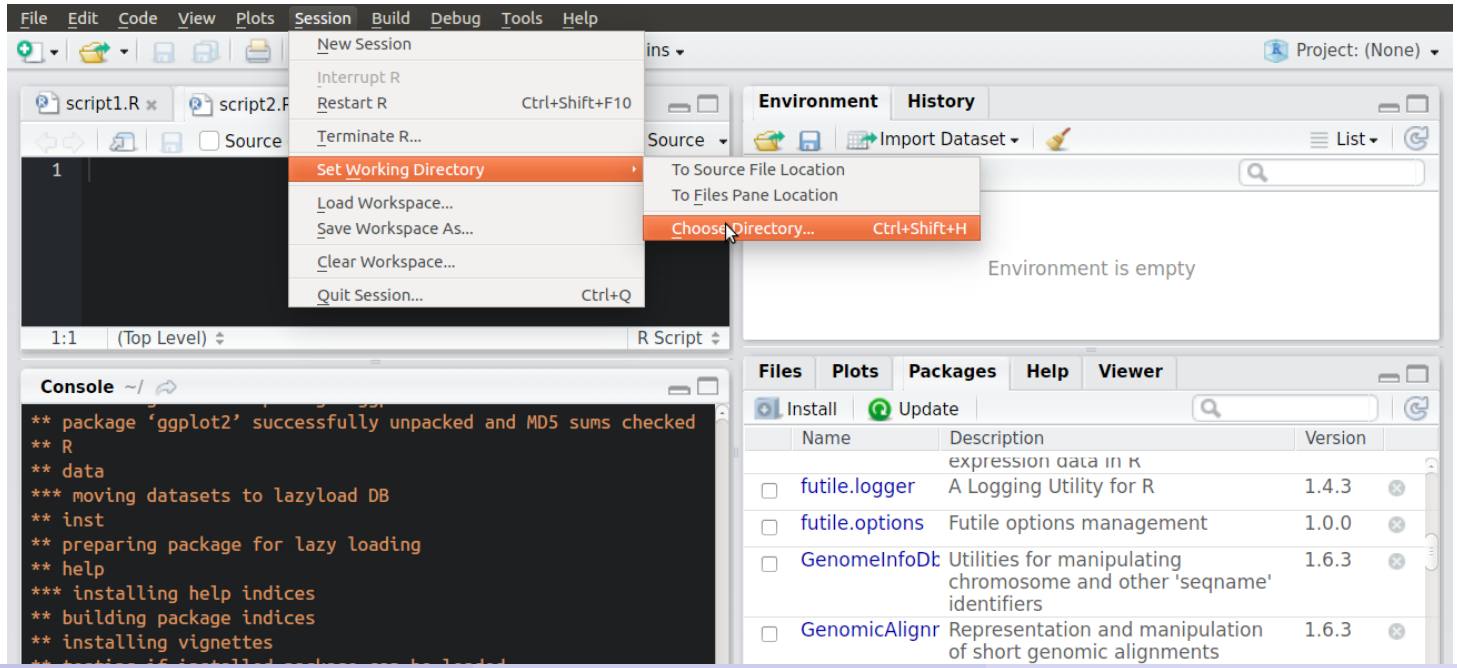


Directorios de trabajo

Para manejar el input/output de datos, es fundamental conocer el directorio de trabajo

Tipeando:

- `getwd()`: Informará la ruta absoluta del directorio de trabajo
- `setwd()`: Cambiará el directorio de trabajo



Iniciar R: Lenguaje orientado a OBJETOS

- Cuando trabajamos en R, las variables, datos, funciones y resultados se guardan en la memoria activa de la computadora en forma de **OBJETOS** que tienen **NOMBRES**
- El usuario puede realizar acciones en esos objetos con operadores: aritméticos, lógicos, comparaciones y funciones

Para asignar valores a los objetos:

```
Obj1<-"nombre"
Obj2<-"apellido"
Obj3<-10
```

Si no asignamos la salida de un comando a un objeto no podremos utilizarlo luego. Si están guardados en objetos, podemos guardarlos en archivos fuera del entorno:

```
save(Obj1, file="Obj1.RData")
load(file="Obj1.RData")
```

Tipos de datos

Hay 4 **Tipos de datos** principales: **NUMÉRICOS, CARACTERES, COMPLEJOS, LÓGICOS**

- Caracteres (character): cadena de caracteres. Cualquier tipo de texto. Se denota con "" o ''

```
var1<- "gen3"
```

- Numérico (numeric): numérico real

```
var2<- 5.1
```

- Entero (integer): (con signo) número entero (cualquier número que no tenga decimales)

```
var3<- -3
```

- Lógico (logical): asume valores binarios TRUE o FALSE

```
var4<- FALSE
```

Tipos de datos

- Los números en R generalmente se tratan como objetos numéricos
- Si se desea especificar un entero (integer), se debe especificar usando el sufijo L

```
var5 <- 10L #es un integer
var6 <-10 #es numeric
```

- Cualquiera sea el tipo de datos, los valores faltantes se representan con "NA" *(not available)*
- Los valores no finitos se representan con Inf, -Inf
- Valores que no son números con "NaN" *(not a number)*

VECTOR: Es una variable unidimensional que aloja elementos del mismo TIPO.

Generación de vectores

- Creamos vectores usando la funcion vector():

```
var1N<-vector(mode="numeric", length=10)
var1C<-vector(mode="character", length=10)
```

- Secuencias regulares numéricas

```
var4<-1:30
var5<-seq(length=9, from=1, to=5)
var6<-seq(1,5,0.5)
var7<-c(2,4,6,8,10)
var8<-rep(1,30)
```

- lo mismo podemos hacer con otros tipos de datos:
  usamos la función c() para concatenar valores en un vector

```
var9<-c("A","B","C")
var10<-rep("gen",10)
```

FACTOR: es un tipo de vector que se usan para representar datos categóricos.

Podemos crear un vector factorial con la función `factor()`, donde `levels` especifica los niveles y el orden del facto y el parámetro `labels` se utiliza para especificar la "etiqueta" a cada nivel:

```
var2<-factor(1:4)
var2<-factor(1:4, levels=1:4)
var2F<-factor(1:5, levels=1:4)
var2F <- factor(LETTERS[3:1], ordered = TRUE)
var3<-factor(c("y","n","y","a","y","y","n","n","n"),
levels=c("y","n"))
var3<-factor(c("y","n","y","a","y","y","n","n","n"),
levels=c("y","n"), labels=c("SI","NO"))
```

Generación de vectores

- Si mezclamos modos de datos en un vector, son coercionadas a un tipo único
- usamos la función c() para concatenar valores en un vector
- usamos la función class() para chequear que clase de objeto resulta:

```
var11<-c("A",2,TRUE); class(var11)
```

- concatenar vectores:

```
NewVar<-c(var8, var9); class(NewVar)
```

- paste

```
nombres<-rep("gen",10)
id<-seq(1:10)
nombresIDa<-paste(nombres, id, sep=".")
```

Funciones para corroborar tipos de datos:

```
varI<-1:10; is.integer(varI)
varL<-rep(c(TRUE, FALSE), each=4); is.logical(varL)
varD<-rep(c("TRUE", "FALSE"), each=4); is.logical(varL)
varC<-"polyA"; is.character(varC)
```

Podemos convertirlos:

```
varI<-as.integer(varI)
varF<-as.factor(varL)
varC<-as.character(varL)
varE<-as.logical(varD)
```

Veamos qué tipo de datos generamos en las variables anteriores

Operadores con vectores

- unique()
- table()
- duplicated()
- summary()
- which()
- is.na()
- length()
- min(); max(); sum()
- mean(); sd(); var()

Apliquemos algunos sobre las variables anteriores
Prestar atención que no todas las funciones se pueden aplicar sobre los vectores

Tipos de objetos para representar datos

MATRIZ: son vectores con un atributo dimensional (dim)

```
m1<-matrix(data=NA, nrow=10, ncol=5, byrow=FALSE); class(m1)
m2<-matrix(1:50, nrow=10, ncol=5, byrow=FALSE); class(m2)
#alternativa para pensar
varM1<-1:15; varM1
dim(varM1)<-c(5,3); varM1
```

Funciones útiles para matrices:

```
tm2<-t(m2);  dim(m2)
m3<- cbind(m2, 1:10); m3
m4<- cbind(m2, LETTERS[1:10]); m4 #nota alguna diferencia?
m5<-rbind(m2, 1:5); m5
```

Tipos de objetos para representar datos

DATAFRAME: es una matriz cuyas columnas pueden ser heterogéneas. Es una tabla que admite con uno o mas vectores de la misma longitud pero pueden ser de diferentes tipos. Es lo más similar a una clśica planilla de cálculo (ej MS Excel)

```
varDf1 <- 1:4; varDf2<-10; varDf3<-c(0,10,35); varDf4<-2:4
df1 <-data.frame(varDf1,varDf2);df1
df2 <-data.frame(varDf3,varDf4); df2
```

Tienen atributos propios:

```
row.names(df1)
colnames(df1)
```

Pueden convertirse a matrices (datos homogéneos):

```
mdf1<-as.matrix(df1); class(mdf1)
```

Construcción de DATAFRAMES

Comenzamos construyendo cuatro vectores para registrar los datos de cinco participantes en una carrera de velocidad.

```
ID <- c(1001,1002,1003,1004,1005)
nombre <- c("Juana", "Ramiro", "Lucas", "Ana", "Emilia")
barrio <- c("Urquiza", "Urquiza", "Belgrano", "Nuñez", "Pueyr
tiempo <- c(16.7, 15.0, 14.3, 19.1, 17.5)
```

Qué tipo de vectores son? Qué funciones podemos aplicarles?

Construcción de DATAFRAMES

Ahora construimos el dataframe y lo exploramos un poco

```
corredores <- data.frame(id=ID, nombre, barrio, tiempo)
corredores
names(corredores)
str(corredores)
dim(corredores)
nrow(corredores); ncol(corredores)
summary(corredores)
```

Tipos de objetos para representar datos

- LISTA: Son conjuntos ordenados de objetos. Puede estar compuesta por elementos de diferentes tipos.

```
L1<-list(varDf1,df1); L2<-list(A=df1, B=df2)
names(L1); names(L2)
```

Tips útiles

Nombres de variables

- Nombres descriptivos
- Se pueden mezclar mayúsculas y minúsculas
- Deben comenzar con letras

scripts

- Podemos ir guardando los comandos en un script, para no perder el trabajo si cerramos la sesión
- Podemos agregar comentarios: usando # al principio de la línea
- Sirven para dejar explicaciones y si compartimos nuestros scripts con otras personas

Accesores: indexar y subsetear

El sistema de indexado es una manera flexible para acceder selectivamente los elementos de un objeto.
Puede ser numérico o lógico, incluyentes o excluyentes.

- Acceso a vectores:

  ```
  x<-1:5; x[3]; x[-3]
  x[3]<-2
  ```

  También puede ser un vector numérico (enteros positivos y/o negativos)

  ```
  i<-c(1,3); x[i]
  ```

- Matrices y dataframes: se accede con 2 posiciones [número filas, número de columnas]:

  ```
  M<-matrix(1:6, 2,3); M
  M[,3]<-21:22; M[,3]
  ```

Accesores

También es posible acceder los valores de un elemento que cumplan una condición:

```
var15<-1:10; var15
var15[var15>=5]<-20; var15
var15[var15==1]<-25; var15
```

Prestar atención los remplazos de valores!

Accesores

Si el objeto que quiero acceder tiene nombres, puedo usarlos para subsetear:

```
df1<-data.frame(matrix(NA, ncol=2, nrow=2))
colnames(df1)<-c("nombre","apellido"); df1
df1$nombre;df1$apellido
rownames(df1)<-c("acc1","acc2")
df1["acc1",]
```

Datasets

En R tenemos un paquete **datasets** instalado por defecto. El mismo contiene numerosos set de datos para usar. Se carga al momento de iniciar la sesión:

```
sessionInfo()
?datasets
library(help="datasets")
```

En nuestro caso, para practicar vamos a utilizar los siguientes: **women, swiss, infert**. Veamos qué contienen:

```
help(women); str(women)
help(swiss); str(swiss)
help(infert); str(infert)
```

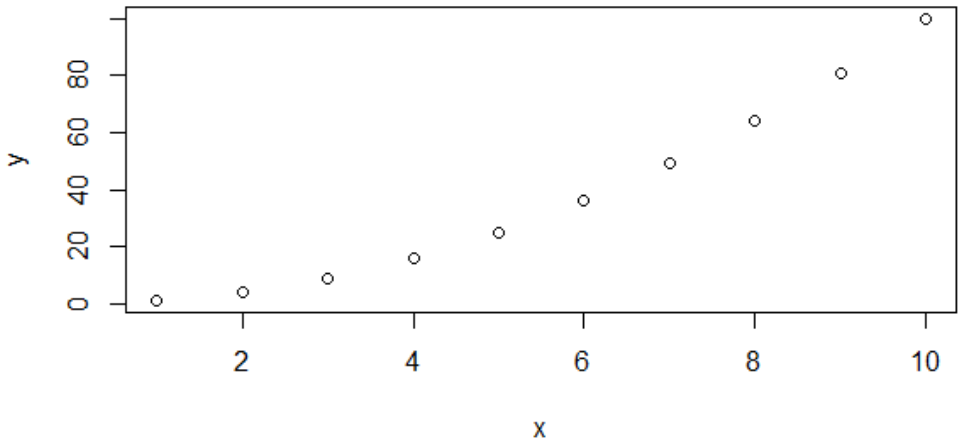
Gráficos

Tenemos el paquete **graphics** instalado por defecto con varias funciones para crear gráficos sencillos y rápidos

```
library(help = "graphics")
```

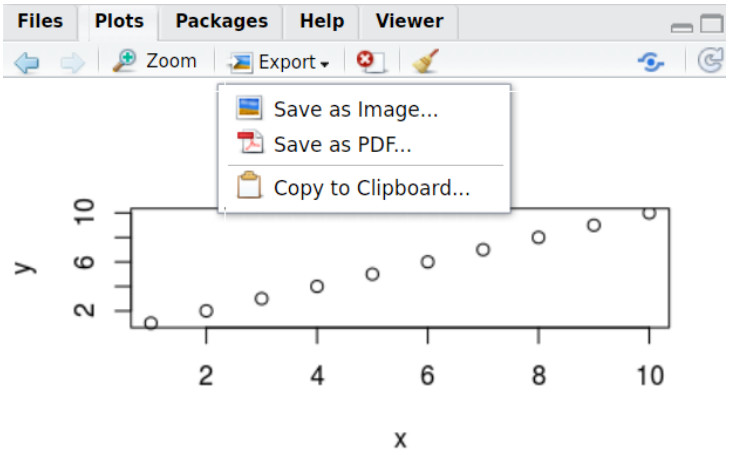
Ejemplos:

```
x<-1:10; y<-x^2;plot(x,y)
```



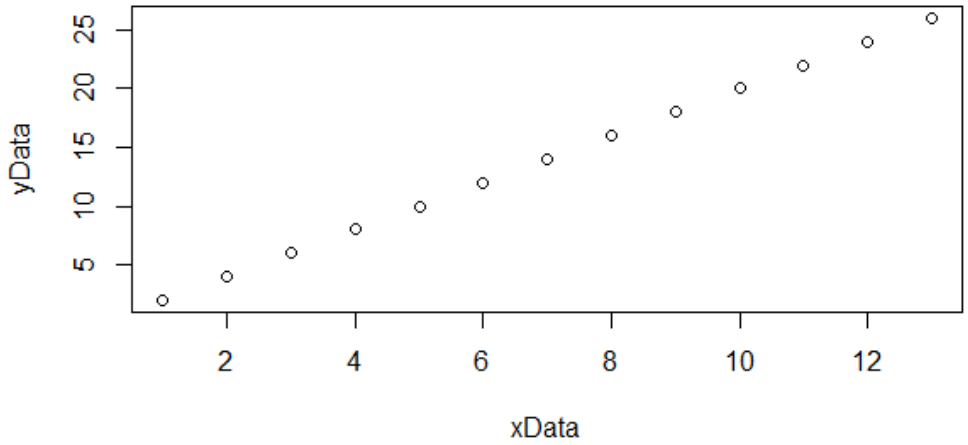
Exportar gráficos

```
x<-1:10
y< x^2
pdf(file="plot1.pdf")
plot(x,y)
dev.off()
png(file="",opt ... )
jpeg(file="",opt ...)
?capabilities
```



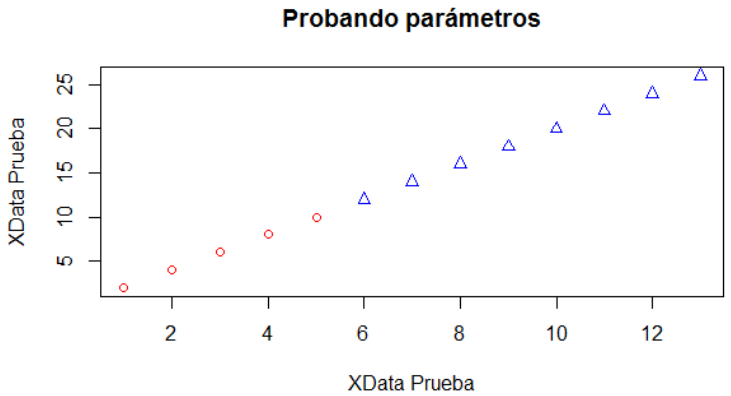
Gráficos

```
xData = seq(1,13)
yData=seq(2,26,2)
plot(xData,yData)
```



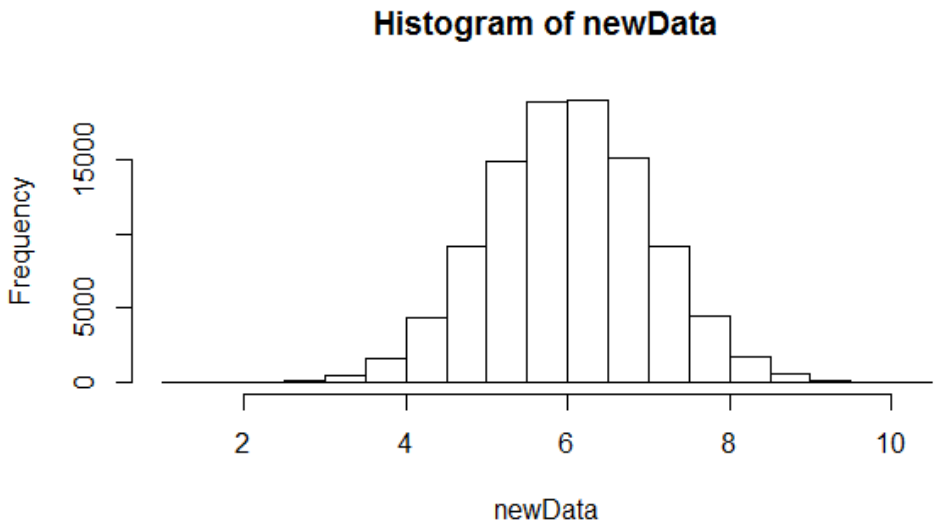
Gráficos

```
xData = seq(1,13); yData=seq(2,26,2)
plot(xData,yData)
color<-rep("red", length(xData))
color[xData>5]<-"blue"
forma<-rep(1, length(xData))
forma[xData>5]<-2
plot(xData,yData, col=color,
pch=forma, main="Probando parametros",
xlab="XData Prueba", ylab="XData Prueba")
```



Gráficos:

```
?rnorm
newData=rnorm(100000,6,1)
hist(newData)
```



Gráficos:

Probemos con los datasets de prueba:

```
plot(women$height, women$weigth)
hist(women$height)
hist(women$weight)
boxplot()
```

Funciones

- Son un tipo de objeto único que puede ser utilizado en el entorno de trabajo.
- Es una manera simple y conveniente de extender nuestra capacidad de trabajo
- Es la forma en que está estructurado el código en los paquetes

Se definen de la siguiente manera:

```
MiFuncion <- function(arg1, arg2, ...) {expresion}
```

Funciones: ejemplo

```
MiFuncion <- function(y1, y2) {
        n1  <- length(y1); n2  <- length(y2)
        s1 <- sum(y1);   s2 <- sum(y2)
        output <- c( s1/n1 , s2/n2)
       return(output)}
```

Luego la función puede ser usada:

```
MiFuncion(arg1, arg2, ...)
```

En nuestro caso:

```
dato1<-1:10
dato2<-2:12
outCalc<-MiFuncion(dato1,dato2)
```

Y devolverá el valor. Luego podemos guardarlas en archivos de extensión .R y cargalas al entorno cuando las necesitemos

```
save(MiFuncion, file="MiFuncion.R")
source("MiFuncion.R")
```

Importar datos:

Es fundamental conocer el directorio de trabajo

```
getwd(); setwd()
```

Tenemos varias opciones para leer datos en R:

- Archivos tabulares / delimitados por caracteres personalizados (son convertidos a dataframes)

```
myTable<-read.table()
myCsv<-read.csv()
readLines()
read.delim()
```

- Funciones / sccripts de extensión `.R`:

```
source()
```

- WorkSpaces / objetos de R (`.rda, .RData, .Rhistory`)

```
load(); loadHistory(); data()
```

Leer

`file`	the name of the file (within `""` or a variable of mode character), possibly with its path (the symbol \ is not allowed and must be replaced by /, even under Windows), or a remote access to a file of type URL (http://...)
`header`	a logical (`FALSE` or `TRUE`) indicating if the file contains the names of the variables on its first line
`sep`	the field separator used in the file, for instance `sep="\t"` if it is a tabulation
`quote`	the characters used to cite the variables of mode character
`dec`	the character used for the decimal point
`row.names`	a vector with the names of the lines which can be either a vector of mode character, or the number (or the name) of a variable of the file (by default: `1`, `2`, `3`, ...)
`col.names`	a vector with the names of the variables (by default: `V1`, `V2`, `V3`, ...)
`as.is`	controls the conversion of character variables as factors (if `FALSE`) or keeps them as characters (`TRUE`); `as.is` can be a logical, numeric or character vector specifying the variables to be kept as character
`na.strings`	the value given to missing data (converted as `NA`)
`colClasses`	a vector of mode character giving the classes to attribute to the columns
`nrows`	the maximum number of lines to read (negative values are ignored)
`skip`	the number of lines to be skipped before reading the data
`check.names`	if `TRUE`, checks that the variable names are valid for R
`fill`	if `TRUE` and all lines do not have the same number of variables, "blanks" are added
`strip.white`	(conditional to `sep`) if `TRUE`, deletes extra spaces before and after the character variables
`blank.lines.skip`	if `TRUE` ignores "blank" lines

Para exportar datos:

- Archivos tabulares / delimitados por caracteres personalizados:

```
write.table()
write.csv()
```

- Funciones de extension .R:

```
save()
 savehistory()
 saveRDS();
```

Imprimir archivos

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"))
```

`x`	the name of the object to be written
`file`	the name of the file (by default the object is displayed on the screen)
`append`	if `TRUE` adds the data without erasing those possibly existing in the file
`quote`	a logical or a numeric vector: if `TRUE` the variables of mode character and the factors are written within `""`, otherwise the numeric vector indicates the numbers of the variables to write within `""` (in both cases the names of the variables are written within `""` but not if `quote = FALSE`)
`sep`	the field separator used in the file
`eol`	the character to be used at the end of each line (`"\n"` is a carriage-return)
`na`	the character to be used for missing data
`dec`	the character used for the decimal point
`row.names`	a logical indicating whether the names of the lines are written in the file
`col.names`	id. for the names of the columns
`qmethod`	specifies, if `quote=TRUE`, how double quotes `"` included in variables of mode character are treated: if `"escape"` (or `"e"`, the default) each `"` is replaced by `\"`, if `"d"` each `"` is replaced by `""`

Leer e imprimir archivos

Podemos guardar el dataframe **corredores** en un archivo delimitado por comas (csv) o tabulaciones (tab). Luego podemos abrirlos en una planilla de cálculos:

```
write.csv(corredores, file="corredores.csv")
write.table(corredores, file="corredores.tab", sep="\t")
```

Para leerlo en el entorno, usaremos las funciones:

```
corr.2 <- read.csv("corredores.csv")
corr.2 <- read.table("corredores.tab")
```

También podemos salvarlo como un archivo de R:

```
saveRDS(corredores, "corredores.rds")
```

Tarea para el miércoles

- Descargar la tabla de la sección siguiente página (Sección Summary, Download):

  `http://mldata.org/repository/data/viewslug/datasets-uci-breast-cancer/`
- Guardarla en el directorio de trabajo
- Leerla en el entorno de R y guardarla en un dataframe con el nombre `uci`
- Explorar el dataframe con los comandos aprendidos: `dim`, `class`, `names`, `str`, etc...
- Guardar los comandos ejecutados en un script R

Estructuras de control

IF

Recorre un objeto y ejecuta una acción cuando se satisface la condición:

```
var <-150
if( var >= 100) {print("TRUE")} else {print("FALSE")}
```

FOR

Recorre todos los objetos y ejecuta una acción:

```
x <- 20:1; for(i in seq_len(length(x))){print(x[i])}
letras <- c("a","b","c","d")
for (letra in letras) {print(letra)}

LETRAS<-LETTERS[1:5]
for (letra in LETRAS) {print(letra)}
```