SOFTWARE PARA ANÁLISIS DE DATOS R

Por Ing. Wilson Castro Z.

R está disponible de forma gratuita y se puede descargar desde el Comprehensive R Archive Network (CRAN) en su sitio web en http://www.r-project.org/. Los análisis se llevan a cabo

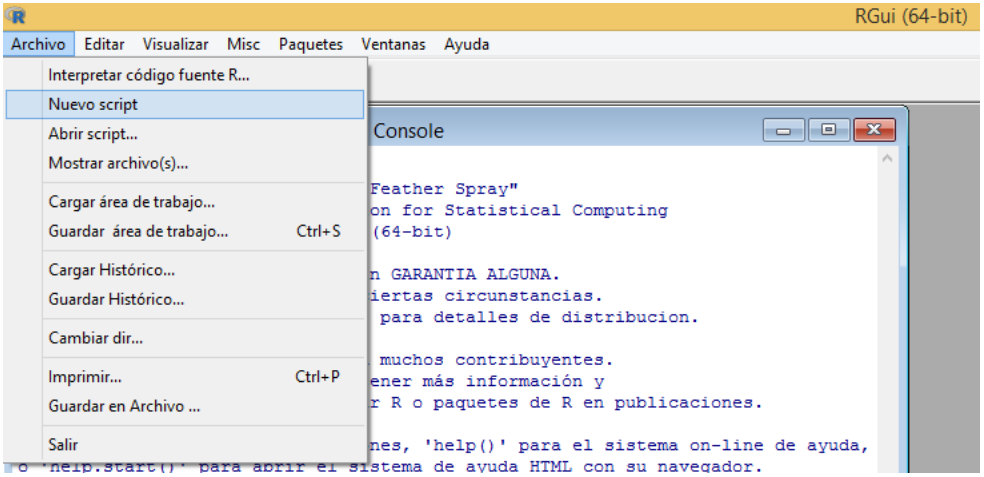
en R aplicando funciones en datos de R (almacenados como objetos R).

Las funciones de R se almacenan en **paquetes**. Solo cuando un paquete está cargado, su contenido está disponible. Los paquetes básicos se instalan cuando instala R.

Los paquetes adicionales deben instalarse por separado.

Una vez que abra R, verá un mensaje: Escriba 1+1 y presione Enter. Con suerte verá la respuesta 2 devuelta en la siguiente línea. Alternativamente, puede escribir comandos en un

script haciendo clic en Archivo/ Nuevo Script.



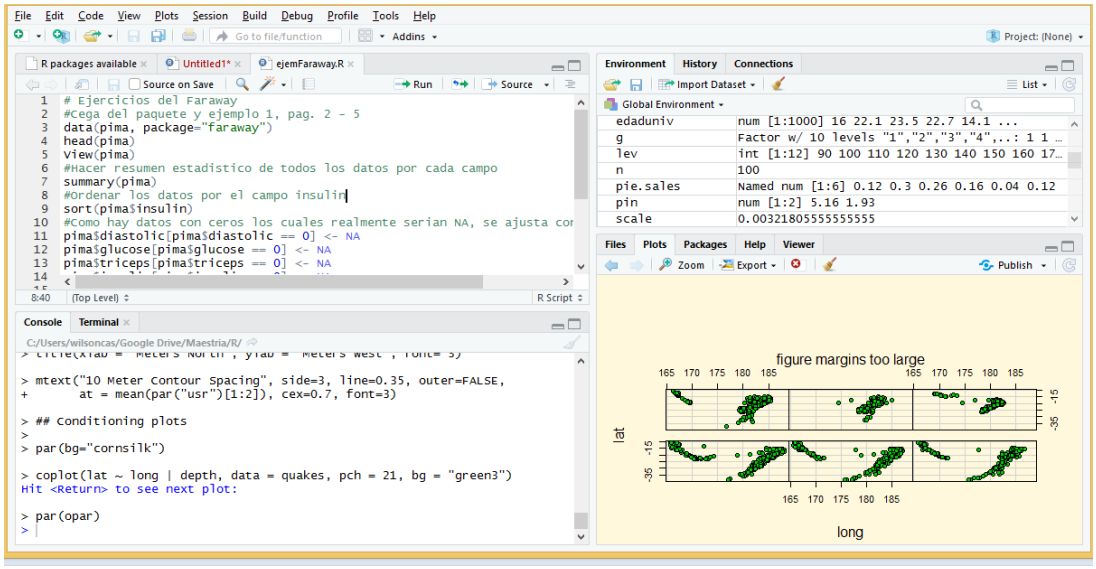
Una vez instale R, también puede instalar un programa que facilite el manejo mediante una interfaz gráfica muy amigable y con una gran disponibilidad de herramientas. Dos excelentes aplicaciones son RStudio y RCmander. Para RStudio, se puede descargar desde su sitio en https://www.rstudio.com/ y el segundo en https://www.rcommander.com/

RSTUDIO

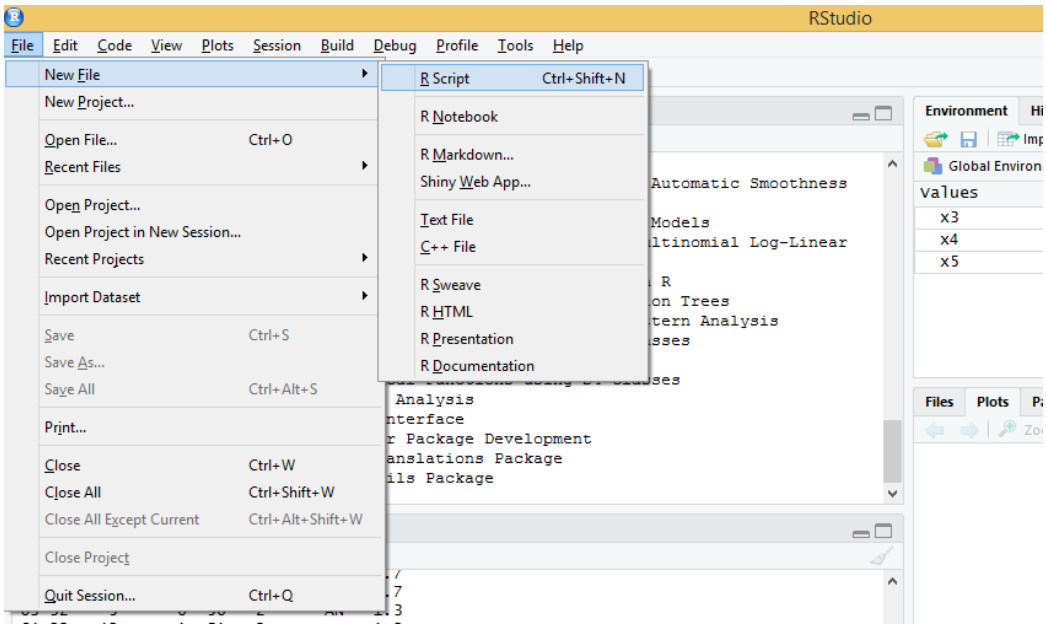
Al instalar RStudio, puede observar en la interfaz gráfica varios paneles como se muestra en la próxima figura.

En el panel superior izquierdo, se pueden crear R scripts, R notebooks y R markdown, este último muy útil para crear documentos especializados como artículos de investigación.

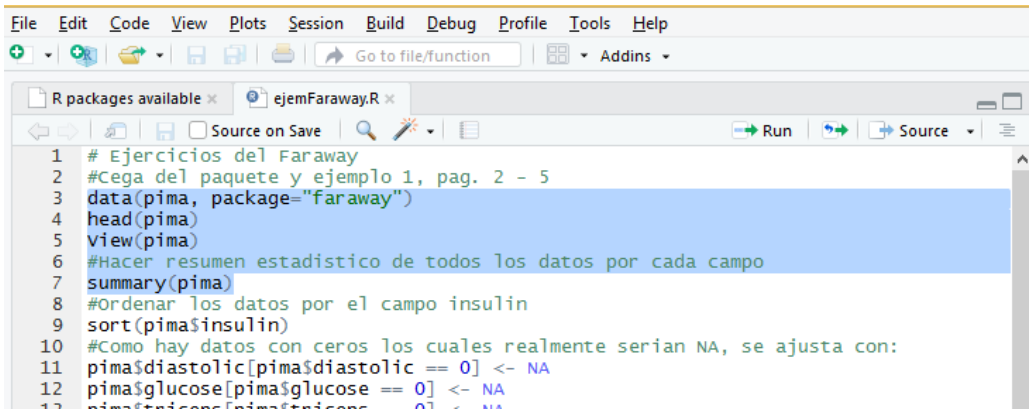
El panel inferior es donde se encuentra la consola, que es donde se despliegan los resultados de análisis y de cálculos numéricos o resultados de llamar funciones como **median**(). En los paneles de la derecha, en el superior se tiene el historial con las variables utilizadas o creadas así como los datasets abiertos, entre otras utilidades. En el panel inferior se muestran gráficos generados, los paquetes o librerías en uso o disponibles, la ayuda y otros archivos.



En RStudio, iniciar un nuevo Script será como se muestra en la siguiente figura:



Con sólo R o RStudio, se abrirá una nueva ventana de script. Al escribir comandos en esta ventana, puede enviar lotes de código a la vez resaltando el código y haciendo clic en Run o presionando las teclas Ctrl+Enter.



Al iniciar a trabajar con R y RStudio, es necesario en primer lugar grabar el **código** que usted escriba. Cuando abra un nuevo documento (Click en File/New file/R Script) como se ilustró arriba, pero para iniciar a trabajar necesita configurar su directorio de trabajo (working directory), lo cual se realiza con setwd(). Recuerde que el símbolo de número es para comentarios:

```
#Iniciar a trabajar con RStudio

Rm(list=ls())  #quita los objetos de un ambiente específico

setwd("C:/Users/fulano/Documentos/R")
```

o puede desde RStudio dar click en la pestaña del menú Session*/Set Working directory/Choose Directory…*

Ejercicio 1.

1. En el panel de la Consola (o zona de líneas de comandos) digite ***demo(graphics)***. Siga las indicaciones y observe los gráficos y código que se requiere para generarlos. Podría preferir ampliar las ventanas.
2. En el panel de la Consola (o zona de líneas de comandos) digite ***demo(image).*** Esta es una demostración o representación de datos en 3D.
3. Obtenga ayuda de la función quit digitando ***?q***.
4. Salga de R digitando ***q().***

R tiene una amplia variedad de tipos de datos como escalares, vectores, matrices, datos lógicos, data frames y listas. Se pueden combinar estos mediante el uso de estas funciones:

- c(objecto1, objecto2, …) # combina objetos en un vector.
- cbind(objecto1, objecto2, …) # Combina objetos como columnas
- rbind(objecto1, objecto2, …) # Combina objetos como renglones.

ESTRUCTURAS DE DATOS EN R

Antes de discutir los análisis específicos en R, puede ser útil dar una breve descripción de algunas de las formas en que se almacenan los datos en R. En particular, describimos cuatro clases de almacenamiento de datos: Vectores, matrices, dataframes (marcos de datos) y listas.

VECTORES

Si escribe e ingresa el código a continuación, R creará un vector numérico con cinco elementos:

c (1,7,12,6,3)

La función **c** combina sus argumentos para formar un vector.

Ejemplo. Operaciones entre vectores:

Cree los vectores u=(1,3,5) y v=(2,4,6)

Realice las operaciones:

a) u + v b) v – u c) 2u + 3v
d) u*v (multiplicación elemento a elemento)
e) Producto escalar entre u y v

Solución:

```
a)
   u<-c(1,3,5)
   v<-c(2,4,6)
   u+v
   [1]  3  7 11
b)
    v-u
   [1] 1 1 1

c) 2*u+3*v
   [1]  8 18 28

d)  u*v
   [1]  2 12 30


  e)

  u %*% v  #Producto escalar o punto de vectores
     [,1]
[1,]   44
```

Así, podemos almacenar un vector como un objeto bajo el nombre (identificador) x1, como:

x1 = c (1,7,12,6,3)

Escriba x1 y presiona enter, y verá el vector x1.

Impreso como salida. El código y la salida se muestran a continuación:

```
x1

1 7 12 6 3
```

Podemos identificar elementos del vector x1 colocando corchetes [] después de x1. Por ejemplo x1[2] identificará al 2do. elemento de x1. El código x1[1: 3] identificará los primeros tres elementos de x1 y el código x1[x1> 6] identificará los elementos en x1 mayor que 6. El código y la salida para estos tres ejemplos son:

```
x1 [2]

7

x1 [1: 3]

1  7  12

x1 [x1> 6]

7  12
```

El operador **:** (usado en el segundo ejemplo) crea una secuencia de enteros incrementados en 1. A continuación, creamos otros cuatro vectores llamados x2, x3, x4 y x5:

```
x2 = 2 * (1: 5)

x3 = 2 * x1 + x2

x4 = x1>6

x5 = c ("azul", "verde", "rojo", "verde", "púrpura")
```

El código x2 = 2 * (1: 5) crea el vector 2, 4, 6, 8 y 10 que llamamos x2. El vector x3 resulta de operaciones aritméticas de x1 y x2. Los vectores x1, x2 y x3 son todos numéricos. Si aplica la función **mode** en x1 (es decir, escriba **mode (x1)** y presione Enter), devuelve la palabra "numeric" como salida. El vector x4 es un vector lógico. La función **mode** devolverá la palabra "logical" si envía el comando **mode(x4)**. Los elementos de un vector de modo lógico son "VERDADERO" o "FALSO".

Ingrese el código **x4** para desplegar este vector (observe la salida abajo):

```
x4

FALSO VERDADERO VERDADERO FALSO FALSO
```

Los elementos segundo y tercero de x4 son VERDADEROS porque estos son mayores que 6. El vector x5 es un vector de caracteres. R distingue entre mayúsculas y minúsculas, por lo que nombrar el vector x5 no es lo mismo que nombrarlo X5.

MATRICES

Podemos crear una **matriz numérica** (llamada **y**) usando los vectores x1, x2 y x3 como columnas de la matriz aplicando la función **cbind** (de column bind -unir columnas-):

```
y = cbind (x1, x2, x3)
```

Ingrese el código **class(y)** y se devolverá la palabra "matrix" como salida. Ingrese **mode(y)** y se devolverá la palabra "numeric" ya que **y** es una matriz numérica. No se pueden mezclar vectores numéricos y de caracteres en una matriz.

Digite **y** seguido de presionar **Enter**, y la matriz se imprimirá (como se muestra a continuación):

y

```
     x1 x2 x3
[1,]  1  2  4
[2,]  7  4 18
[3,] 12  6 30
[4,]  6  8 20
[5,]  3 10 16
```

DATAFRAMES

Un **dataframe** (marco de datos) proporciona una clase más general de almacenamiento de datos que una matriz en R porque un dataframe puede contener una mezcla de variables numéricas, de caracteres y lógicas. Un dataframe en R es similar al dataset (conjunto de datos) de otros programas de análisis de datos como Stata, SAS o SPSS, ya que puede almacenar diferentes tipos de variables. La función **data.frame** se puede utilizar para combinar vectores y matrices de la siguiente manera:

z = data.frame (x1, x2, x3, x4, x5)

o, equivalentemente,

z = data.frame (y, x4, x5)

Escriba **z** y pulse Intro, y se imprimirá el marco de datos (mostrado a continuación):

```
> z = data.frame (x1, x2, x3, x4, x5)
> z
  x1 x2 x3    x4      x5
1  1  2  4 FALSE    azul
2  7  4 18  TRUE   verde
3 12  6 30  TRUE    rojo
4  6  8 20 FALSE   verde
5  3 10 16 FALSE púrpura
```

Los corchetes [] se pueden usar para acceder a filas específicas y/o columnas de un dataframe (marco de datos) o de una matriz. Introduzca el código: **z [2,5]** y la segunda fila, quinta columna se imprimirá desde la matriz **z** (el elemento "green" en este ejemplo). Si quiere acceder a las tres primeras filas (observaciones) de la quinta columna, escriba el código

z [1: 3,5]

o, de manera equivalente,

z [c (1,2,3), 5].

Si desea acceder a la quinta columna completa, ingrese **z [, 5]**. Alternativamente, ya que la 5ta columna (o variable) es nombrada **x5**, puede acceder a la quinta columna completa ingresando el código **z$x5**. La **$** en este ejemplo apunta a la variable llamado **x5** desde el marco de datos o dataframe llamado **z**.

LISTAS

Una **lista** ofrece un tipo más general de almacenamiento de datos que el vector, la matriz o el dataframe (marco de datos), y puede incluir cualquiera de esos **objetos de datos** como parte de la **lista**. El siguiente código crea una lista llamada **w** que contiene un vector de caracteres de

longitud 2 como su primer elemento, el vector **x1** como su segundo elemento, la matriz **y** como su tercer elemento, y el **dataframe z** como su cuarto elemento:

```
> w = list(c("hola", "hasta pronto"),x1,y,z)
> w
[[1]]
[1] "hola"         "hasta pronto"

[[2]]
[1]  1  7 12  6  3

[[3]]
     x1 x2 x3
[1,]  1  2  4
[2,]  7  4 18
[3,] 12  6 30
[4,]  6  8 20
[5,]  3 10 16

[[4]]
  x1 x2 x3    x4      x5
1  1  2  4 FALSE    azul
2  7  4 18  TRUE   verde
3 12  6 30  TRUE    rojo
4  6  8 20 FALSE   verde
5  3 10 16 FALSE púrpura
```

Los corchetes dobles [[]] se pueden utilizar para acceder a elementos particulares de una **lista**. Si quiere acceder al dataframe **z** desde la lista, ingrese el código **w [[4]]** ya que **z** es el cuarto elemento de **w**. Si desea acceder a la primera fila de la tercera columna del cuarto elemento de **w** de la lista, ingrese el siguiente código:

w[[4]] [1,3]

Así la 1ra. fila de la 3ra. columna del 4to. elemento (que es **z**) de **w** tiene el valor 4.

```
> w[[4]] [1,3]
[1] 4
```

Y si se quiere que aparezca “Rojo”:

```
> w[[4]] [3,5]
[1] rojo
Levels: azul púrpura rojo verde
```

Observe que de paso, R da los niveles de esta columna.

PAQUETES, LIBRERÍAS Y DATOS

Los ***paquetes*** (***packages*** en Inglés) son colecciones de funciones de R, datos y código compilado en un formato bien definido. Los directorios donde se almacenan los paquetes se llaman ***librerías*** (***library***).

Para ver qué paquetes están instalados en su programa R, escriba y dé Enter con el comando

library(). Para ejecutar muchas de las funciones necesarias para realiza un análisis específico, por ejemplo de supervivencia en Bioestadística, necesitará instalar el paquete correspondiente (por ejemplo el paquete ***survivival***).

Para instalar un paquete especifico (como el de supervivencia), directamente en R haga clic en Paquetes/ Instalar Paquete. Verá un encabezado llamado espejo CRAN con una lista de muchos países diferentes bajo ese título. Haga clic en uno de estos (por ejemplo, EE. UU. (AZ)) y luego desplácese abajo y haga clic en el paquete de su interés (por ejemplo Survivial) y luego haga clic en Aceptar. El paquete con sus muchas funciones específicas ahora debe ser instalado. En RStudio el proceso es más directo. Escriba en la consola:

>library(nombre_paquete)

Por ejemplo library(survival) y presione enter, y el paquete estará listo para su uso. Como prueba, si instaló el paquete survival, escriba la palabra kidney y pulsa enter. Un conjunto de datos o dataset llamado kidney (que es parte del paquete Survival) debe imprimirse en su pantalla.

Una vez que el paquete de su interés está instalado, no tiene que reinstalarlo en cada sesión. Sin embargo, tendrá que escribir ***library (nombre_paquete)*** cada sesión antes de ejecutar las funciones de ese paquete o librería.

Ejercicios. Describa la incorporación del conjunto de datos IRIS en la lbrería MASS. Que variables hay en este conjunto de datos? ¿Cuántos casos hay en este conjunto de datos? ¿Cuántas variables?

Para cada variable, identifique su tipo de datos (por ejemplo, categórico, discreto). ¿Hay NA en el conjunto de datos?

```
## How to incorporate the IRIS dataset in MASS:
library() ## list of packages. Open the list in other tab.
require(mass) ## load mass package

## Loading required package: mass

data() ## list the avalaible data sets. Open a list in other tab.
data(iris) ## loads specificied data sets

## What information do I have?

length(iris)## number of variables

## [1] 5

names(iris) ##names of variables

## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"

head(iris) ## the first few rows from a data frame

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

tail(iris) ## the last few rows from a data frame
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica

str(iris)## internal structure of an R object

## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
1 1 1 1 1 1 1 ...

ls() ##list of current elements

## [1] "A"    "B"    "C"    "iris" "n"    "s"

## Type of variables
class (iris)

## [1] "data.frame"

class (iris$Species)

## [1] "factor"
```

Utilice otros comandos como:

names(iris)

summary(names)

Datos NO existententes o NA. Al analizar un dataset o conjunto de datos, se debe tener cuidado con los datos inexistentes que se representan con NA o en algunos casos simplemente aparecen vacíos. En el siguiente código observe el manejo de estos.

```
## Create an invented data frame
a = c(2, 3, 5, 4)
s = c("aa", "bb", "cc", NA)
b = c(TRUE, FALSE, TRUE, FALSE)
df = data.frame(a,s,b)       # df is a data frame

## NA?
table(is.na(df))

##
## FALSE  TRUE
##    11     1

table (is.na (df))

##
## FALSE  TRUE
##    11     1

##List rows of data that have missing values
df[complete.cases(df),]

##   a  s     b
## 1 2 aa  TRUE
## 2 3 bb FALSE
## 3 5 cc  TRUE

## Create a dataset without missing data
dfnew=na.omit(df)
dfnew ## in this case there are no missing data

##   a  s     b
## 1 2 aa  TRUE
## 2 3 bb FALSE
## 3 5 cc  TRUE
```

SELECCIONAR SUBCONJUNTOS DE DATOS DE UN DATASET

A menudo es útil extraer los individuos (casos) de un conjunto de datos que tienen características específicas. Se logra esto a través de comandos con condicionales. Considere este ejemplo con el famoso dataset iris que contiene información de plantas:

```
attach(iris)
iris
iris$Species=="virginica"
```

Estos comandos producen una serie lógica TRUE y FALSE, que indica si una planta es ***virginica*** o no. Es importante escribirlos exactamente según los conjuntos de datos.

Supongamos que queremos extraer solo los datos de las especies VIRGINICA en la muestra. Podemos usar la función de ***subset*** de R para hacerlo. Por ejemplo, el comando:

mdata <- subset(iris, iris$Species == 'virginica')

este comando crea nuevos datos en ***mdata*** que contiene información solamente de ***Virginica.***

Ejemplos.

De la especie VIRGINICA, tome los datos de longitud de pétalo entre 5.2 y 5.5:

mdata=subset(iris,iris$Species=="virginica")

subset(mdata,mdata$Petal.Length<=5.5 & mdata$Petal.Length>=5.2) #Trae entre 5.2 y 5.5.

Cree un nuevo objeto llamado ***Setosa*** que contenga todas las observaciones sobre las especies de Setosa que tienen longitud del pétalo según:

a) Longitud de pétalo > 6.0.

Setosa<-subset(iris,iris$Species=="setosa" & iris$Petal.Length>6)

El retultado es vacío porque no hay datos con estas características, como se puede observar al hacer un View(iris).

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa

Showing 1 to 21 of 150 entries

Se realizan otras consultas como <1.6 ya que valores >6 no existen para Setosa.

Setosa<-subset(iris,iris$Species=="setosa" & iris$Petal.Length>1.5)

Setosa2<-subset(iris,iris$Species=="setosa" & iris$Petal.Length<1.3)

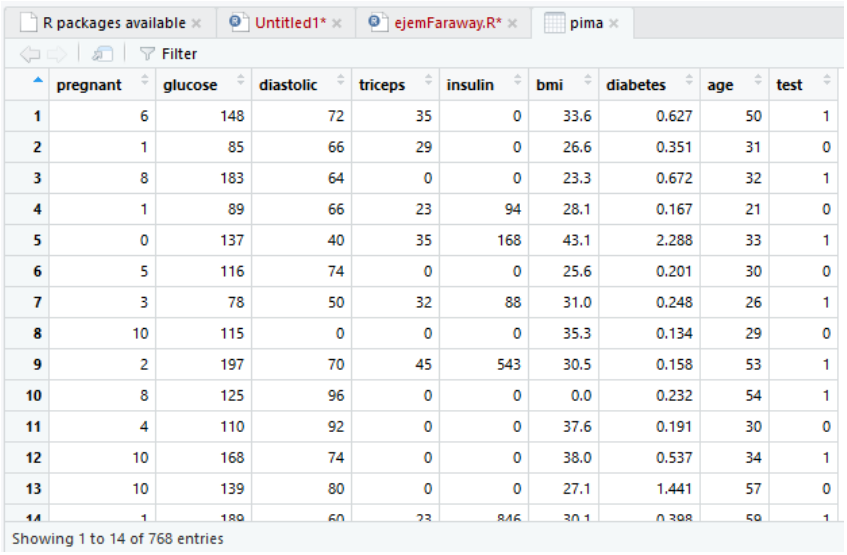
Ahora observe otro ejemplo en que aparecen los datos vacío con cero, y esto obviamente afecta el resumen de estadísticos (summary()). Luego se deben “arreglar estos datos con NA en vez de cero. Debe instalar la librería faraway si quiere correr el ejemplo.

Ejercicios del Faraway

data(pima, package="faraway")

head(pima)

View(pima) #Muestra los datos del package pima (Se muestra la imagen a continuación):



R packages available | Untitled1* | ejemFaraway.R* | pima

Filter

	pregnant	glucose	diastolic	triceps	insulin	bmi	diabetes	age	test
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31.0	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0.0	0.232	54	1
11	4	110	92	0	0	37.6	0.191	30	0
12	10	168	74	0	0	38.0	0.537	34	1
13	10	139	80	0	0	27.1	1.441	57	0
14	1	189	60	23	846	30.1	0.398	59	1

Showing 1 to 14 of 768 entries

#Hacer resumen estadístico de todos los datos por cada campo

summary(pima) #Se muestra el resultado:

```
> summary(pima)
    pregnant         glucose        diastolic        triceps         insulin           bmi
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00   Min.   :  0.0   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00   1st Qu.:  0.0   1st Qu.:27.30
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00   Median : 30.5   Median :32.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54   Mean   : 79.8   Mean   :31.99
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00   3rd Qu.:127.2   3rd Qu.:36.60
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.0   Max.   :67.10
    diabetes           age             test
 Min.   :0.0780   Min.   :21.00   Min.   :0.000
 1st Qu.:0.2437   1st Qu.:24.00   1st Qu.:0.000
 Median :0.3725   Median :29.00   Median :0.000
 Mean   :0.4719   Mean   :33.24   Mean   :0.349
 3rd Qu.:0.6262   3rd Qu.:41.00   3rd Qu.:1.000
 Max.   :2.4200   Max.   :81.00   Max.   :1.000
```

#Ordenar los datos por el campo insulin

sort(pima$insulin)

#Como hay datos con ceros los cuales realmente serían NA, se corrige con:

pima$diastolic[pima$diastolic == 0] <- NA

pima$glucose[pima$glucose == 0] <- NA

pima$triceps[pima$triceps == 0] <- NA

pima$insulin[pima$insulin == 0] <- NA

pima$bmi[pima$bmi == 0] <- NA

#Verificar nuevamente con

sort(pima$insulin)

#Y al hacer de nuevo el summary:

summary(pima) #Cambiaron los estadisticos de los campos que se 'arreglaron'

```
> summary(pima) #Cambiaron los estadisticos de los campos que se 'arreglaron'
    pregnant         glucose        diastolic        triceps         insulin           bmi
 Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00   Min.   : 14.00   Min.   :18.20
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:22.00   1st Qu.: 76.25   1st Qu.:27.50
 Median : 3.000   Median :117.0   Median : 72.00   Median :29.00   Median :125.00   Median :32.30
 Mean   : 3.845   Mean   :121.7   Mean   : 72.41   Mean   :29.15   Mean   :155.55   Mean   :32.46
 3rd Qu.: 6.000   3rd Qu.:141.0   3rd Qu.: 80.00   3rd Qu.:36.00   3rd Qu.:190.00   3rd Qu.:36.60
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.00   Max.   :67.10
                  NA's   :5       NA's   :35       NA's   :227     NA's   :374      NA's   :11
    diabetes           age             test
 Min.   :0.0780   Min.   :21.00   Min.   :0.000
 1st Qu.:0.2437   1st Qu.:24.00   1st Qu.:0.000
 Median :0.3725   Median :29.00   Median :0.000
 Mean   :0.4719   Mean   :33.24   Mean   :0.349
 3rd Qu.:0.6262   3rd Qu.:41.00   3rd Qu.:1.000
 Max.   :2.4200   Max.   :81.00   Max.   :1.000
```

Observe como los estadísticos reales ahora son muy diferentes. Esto es algo que el ***científico de datos*** debe hacer con frecuencia.

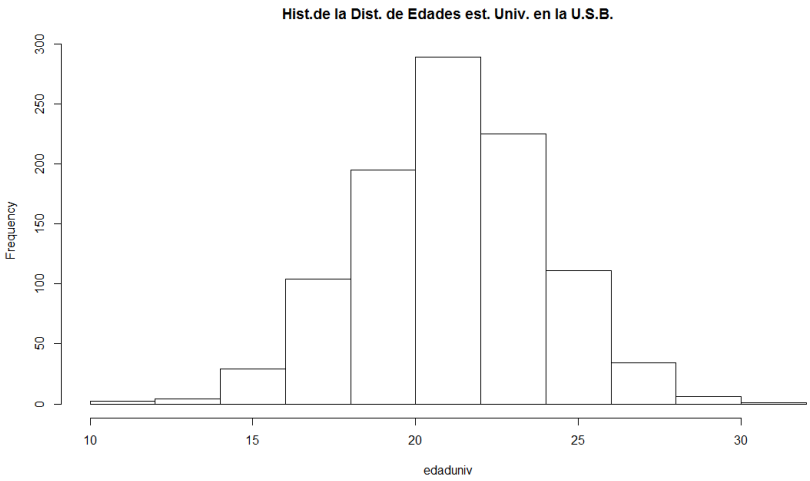
GRAFICOS

El paquete o librería básica de R contiene funciones para graficar como ***plot(), hist(), boxplot(), barplot()***, etc. Para abrir la ayuda en R sobre cualquiera de estas funciones, con lo cual se pueden observer los parámetros requeridos, use ***?nombre_funcion***, como en ***?barplot***

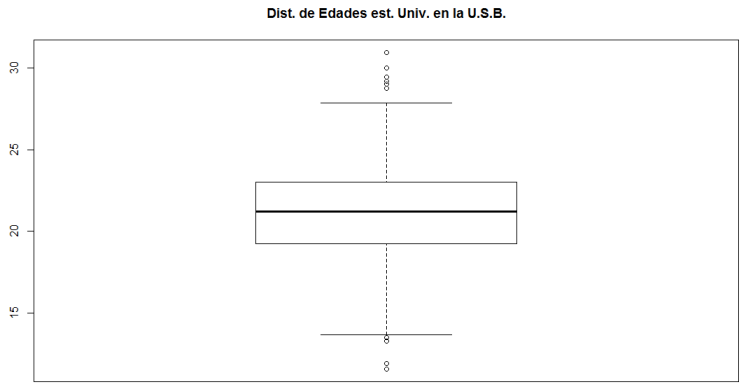
Cree un vector simulando 1000 números aleatorios dados por la distribución normal con media μ= 21 y desviación estándar σ = 3, que bien podría representar las edades de 1000 estudiantes universitarios de la U.S.B., y luego cree gráficos como el histograma de estos datos creados:

edaduniv<-rnorm(1000,mean=21,sd=3)

hist(edaduniv,main="Hist.de la Dist. de Edades est. Univ. en la U.S.B.")

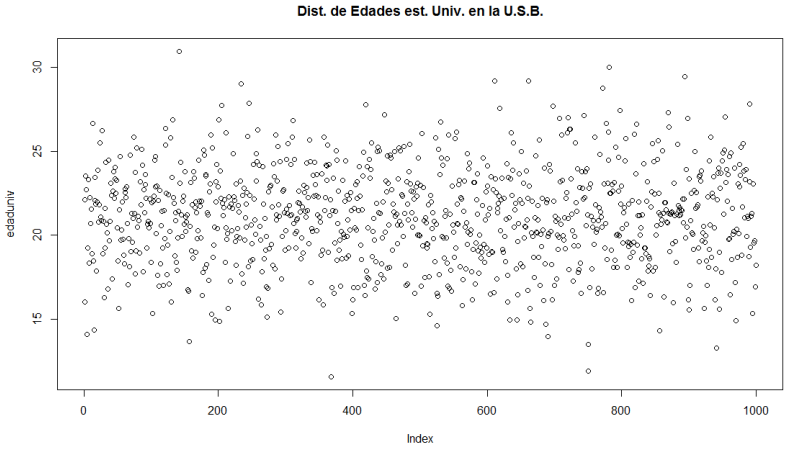


boxplot(edaduniv,main="Dist. de Edades est. Univ. en la U.S.B.")



Ahora si se grafica con plot, se genera un gráfico de dispersión, pues se grafica cada dato:

plot(edaduniv,main="Dist. de Edades est. Univ. en la U.S.B.")



Ahora el barplot o gráfica de barras, se la aplicamos a la matriz **z** de los vectores **x1, x2** y **x3** según:

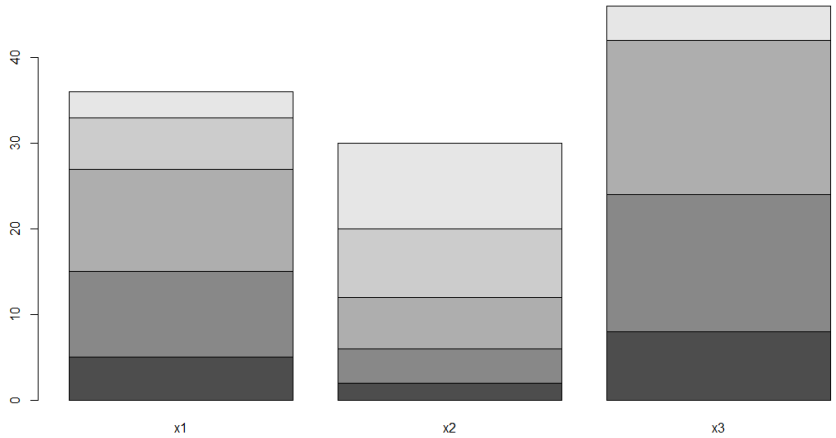
x1 = c (5,10,12,6,3)

x2 = 2 * (1: 5)

x3 = 2 * x1 - x2

y = cbind (x1, x2, x3)

```
barplot(y)
```



Observe que cada franja horizontal de toda columna representa al elemento del vector correspondiente.

IMPORTAR DATOS EN R

En la actualidad existen muchas fuentes de datos de libre descarga y disposición, de personas e investigadores y de entidades privadas y públicas. En Colombia hay gran cantidad de datos y de información en https://www.datos.gov.co muchos se pueden descargar como .csv

Se pueden importar diferentes tipos de datos en R de acuerdo a su formato y extensiones. El archivo con los datos debe estar en la carpeta de trabajo (en la que estableció con el comando ***setwd()***) para mayor facilidad, o especificar la ruta completa como se observa en los ejemplos que siguen. Entre los tipos de archivos están:

a) De un archivo de texto:
mydata=read.table("c:/temp/mydata.txt",dec=",",header=TRUE).

b) **De un archivo de texto delimitado por comas o .csv**, En algunos casos se debe probar de una u otra forma:

mydata <- read.table("c:/temp/mydata.csv", header=TRUE, sep=",",row.names="id")

mydata=read.csv2("c:/temp/mydata.csv", dec=",").

c) De una hoja de Excel. Si tiene datos en Excel, puede si es posible, dejar una sola hoja y Guardar como archivo de extensión .csv y abrirlo en R según se indicó para estos archivos. De lo contrario necesita cargar el paquete xslx :

```
library(xlsx)
mydata <- read.xlsx("c:/myexcel.xlsx", 1) #Datos de la hoja 1
mydata <- read.xlsx("c:/myexcel.xlsx", sheetName = "mysheet") #Datos de la hoja 'mysheet'
```

d) De programas comerciales como *SPSS (".sav") / STATA(".dta") / SAS(".sas7bda")*:

```
install.packages ("foreign")
require ("foreign")
mydata<-read.spss("c:/mydata.sav")
mydata<-read.dta ("c:/mydata.dta")
```