Procesamiento de datos con R

Clase: Introducción a R

MSc. Orlando Joaqui Barandica

Universidad del Valle

2023

Contenido

Contenido

Estructuras de control

Las estructuras de control establecen condicionales en nuestros código. Por ejemplo, qué condiciones deben cumplirse para realizar una operación o qué debe ocurrir para ejecutar una función.

Esto es de gran utilidad para determinar la lógica y el orden en que ocurren las operaciones, en especial al definir funciones.
Las estructuras de control más usadas en R son las siguientes.

Estructura de control	Descripción
`if` , `else`	Si, de otro modo
`for`	Para cada uno en
`while`	Mientras
`break`	Interrupción
`next`	Siguiente

Contenido

If, else

- if (si) es usado cuando deseamos que una operación se ejecute únicamente cuando una condición se cumple.
- else (de otro modo) es usado para indicarle a R qué hacer en caso de la condición de un if no se cumpla.

Un if es la manera de decirle a R:

SI esta condición es cierta, **ENTONCES** haz estas operaciones.

```
if (condición) sentencia
ó
if (condición) sentencia1 else sentencia2
```

Ejemplos

```
if ( 3>2 ) {
  print( "Verde" )
}
```

Ejemplos

```
if ( 3>2 ) {
  print( "Verde" )
}
```

```
[1] ‘‘Verde’’
```

Ejemplos

```
if ( 3>2 ) {
  print( "Verde" )
}
```

```
[1] ‘‘Verde’’
```

```
if ( 1>5 ) {
  print( "Verde" )
} else {
  print( "Rojo" )
}
```

Ejemplos

```
if ( 3>2 ) {
  print( "Verde" )
}
```

```
[1] ''Verde''
```

```
if ( 1>5 ) {
  print( "Verde" )
} else {
  print( "Rojo" )
}
```

```
[1] ''Rojo''
```

Ejemplos

```
x <- 3

  if(x >2 ){
  y <- 3*x
  } else {
  y <- 0}

y
```

Ejemplos

```
x <- 3

  if(x >2 ){
  y <- 3*x
  } else {
  y <- 0}

y
```

```
[1] 9
```

Ejemplos

```
x <- 3

  if(x >2 ){
  y <- 3*x
  } else {
  y <- 0}

y
```

```
[1] 9
```

Equivalente a lo anterior:

```
ifelse(x>2 , 3*x ,0)
```

Ejemplos

```
x <- 3

  if(x >2 ){
  y <- 3*x
  } else {
  y <- 0}

y
```

```
[1] 9
```

Equivalente a lo anterior:

```
ifelse(x>2 , 3*x ,0)
```

```
[1] 9
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

```
[1] ‘‘Verdadero’’
Warning message:
In if (1:10 < 3) { :
  the condition has length > 1 and only the first element will be used
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

```
[1] ‘‘Verdadero’’
Warning message:
In if (1:10 < 3) { :
  the condition has length > 1 and only the first element will be used
```

```
ifelse(1:10 < 3, TRUE, FALSE)
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

```
[1] ‘‘Verdadero’’
Warning message:
In if (1:10 < 3) { :
  the condition has length > 1 and only the first element will be used
```

```
ifelse(1:10 < 3, TRUE, FALSE)
```

```
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10] FALSE
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

```
[1] ‘‘Verdadero’’
Warning message:
In if (1:10 < 3) { :
  the condition has length > 1 and only the first element will be used
```

```
ifelse(1:10 < 3, TRUE, FALSE)
```

```
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10] FALSE
```

```
ifelse(1:10 < 3, "AMERICA", "CALI")
```

Ejemplos

```
if(1:10 < 3) {
  "Verdadero"
}
```

```
[1] ‘‘Verdadero’’
Warning message:
In if (1:10 < 3) { :
  the condition has length > 1 and only the first element will be used
```

```
ifelse(1:10 < 3, TRUE, FALSE)
```

```
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10] FALSE
```

```
ifelse(1:10 < 3, "AMERICA", "CALI")
```

```
[1] ‘‘AMÉRICA’’ ‘‘AMÉRICA’’ ‘‘CALI’’    ‘‘CALI’’    ‘‘CALI’’
[6] ‘‘CALI’’    ‘‘CALI’’    ‘‘CALI’’    ‘‘CALI’’    ‘‘CALI’’
```

Contenido

for()

La estructura for nos permite ejecutar un bucle (loop), realizando una operación para cada elemento de un conjunto de datos.

Su estructura es la siguiente:

```
for(elemento in objeto) {
  operacion_con_elemento
}
```

for()

La estructura for nos permite ejecutar un bucle (loop), realizando una operación para cada elemento de un conjunto de datos.

Su estructura es la siguiente:

```
for(elemento in objeto) {
  operacion_con_elemento
}
```

Con lo anterior le decimos a R:

- **PARA** cada elemento **EN** un objeto (inicio:fin), haz la siguiente operación.
- Al elemento lo podemos llamar como nosotros deseemos.
- El objeto debe ser el nombre de un objeto existente.
- Tradicionalmente se usa la letra **i** para denotar al elemento

Ejemplos

```
for (i in 1:5){
i^2
}
```

Ejemplos

```
for (i in 1:5){
i^2
}
```

¿Y cuál es la salida?

Ejemplos

```
for (i in 1:5){
i^2
}
```

¿Y cuál es la salida?

```
for (i in 1:5){
print(i^2)
}
```

Ejemplos

```
for (i in 1:5){
i^2
}
```

¿Y cuál es la salida?

```
for (i in 1:5){
print(i^2)
}
```

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

Ejemplos

Podemos asignar cada valor resultante de nuestras operaciones a una posición específica en un vector, incluso si este está vacio.

```
x<-0

for (i in 1:5){
x[i]<-i^2
}

x
```

Ejemplos

Podemos asignar cada valor resultante de nuestras operaciones a una posición específica en un vector, incluso si este está vacio.

```
x<-0

for (i in 1:5){
x[i]<-i^2
}

x
```

```
[1]  1  4  9 16 25
```

Ejercicio

Si se tiene el vector **y <- c(3,3,3,5,5,5,8,8,8)**, codifique los valores de **y** iguales a 5 como "a" y los diferentes como "b".

Utilice un ciclo for() y condicionales

Ejercicio

Si se tiene el vector **y <- c(3,3,3,5,5,5,8,8,8)**, codifique los valores de **y** iguales a 5 como "a" y los diferentes como "b".

Utilice un ciclo for() y condicionales

```
y <-c(3,3,3,5,5,5,8,8,8)

for(j in 1:length(y)){
        if (y[j]==5){
           y[j]<-"a"
        }else{
             y[j]<-"b"}
      }
y
```

Ejercicio

Si se tiene el vector **y <- c(3,3,3,5,5,5,8,8,8)**, codifique los valores de **y** iguales a 5 como "a" y los diferentes como "b".

Utilice un ciclo for() y condicionales

```
y <-c(3,3,3,5,5,5,8,8,8)

for(j in 1:length(y)){
        if (y[j]==5){
           y[j]<-"a"
        }else{
             y[j]<-"b"}
      }
y
```

```
[1] "b" "b" "b" "a" "a" "a" "b" "b" "b"
```

Contenido

while()

Este es un tipo de bucle que ocurre mientras una condición es verdadera (TRUE) . La operación se realiza hasta que se se llega a cumplir un criterio previamente establecido.

El modelo while() es:

while()

Este es un tipo de bucle que ocurre mientras una condición es verdadera (TRUE) . La operación se realiza hasta que se se llega a cumplir un criterio previamente establecido.

El modelo while() es:

```
while(condicion) {
  operaciones
}
```

while()

Este es un tipo de bucle que ocurre mientras una condición es verdadera (TRUE) . La operación se realiza hasta que se se llega a cumplir un criterio previamente establecido.

El modelo while() es:

```
while(condicion) {
  operaciones
}
```

Con lo anterior le decimos a R:

- **MIENTRAS** esta condición sea **VERDADERA**, haz estas operaciones.

La condición generalmente es expresada como el resultado de una o varias operaciones de comparación.

Ejemplo

```
umbral <- 5
valor <- 0

while(valor < umbral) {
  print("Todavia no.")
  valor <- valor + 1
}
```

Ejemplo

```
umbral <- 5
valor <- 0

while(valor < umbral) {
  print("Todavia no.")
  valor <- valor + 1
}
```

```
[1] "Todavía no."
[1] "Todavía no."
[1] "Todavía no."
[1] "Todavía no."
[1] "Todavía no."
```

¡Ten cuidado con crear bucles infinitos! Si ejecutas un while con una condición que nunca será **FALSE**, este nunca se detendrá.

Ejemplo

Si corres lo siguiente, presiona la tecla ESC para detener la ejecución, de otro modo, correrá por siempre y puede llegar a congelar tu equipo.

```
while(1 < 2) {
  print("Presiona ESC para detener")
}
```

Ejemplo

Si corres lo siguiente, presiona la tecla ESC para detener la ejecución, de otro modo, correrá por siempre y puede llegar a congelar tu equipo.

```
while(1 < 2) {
  print("Presiona ESC para detener")
}
```

```
[1] "Presiona ESC para detener"
[1] "Presiona ESC para detener"
[1] "Presiona ESC para detener"
[1] "Presiona ESC para detener"
[1] "Presiona ESC para detener"
...
...
...
```

Ejemplo

El siguiente es un error común.

```
i <- 0
while(i < 10) {
  i + 1
}
```

Ejemplo

El siguiente es un error común.

```
i <- 0
while(i < 10) {
  i + 1
}
```

Estamos sumando 1 a **i** con cada iteración del bucle, pero como no estamos asignando este nuevo valor á i, su valor se mantiene igual, entonces la condición nunca se cumplirá y el bucle será infinito.

Ejemplo

El siguiente es un error común.

```
i <- 0
while(i < 10) {
  i + 1
}
```

Estamos sumando 1 a **i** con cada iteración del bucle, pero como no estamos asignando este nuevo valor á i, su valor se mantiene igual, entonces la condición nunca se cumplirá y el bucle será infinito.

```
i <- 0
while(i < 10) {
 i<- i + 1
}
```

Ejemplo

El siguiente es un error común.

```
i <- 0
while(i < 10) {
  i + 1
}
```

Estamos sumando 1 a **i** con cada iteración del bucle, pero como no estamos asignando este nuevo valor á i, su valor se mantiene igual, entonces la condición nunca se cumplirá y el bucle será infinito.

```
i <- 0
while(i < 10) {
 i<- i + 1
}
```

```
[1] 10
```

Ejercicio

Supongamos que, recibe unidades al azar de un producto A con un peso aletorio en Kg entre entre 1 y 10 Kg. Usted necesita almacenar dichos productos en cajas que soportan máximo 50 Kg.

- Realice la simulación de un caso en particular, en el cual, usted reciba aleatoriamente valores entre 1 y 10 (enteros) que representen los kg. Use la función sample()
- Interesa conocer cuantos kg máximo logró almacenar en la caja.
- Interesa conocer cuantos productos logro almacenar en la caja.

Ejercicio

```
conteo <- 0
peso <- 0
caja <- 0

while(peso < 50) {
  peso <- peso + sample(x = 1:10, size = 1)
  conteo <- conteo + 1
  caja[conteo]<-peso
}

conteo
peso
caja

Peso_acum<-caja[length(caja)-1]
Productos<-length(caja)-1

Peso_acum
Productos
```

Contenido

break y next

- **break** y **next** son palabras reservadas en R, no podemos asignarles nuevos valores y realizan una operación específica cuando aparecen en nuestro código.

- **break** nos permite interrumpir un bucle, mientras que **next** nos deja avanzar a la siguiente iteración del bucle, "saltándose" la actual. Ambas funcionan para for y while.

break y next

```
for(i in 1:10) {
  if(i == 3) {
    break
  }
  print(i)
}
```

break y next

```
for(i in 1:10) {
  if(i == 3) {
    break
  }
  print(i)
}
```

```
[1] 1
[1] 2
```

break y next

```
for(i in 1:10) {
  if(i == 3) {
    break
  }
  print(i)
}
```

```
[1] 1
[1] 2
```

```
for(i in 1:4) {
  if(i == 3) {
    next
  }
  print(i)
}
```

break y next

```
for(i in 1:10) {
  if(i == 3) {
    break
  }
  print(i)
}
```

```
[1] 1
[1] 2
```

```
for(i in 1:4) {
  if(i == 3) {
    next
  }
  print(i)
}
```

```
[1] 1
[1] 2
[1] 4
```

Contenido

Contenido

apply()

apply() permite la ejecución de funciones sobre partes de una matriz (o array); sólo será necesario indicar la función a ejecutar (suma, producto, media, varianza...) y si ésta se aplicará sobre las filas de la matriz (1), o sobre las columnas (2).

```
x <- matrix (1:8, ncol = 2)
x
```

apply()

apply() permite la ejecución de funciones sobre partes de una matriz (o array); sólo será necesario indicar la función a ejecutar (suma, producto, media, varianza...) y si ésta se aplicará sobre las filas de la matriz (1), o sobre las columnas (2).

```
x <- matrix (1:8, ncol = 2)
x
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

```
[1]  6  8 10 12
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

```
[1]  6  8 10 12
```

```
apply (x, 2, prod) #multiplica los valores de las columnas de x
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

```
[1]  6  8 10 12
```

```
apply (x, 2, prod) #multiplica los valores de las columnas de x
```

```
[1]   24 1680
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

```
[1]  6  8 10 12
```

```
apply (x, 2, prod) #multiplica los valores de las columnas de x
```

```
[1]   24 1680
```

```
apply (x,1, mean) #calcula la media de cada fila
```

apply()

```
apply (x, 1, sum) #suma los valores de las filas de x
```

```
[1]  6  8 10 12
```

```
apply (x, 2, prod) #multiplica los valores de las columnas de x
```

```
[1]   24 1680
```

```
apply (x,1, mean) #calcula la media de cada fila
```

```
[1] 3 4 5 6
```

Contenido

sapply(), lapply()

Existen funciones como lapply() o sapply() que se aplican sobre los elementos de una lista. El resultado de lapply() es siempre una lista, y el resultado de sapply() es un vector o array.

```
x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
x
```

sapply(), lapply()

Existen funciones como lapply() o sapply() que se aplican sobre los elementos de una lista. El resultado de lapply() es siempre una lista, y el resultado de sapply() es un vector o array.

```
x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
x
```

```
sapply (x, mean)
```

sapply(), lapply()

Existen funciones como lapply() o sapply() que se aplican sobre los elementos de una lista. El resultado de lapply() es siempre una lista, y el resultado de sapply() es un vector o array.

```
x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
x
```

```
sapply (x, mean)
```

```
         a          b          c
-0.2655813 -0.1122553 -0.2829864
```

sapply(), lapply()

Existen funciones como lapply() o sapply() que se aplican sobre los elementos de una lista. El resultado de lapply() es siempre una lista, y el resultado de sapply() es un vector o array.

```
x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
x
```

```
sapply (x, mean)
```

```
         a          b          c
-0.2655813 -0.1122553 -0.2829864
```

```
lapply (x, mean)
```

sapply(), lapply()

Existen funciones como lapply() o sapply() que se aplican sobre los elementos de una lista. El resultado de lapply() es siempre una lista, y el resultado de sapply() es un vector o array.

```
x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
x
```

```
sapply (x, mean)
```

```
         a          b          c
-0.2655813 -0.1122553 -0.2829864
```

```
lapply (x, mean)
```

```
$a
[1] -0.2655813

$b
[1] -0.1122553

$c
[1] -0.2829864
```

Contenido

tapply()

Si el objetivo es realizar cálculos condicionados sobre una variable en función de los niveles de un factor la función a utilizar es tapply(). Por ejemplo, calcular las medias de la variable altura en función del sexo.

```
sexo <- factor(c("mujer", "hombre", "hombre", "mujer",
"mujer", "hombre", "hombre", "hombre"))
altura <- c(1.60, 1.87, 1.70, 1.69, 1.58, 1.4, 1.80,1.79)
data <- data.frame(sexo,altura)
data
```

tapply()

Si el objetivo es realizar cálculos condicionados sobre una variable en función de los niveles de un factor la función a utilizar es tapply(). Por ejemplo, calcular las medias de la variable altura en función del sexo.

```
sexo <- factor(c("mujer", "hombre", "hombre", "mujer",
"mujer", "hombre", "hombre", "hombre"))
altura <- c(1.60, 1.87, 1.70, 1.69, 1.58, 1.4, 1.80,1.79)
data <- data.frame(sexo,altura)
data
```

```
    sexo altura
1  mujer   1.60
2 hombre   1.87
3 hombre   1.70
4  mujer   1.69
5  mujer   1.58
6 hombre   1.40
7 hombre   1.80
8 hombre   1.79
```

tapply()

```
    sexo altura
1  mujer   1.60
2 hombre   1.87
3 hombre   1.70
4  mujer   1.69
5  mujer   1.58
6 hombre   1.40
7 hombre   1.80
8 hombre   1.79
```

tapply()

```
    sexo altura
1  mujer   1.60
2 hombre   1.87
3 hombre   1.70
4  mujer   1.69
5  mujer   1.58
6 hombre   1.40
7 hombre   1.80
8 hombre   1.79
```

```
tapply(data$altura,data$sexo,mean)
```

tapply()

```
    sexo altura
1  mujer   1.60
2 hombre   1.87
3 hombre   1.70
4  mujer   1.69
5  mujer   1.58
6 hombre   1.40
7 hombre   1.80
8 hombre   1.79
```

```
tapply(data$altura,data$sexo,mean)
```

```
   mujer   hombre
1.623333 1.712000
```

Contenido

Funciones

Una función es un tipo de algoritmo, que describe una secuencia de ordenes para hacer una tarea especifica. En general toda función debe de tener un nombre, lista de parámetros, un algoritmo y un resultado:

- Una función se define con function()

```
name <- function(arg_1, arg_2, ...){

 expression

 }
```

Funciones

Una función es un tipo de algoritmo, que describe una secuencia de ordenes para hacer una tarea especifica. En general toda función debe de tener un nombre, lista de parámetros, un algoritmo y un resultado:

- Una función se define con function()

```
name <- function(arg_1, arg_2, ...){

 expression

 }
```

Está compuesta por:

- Nombre de la función (name)
- Argumentos (arg_1, arg_2, ...)
- Cuerpo (expression): emplea los argumentos para generar un resultado

Funciones

```
myFun <- function(x, y)
{
x + y
}

myFun
```

Funciones

```
myFun <- function(x, y)
{
x + y
}

myFun
```

```
function(x, y)
{
  x + y
}
```

Funciones

```
myFun(1, 2)
```

Funciones

```
myFun(1, 2)
```

```
[1] 3
```

Funciones

```
myFun(1, 2)
```

```
[1] 3
```

```
myFun(1:10, 21:30)
```

Funciones

```
myFun(1, 2)
```

```
[1] 3
```

```
myFun(1:10, 21:30)
```

```
[1] 22 24 26 28 30 32 34 36 38 40
```

Funciones

```
myFun(1, 2)
```

```
[1] 3
```

```
myFun(1:10, 21:30)
```

```
[1] 22 24 26 28 30 32 34 36 38 40
```

```
myFun(1:10, 3)
```

Funciones

```
myFun(1, 2)
```

```
[1] 3
```

```
myFun(1:10, 21:30)
```

```
[1] 22 24 26 28 30 32 34 36 38 40
```

```
myFun(1:10, 3)
```

```
[1] 4 5 6 7 8 9 10 11 12 13
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(1:10, exp=2)
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(exp=2, x=1:10)
```

Funciones

```
power <- function(x, exp)
{
x^exp
}

power(x=1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(1:10, exp=2)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

```
power(exp=2, x=1:10)
```

```
[1]  1  4  9 16 25 36 49 64 81 100
```

Funciones

Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp = 2)
{
x ^ exp
}
```

Funciones

Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp = 2)
{
x ^ exp
}
```

```
power(1:10)
```

Funciones

Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp = 2)
{
x ^ exp
}
```

```
power(1:10)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

Funciones

Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp = 2)
{
x ^ exp
}
```

```
power(1:10)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

```
power(1:10, 3)
```

Funciones

Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp = 2)
{
x ^ exp
}
```

```
power(1:10)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

```
power(1:10, 3)
```

```
 [1]    1    8   27   64  125  216  343  512  729 1000
```

Funciones

Funciones sin argumento

```
hello <- function()
{
print('Hello world!')
}
```

Funciones

Funciones sin argumento

```
hello <- function()
{
print('Hello world!')
}
```

```
hello()
```

Funciones

Funciones sin argumento

```
hello <- function()
{
print('Hello world!')
}
```

```
hello()
```

```
[1] ''Hello world!''
```

Funciones

Clases de variables en una función

- Parámetros formales (argumentos): x, y
- Variables locales (definiciones internas): z, w, m
- Variables libres: a, b
- return() devuelve siempre el resultado de una función

```
myFun <- function(x, y){
z <- x^2
w <- y^3
m <- a*z + b*w
m
}
```

Funciones

Clases de variables en una función

- Parámetros formales (argumentos): x, y
- Variables locales (definiciones internas): z, w, m
- Variables libres: a, b
- return() devuelve siempre el resultado de una función

```
myFun <- function(x, y){
z <- x^2
w <- y^3
m <- a*z + b*w
m
}
```

```
a <- 10
b <- 20

myFun(2, 3)
```

Funciones

Clases de variables en una función

- Parámetros formales (argumentos): x, y
- Variables locales (definiciones internas): z, w, m
- Variables libres: a, b
- return() devuelve siempre el resultado de una función

```
myFun <- function(x, y){
z <- x^2
w <- y^3
m <- a*z + b*w
m
}
```

```
a <- 10
b <- 20

myFun(2, 3)
```

```
[1] 580
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien
```

```
warning('Quizás algo no es como debiera...')
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien
```

```
warning('Quizás algo no es como debiera...')
```

```
Warning message:
Quizás algo no es como debiera...
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien
```

```
warning('Quizás algo no es como debiera...')
```

```
Warning message:
Quizás algo no es como debiera...
```

```
message('Todo en orden por estos lares.')
```

Funciones

Mensajes para el usuario:

- stop para la ejecución y emite un mensaje de error
- warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias
- message emite un mensaje

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien
```

```
warning('Quizás algo no es como debiera...')
```

```
Warning message:
Quizás algo no es como debiera...
```

```
message('Todo en orden por estos lares.')
```

```
Todo en orden por estos lares.
```

Ejercicio

- Construya una función que calcule el área de un rectángulo.
- Construya una función que se llame convertidorPD. La cuál va a convertir de dólares a pesos colombianos. Suponga que 1 dólar equivale a 3337 pesos colombianos.

  Construya el resultado de tal manera que la salida sea en texto especificando por ejemplo: "1 US$ dólar(es) equivale a 3337 pesos COP". Realice el ejercicio simulando 20 valores de US$.

Ejercicio

```
area.rectangulo <- function(base, altura) {
area = base*altura
area
}

area.rectangulo(7, 5)
```

Ejercicio

```
area.rectangulo <- function(base, altura) {
area = base*altura
area
}

area.rectangulo(7, 5)
```

```
dolar<-sample(1:50,20)

ConvertidorPD <- function(x){ ## x es el no de dólares a convertir
  x1<- x*3337
  paste(x,"US$ dólar(es)","equivale a", x1,"pesos COP")
}

ConvertidorPD(dolar)
```

Contenido

Fechas as.Date()

```
as.Date('2013-02-06')
```

Fechas as.Date()

```
as.Date('2013-02-06')
```

```
[1] "2013-02-06"
```

Fechas as.Date()

```
as.Date('2013-02-06')
```

```
[1] "2013-02-06"
```

```
as.Date('2013/02/06')
```

Fechas as.Date()

```
as.Date('2013-02-06')
```

```
[1] "2013-02-06"
```

```
as.Date('2013/02/06')
```

```
[1] "2013-02-06"
```

Fechas as.Date()

```
as.Date('2013-02-06')
```

```
[1] "2013-02-06"
```

```
as.Date('2013/02/06')
```

```
[1] "2013-02-06"
```

```
as.Date('06.02.2013', format=' %d. %m. %Y')
```

Fechas as.Date()

```
as.Date('2013-02-06')
```

```
[1] "2013-02-06"
```

```
as.Date('2013/02/06')
```

```
[1] "2013-02-06"
```

```
as.Date('06.02.2013', format=' %d. %m. %Y')
```

```
[1] "2013-02-06"
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

```
[1] "2004-01-01" "2004-01-02" "2004-01-03" "2004-01-04" "2004-01-05"
[6] "2004-01-06" "2004-01-07" "2004-01-08" "2004-01-09" "2004-01-10"
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

```
[1] "2004-01-01" "2004-01-02" "2004-01-03" "2004-01-04" "2004-01-05"
[6] "2004-01-06" "2004-01-07" "2004-01-08" "2004-01-09" "2004-01-10"
```

```
seq(as.Date('2004-01-01'), by='month', length=10)
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

```
[1] "2004-01-01" "2004-01-02" "2004-01-03" "2004-01-04" "2004-01-05"
[6] "2004-01-06" "2004-01-07" "2004-01-08" "2004-01-09" "2004-01-10"
```

```
seq(as.Date('2004-01-01'), by='month', length=10)
```

```
[1] "2004-01-01" "2004-02-01" "2004-03-01" "2004-04-01" "2004-05-01"
[6] "2004-06-01" "2004-07-01" "2004-08-01" "2004-09-01" "2004-10-01"
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

```
[1] "2004-01-01" "2004-01-02" "2004-01-03" "2004-01-04" "2004-01-05"
[6] "2004-01-06" "2004-01-07" "2004-01-08" "2004-01-09" "2004-01-10"
```

```
seq(as.Date('2004-01-01'), by='month', length=10)
```

```
[1] "2004-01-01" "2004-02-01" "2004-03-01" "2004-04-01" "2004-05-01"
[6] "2004-06-01" "2004-07-01" "2004-08-01" "2004-09-01" "2004-10-01"
```

```
seq(as.Date('2004-01-01'), by='10 day', length=10)
```

Fechas as.Date()

```
seq(as.Date('2004-01-01'), by='day', length=10)
```

```
[1] "2004-01-01" "2004-01-02" "2004-01-03" "2004-01-04" "2004-01-05"
[6] "2004-01-06" "2004-01-07" "2004-01-08" "2004-01-09" "2004-01-10"
```

```
seq(as.Date('2004-01-01'), by='month', length=10)
```

```
[1] "2004-01-01" "2004-02-01" "2004-03-01" "2004-04-01" "2004-05-01"
[6] "2004-06-01" "2004-07-01" "2004-08-01" "2004-09-01" "2004-10-01"
```

```
seq(as.Date('2004-01-01'), by='10 day', length=10)
```

```
[1] "2004-01-01" "2004-01-11" "2004-01-21" "2004-01-31" "2004-02-10"
[6] "2004-02-20" "2004-03-01" "2004-03-11" "2004-03-21" "2004-03-31"
```

Contenido

Contenido

Exploración de datos

Indicadores de tendencia central

- Indicadores de tendencia central: Muestran hacia donde tienden la mayoría de los datos, un ejemplo es el promedio, que también se conoce como "el centro de gravedad de los datos".
- Los indicadores son: Promedio, Mediana y Moda.
- Las funciones respectivas en R son: mean(), median(). R no dispone de una función en su paquete base (stats). Por lo cuál se utiliza el paquete "modeest"

```
install.packages("modeest")

library(modeest)

mlv(runif(20), method = "mfv")[1] ## Genera el valor más frecuente
```

Exploración de datos

Indicadores de tendencia central

Para el siguiente vector, calcule la media, mediana y moda

```
x <- c(20,NA,10,15,NA,25,22,NA)
```

Exploración de datos

Indicadores de tendencia central

Para el siguiente vector, calcule la media, mediana y moda

```
x <- c(20,NA,10,15,NA,25,22,NA)
```

```
mean(x,na.rm=T)
median(x,na.rm=T)
mlv(x, method = "mfv",na.rm=T)[1]
```

Exploración de datos

Indicadores de tendencia central

Para el siguiente vector, calcule la media, mediana y moda

```
x <- c(20,NA,10,15,NA,25,22,NA)
```

```
mean(x,na.rm=T)
median(x,na.rm=T)
mlv(x, method = "mfv",na.rm=T)[1]
```

```
> mean(x,na.rm=T)
[1] 18.4

> median(x,na.rm=T)
[1] 20

> mlv(x, method = "mfv",na.rm=T)[1]
[1] 10
```

En el caso de variable cuantitativas continuas la moda corresponde a los valores alrededor de los cuales se produce la mayor concentración de los datos.

Exploración de datos

Indicadores de dispersión

Suponga que se tienen tres grupos de personas con las siguientes estaturas:

```
Grupo1 <- c(60, 100, 140, 180)
Grupo2 <- c(100, 100, 140, 140)
Grupo3 <- c(120, 120, 120, 120)
```

Exploración de datos

Indicadores de dispersión

Suponga que se tienen tres grupos de personas con las siguientes estaturas:

```
Grupo1 <- c(60, 100, 140, 180)
Grupo2 <- c(100, 100, 140, 140)
Grupo3 <- c(120, 120, 120, 120)
```

```
mean(Grupo1); mean(Grupo2); mean(Grupo3)
var(Grupo1); var(Grupo2); var(Grupo3)
sd(Grupo1); sd(Grupo2); sd(Grupo3)
```

Analice que dichos grupos tienen igual promedio pero su variabilidad es distinta

Exploración de datos

Coeficiente de variación

Coeficiente de Variación: Ayuda a identificar si los datos son homogéneos o heterogéneos. Determinar si cierta media es consistente con cierta varianza. Su cálculo es:

$$CV(x) = \frac{\sigma}{\overline{x}} * 100 \tag{1}$$

Ejercicio: Genere una función que permita calcular el coeficiente de variación a los grupos anteriormente establecidos.

Exploración de datos

Indicadores de posición

- Los indicadores de posición que más se trabaja en el análisis descriptivo son los cuartiles, estos dividen la muestra ordenada en cuatro partes que contienen aproximadamente el mismo número de datos
- La función en R sería quantile()
- También se pueden calcular los percentiles. Por ejemplo: P80 A partir de que valor se encuentra el 80 % de los datos.

Exploración de datos

Indicadores de posición

En la base de datos decathlon del paquete FactoMineR, calcule el percentil 70 con la prueba 100m. Interprete dichos valores.

```
library(FactoMineR)
data(decathlon)

quantile(decathlon[,1],seq(0,1,0.1))
```

Exploración de datos

Indicadores de posición

En la base de datos decathlon del paquete FactoMineR, calcule el percentil 70 con la prueba 100m. Interprete dichos valores.

```
library(FactoMineR)
data(decathlon)

quantile(decathlon[,1],seq(0,1,0.1))
```

```
  70%
11.11
```

El 70 % de los competidores presentaron tiempos menores o iguales a 11.11 segundos

Contenido

Exploración de datos

Datos agrupados

Sea la variable Ingreso en millones, conforme una tabla de frecuencias. Explore la función table.freq() de la librería "agricolae".

```
library(agricolae)

Ingreso <- c(2, 1.5, 3, 2.2, 1, 1.2, 3, 4, 5, 1, 2)
tbFreq <- table.freq(hist(Ingreso, plot=FALSE))

tbFreq
```

Exploración de datos

Datos agrupados

Sea la variable Ingreso en millones, conforme una tabla de frecuencias. Explore la función table.freq() de la librería "agricolae".

```
library(agricolae)

Ingreso <- c(2, 1.5, 3, 2.2, 1, 1.2, 3, 4, 5, 1, 2)
tbFreq <- table.freq(hist(Ingreso, plot=FALSE))

tbFreq
```

```
  Lower Upper Main Frequency Percentage CF   CPF
1     1     2  1.5         6       54.5  6  54.5
2     2     3  2.5         3       27.3  9  81.8
3     3     4  3.5         1        9.1 10  90.9
4     4     5  4.5         1        9.1 11 100.0
```

Y calcule la media en datos agrupados utilizando las funciones apply().
R// 2.228

Contenido

Actividad 1

Decathlon es una base de datos con 41 filas y 13 columnas: las primeras diez columnas corresponden al desempeño de los atletas para los 10 eventos del decatlón. Las columnas 11 y 12 corresponden respectivamente al rango y los puntos obtenidos. La última columna es una variable categórica correspondiente al evento deportivo (2004 Olympic Game o 2004 Decastar).

- Genere una función y apliquela mediante un ciclo sobre las variables cuantitativas (columna 1 a la 10) de tal manera que guarde en una matriz la siguiente información por fila: Media, Mediana, Desviación Estándar, Mínimo, Máximo.
- Realice el cálculo de la media y desviación estándar por la variable competition de las columnas 1 a la 10. Y acomode la información en una matriz. Sug. tapply()
- Seleccione una variable de la data y realice una tabla de frecuencias de datos agrupados.

Actividad 1

Punto 1.

```
Estadisticas<-function(variable){

 media_variable<-mean(variable)
 mediana_variable<-median(variable)
 desvest_variable<-sd(variable)
 min_variable<-min(variable)
 max_variable<-max(variable)

 Resultado<-c(media_variable,mediana_variable,
              desvest_variable,min_variable,max_variable)

 names(Resultado)<-c("Media", "Mediana", "Desv. Est.", "Min", "Max")

 return(Resultado)

 }
```

Actividad 1

```
M<-matrix(,5,10)

for(i in 1:10){

M[,i]<- Estadisticas(decathlon[,i])

}

rownames(M)<-c("Media", "Mediana", "Desv. Est.", "Min", "Max")
colnames(M)<-names(decathlon)[1:10]

M
```

Actividad 1

Punto 2.

```
M2<-matrix(,10,2)
M3<-matrix(,10,2)

for(i in 1:10){
  M2[i,]<-tapply(decathlon[,i],decathlon$Competition,mean)
  M3[i,]<-tapply(decathlon[,i],decathlon$Competition,sd)
}

rownames(M2)<-names(decathlon)[1:10]
colnames(M2)<-levels(decathlon$Competition)

rownames(M3)<-names(decathlon)[1:10]
colnames(M3)<-levels(decathlon$Competition)

M2
M3
```

Actividad 1

Punto 3.

```
tbFreq <- table.freq(hist(decathlon$Long.jump, plot=FALSE))

tbFreq
```

Contenido

Contenido

Gráficos

Función plot()

- La función plot() esta en el paquete "graphics". Sirve para graficar funciones en 2d.
- Se utiliza para hacer diagramas de dispersión, donde se relacionan 2 variables una X (independiente) y Y(dependiente).
- plot(x,y).
- También se utiliza para gráficas series tiempo. Aunque en este aspecto se recomienda la función ts.plot()

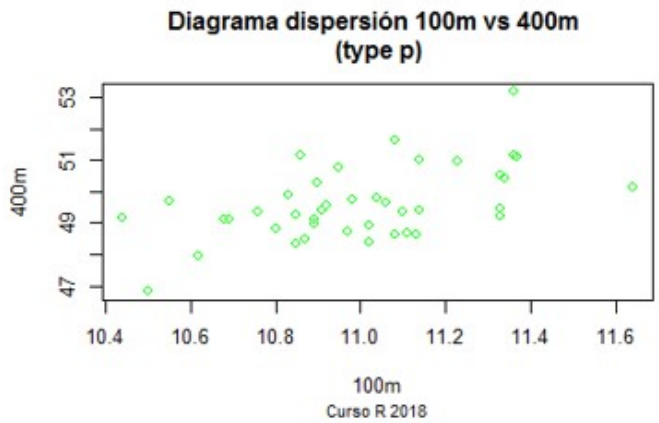
Gráficos

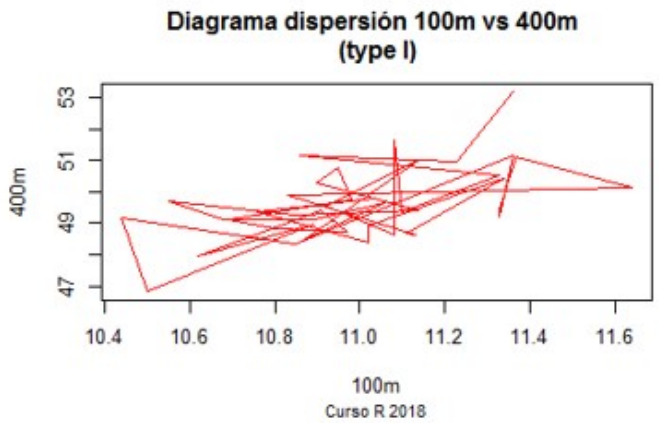
Función plot()

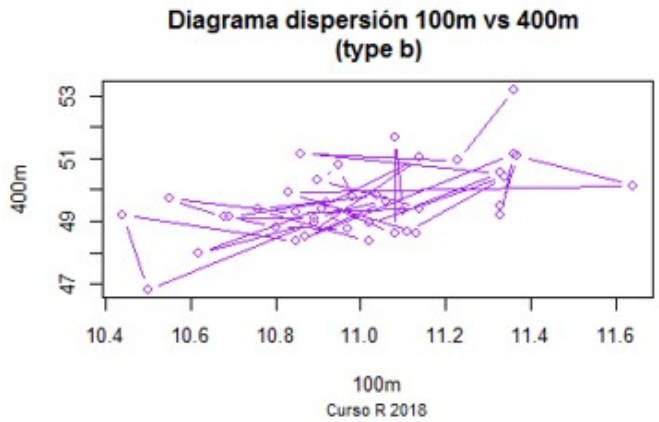
- Algunos parámetros de la función plot() son:
  - main: título del gráfico
  - xlab: nombre del eje x
  - ylab: nombre del eje y
  - xlim y ylim: Indica el rango visible de los ejes.
  - col: color de los puntos o lineas del gráfico
  - type: como quiero que sean los puntos “p” “l” “s” , etc

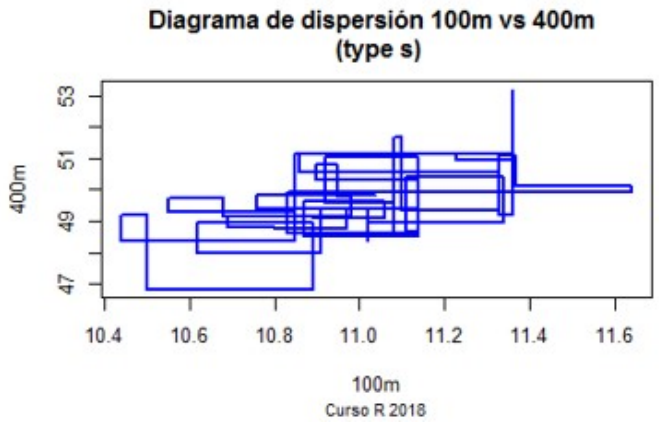
Gráficos

```
plot(decathlon[,1],decathlon[,5],xlab="100m",ylab="400m",
    col="green",main="Diagrama dispersión 100m vs 400m \n (type p)",
     sub="Curso R 2020",cex.sub=0.8)
```









Gráficos

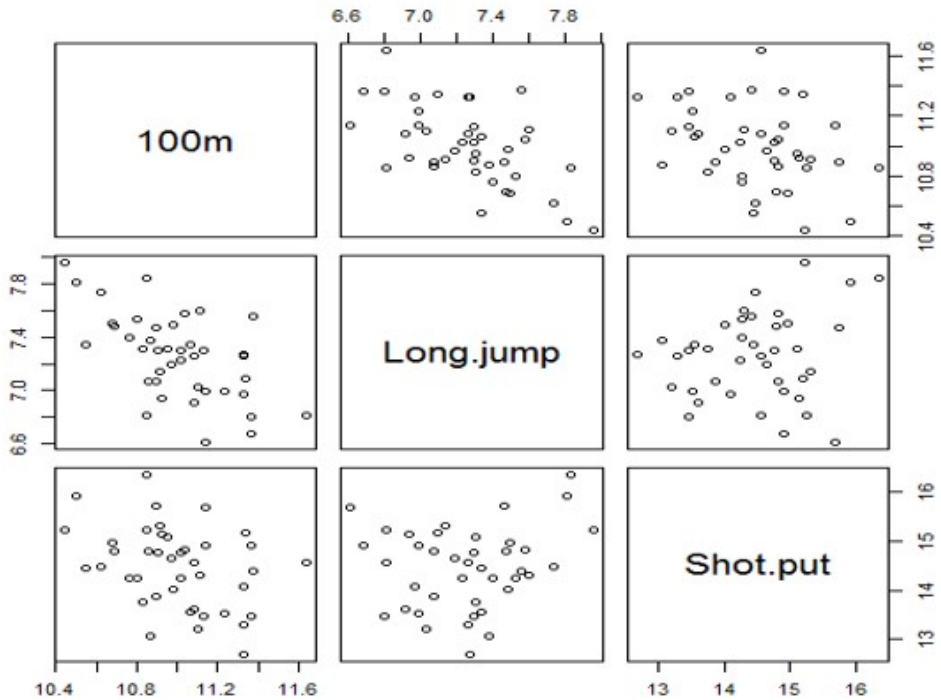
```
plot(decathlon[,1],decathlon[,5],xlab="100m",ylab="400m",
    col="green",main="Diagrama dispersión 100m vs 400m \n (type p)",
     sub="Curso R 2020",cex.sub=0.8)
```

El comando \n es un "enter" dentro del título del gráfico
Involucre el parámetro type = "l"

Observe que pasa en el gráfico. Interactúe con las demás opciones del parámetro type

Gráficos

```
plot(decathlon[,1:3])
```



Contenido

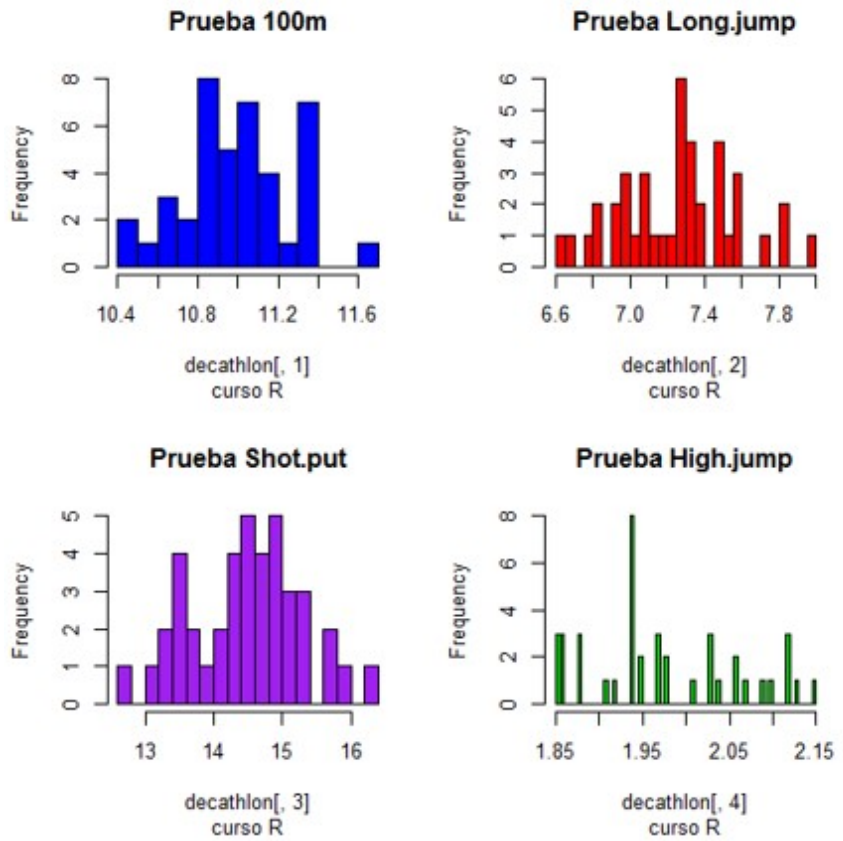
Gráficos

Función hist()

- La función hist() realiza un histograma, gráfico que se utiliza para variables cuantitativas continuas. La función hist() por defecto realiza los intervalos de clase con la regla de sturges.
- Los parametros de la función hist son similares a los de plot()
- Breaks indica el número de clases

Gráficos

```
hist(decathlon[,1],main="Prueba 100m",sub="curso R",
 breaks=10,col="blue")
```



Contenido

Gráficos

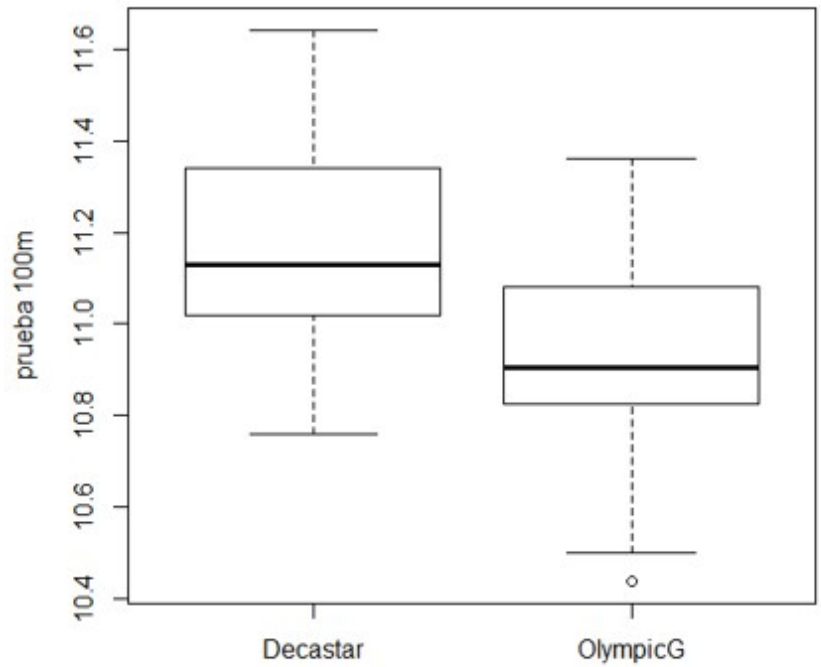
Función boxplot()

- Sirve para realizar un diagrama de cajas, el cuál puede ayudar en la identificación de datos atípicos “outliers”.
- La función recibe vectores o matrices

Gráficos

```
boxplot(decathlon[,1]~decathlon[,13],ylab="prueba 100m")

## Prueba 100m en func. del evento Decastar u OlympicG
## y~x
## Donde x es cualitativa
```



Contenido

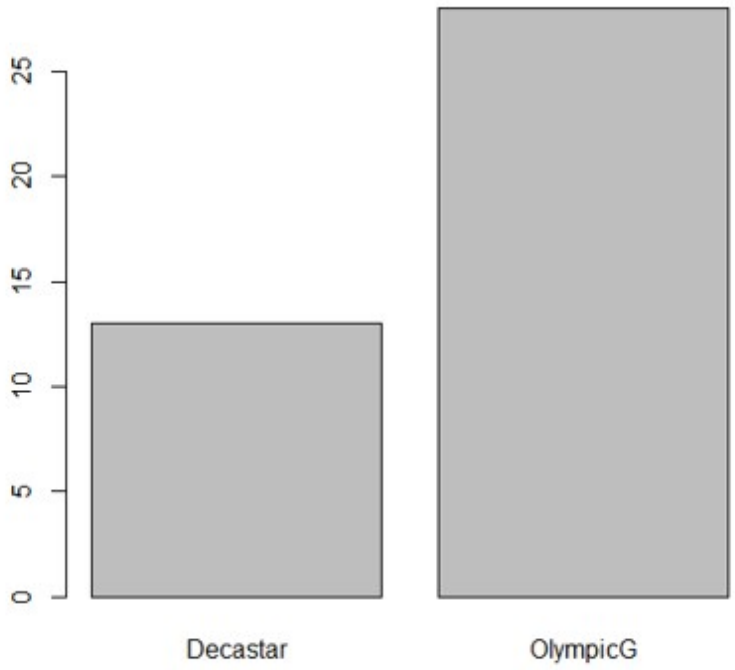
Gráficos

Función barplot()

- La función barplot() nos ayuda a crear diagramas de barras, los cuales se utilizan con variables cualitativas. Es necesario tener los conteos, los cuales se pueden hacer con la función table()

Gráficos

```
rr <- table(decathlon[,13])
barplot(rr)
```



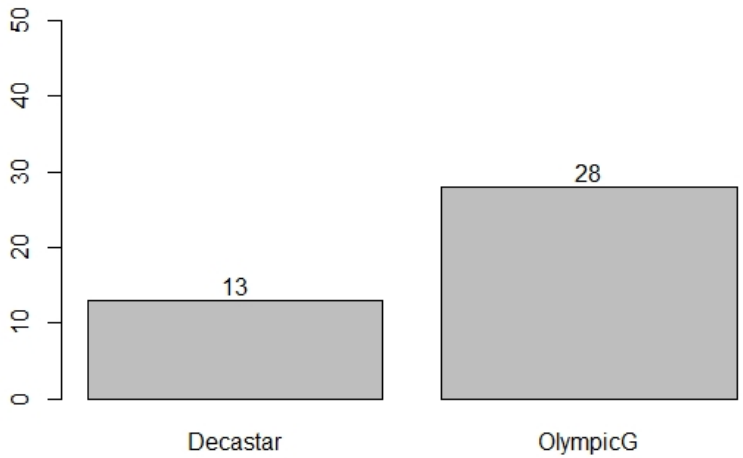
Gráficos

Función barplot()

- ¿Cómo agregar las frecuencias encima de las gráficas?
- Agregar un texto dentro del gráfico
- Función text()

Gráficos

```
a<-barplot(table(decathlon[,13]),ylim=c(0,50))
text(a,table(decathlon[,13])+2,table(decathlon[,13]))
```

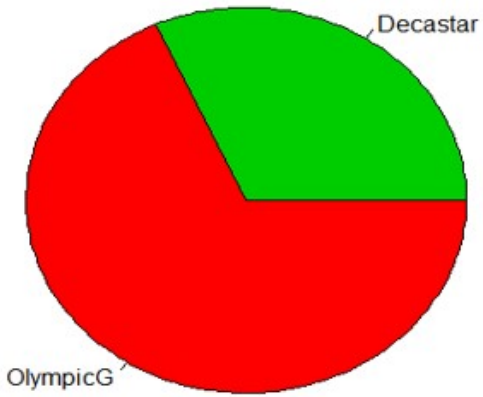


Contenido

Gráficos

Función pie()

```
rr <- table(decathlon[,13])
pie(rr,col=c(3,2))
```



Contenido

Gráficos

Función ggplot2()

https://rstudio.com/wp-content/uploads/2015/04/ggplot2-spanish.pdf

```
library(ggplot2)
```

```
## Histograma
ggplot(data=decathlon,aes(decathlon[,1]))+
geom_histogram(fill="blue")

## El signo + indica que se van agregando elementos al gráfico

## Boxplot
ggplot(data=decathlon,aes(decathlon[,13],decathlon[,1]))+
geom_boxplot(color="blue")
```

Gráficos

Función ggplot2()

```
library(ggplot2)
```

```
## Diagrama de disper.

qplot(decathlon[,1],decathlon[,5],data=decathlon,color="")
ggplot(data=decathlon,aes(decathlon[,1],decathlon[,5]))+
geom_jitter(color="red")+
ggtitle("Gráfico de Dispersión")+ ### Titulo
xlab("100m")+ ## label eje x
ylab("400m") ## label eje y
```

Gráficos

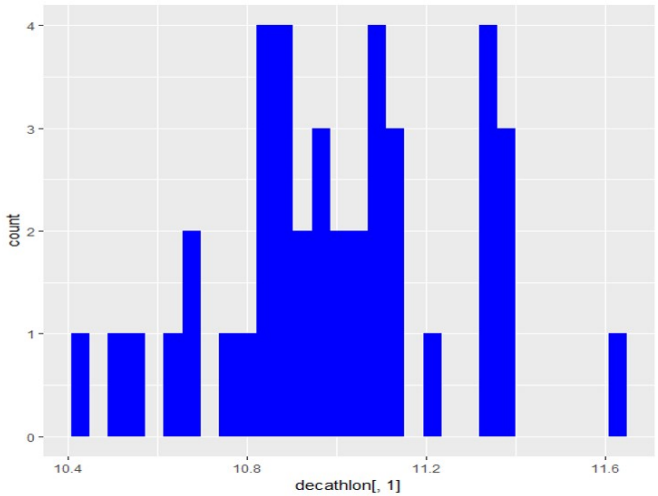
Función ggplot2()

```
library(ggplot2)
```

```
### Diagrama de Barras

ggplot(data=decathlon,aes(decathlon[,13]))+
geom_bar(fill="skyblue")+
ggtitle("Diagrama de Barras")+ ### Titulo
xlab("Evento") ## label eje x
```

Gráficos


count
decathlon[, 1]
0
1
2
3
4
10.4
10.8
11.2
11.6

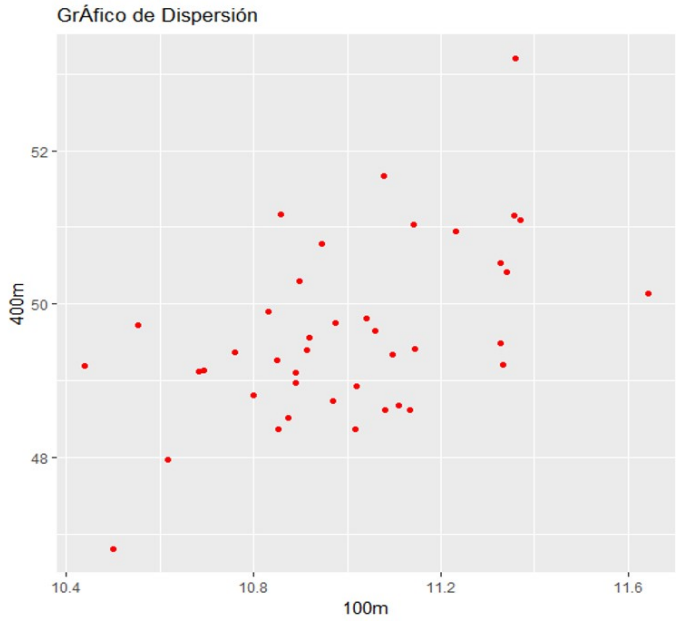


GrÁfico de Dispersión
400m
100m
48
50
52
10.4
10.8
11.2
11.6

Gráficos

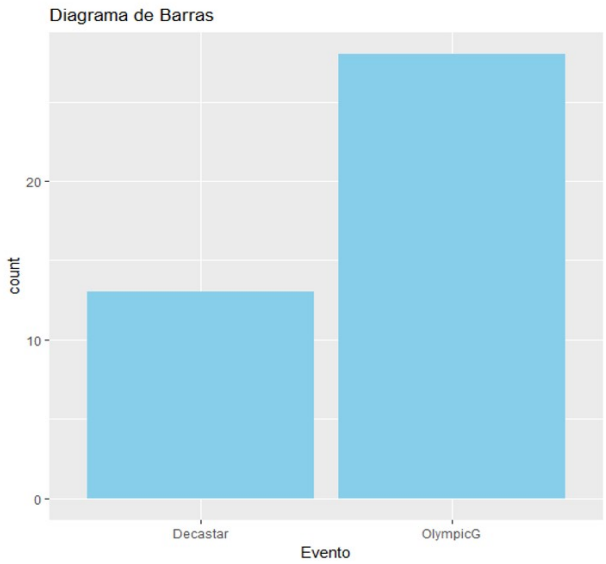
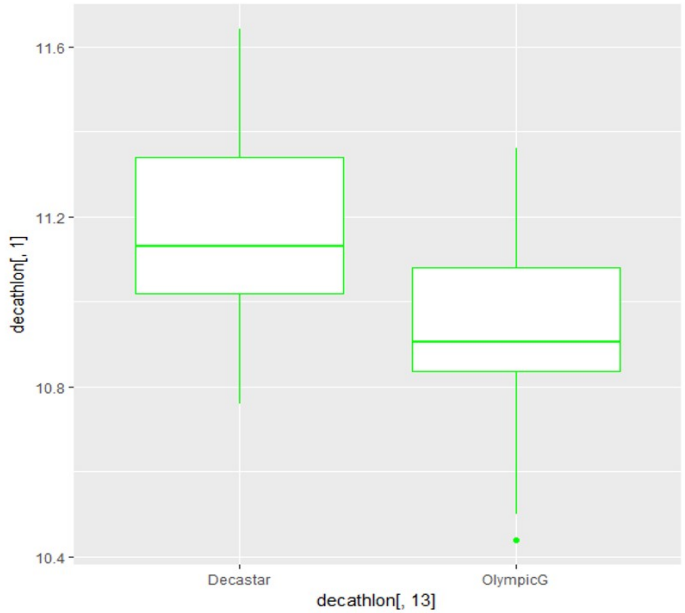


Diagrama de Barras
count
20
10
0
Decastar
OlympicG
Evento


11.6
11.2
10.8
10.4
decathlon[, 1]
Decastar
OlympicG
decathlon[, 13]

Contenido

Correlación

- La función cor() permite trabajar con dos vectores X y Y.
- También permite trabajar con una matriz numérica, la cuál generaría una matriz de correlación.
- Además la función cor() permite calcular la correlación de pearson, spearma o el tau de kendall.

```
cor(decathlon[,1],decathlon[,5])
```

Correlación

- La función cor() permite trabajar con dos vectores X y Y.
- También permite trabajar con una matriz numérica, la cuál generaría una matriz de correlación.
- Además la función cor() permite calcular la correlación de pearson, spearma o el tau de kendall.

```
cor(decathlon[,1],decathlon[,5])
```

```
[1] 0.5202982
```

Correlación

- La función cor() permite trabajar con dos vectores X y Y.
- También permite trabajar con una matriz numérica, la cuál generaría una matriz de correlación.
- Además la función cor() permite calcular la correlación de pearson, spearma o el tau de kendall.

```
cor(decathlon[,1],decathlon[,5])
```

```
[1] 0.5202982
```

```
cor(decathlon[,1:10])
```

Contenido

Regresión Lineal

- La función lm() permite estimar un modelo lineal Y $\sim$ X1 + X2 + ... + Xn .
- lm(Y $\sim$ X1 + X2 + ... + Xn , data = ...) Principalmente es necesario solo especificar la fórmula y el conjunto de datos en el cual se encuentran las variables.

```
m1<-lm(decathlon[,1] ~ decathlon[,5])
summary(m1)
```

Regresión Lineal

```
Call:
lm(formula = decathlon[, 1] ~ decathlon[, 5])

Residuals:
     Min       1Q   Median       3Q      Max
-0.50747 -0.16509 -0.02696  0.16982  0.57982

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     5.11135    1.54757   3.303 0.002056 **
decathlon[, 5]  0.11864    0.03118   3.805 0.000489 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2275 on 39 degrees of freedom
Multiple R-squared:  0.2707,Adjusted R-squared:  0.252
F-statistic: 14.48 on 1 and 39 DF,  p-value: 0.0004885
```

Regresión Lineal

```
names(m1)
```

Regresión Lineal

```
names(m1)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

Regresión Lineal

```
names(m1)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

```
m1$coefficients
```

Regresión Lineal

```
names(m1)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

```
m1$coefficients
```

```
  (Intercept) decathlon[, 5]
    5.1113520      0.1186443
```

Regresión Lineal

```
names(m1)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

```
m1$coefficients
```

```
   (Intercept) decathlon[, 5]
     5.1113520      0.1186443
```

```
m1$residuals
```

Regresión Lineal

```
names(m1)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

```
m1$coefficients
```

```
  (Intercept) decathlon[, 5]
    5.1113520      0.1186443
```

```
m1$residuals
```

```
          1           2           3           4
 0.018974735 -0.208821767  0.169822547  0.103381732
          5           6           7           8
 0.246601704  0.223042810  0.250161469 -0.202889696
 .....
 .....
 .....
```

Contenido

Actividad 2

Decathlon es una base de datos con 41 filas y 13 columnas: las primeras diez columnas corresponden al desempeño de los atletas para los 10 eventos del decatlón. Las columnas 11 y 12 corresponden respectivamente al rango y los puntos obtenidos. La última columna es una variable categórica correspondiente al evento deportivo (2004 Olympic Game o 2004 Decastar).

- Considera usted que hay relación entre la prueba 400m y Long Jump? (Realice un diagrama de dispersión, correlación y modelo de regresión)
- Realice un histograma de los residuales del modelo estimado en el punto anterior.

Actividad 1

Punto 1.

```
cor(decathlon$‘400m‘,decathlon$Long.jump)

plot(decathlon$‘400m‘,decathlon$Long.jump)

m1<-lm(decathlon$‘400m‘ ~ decathlon$Long.jump)
summary(m1)
```

Punto 2.

```
hist(m1$residuals)
```

Preguntas?

Gracias!! ,

Jr.

orlando.joaqui@correounivalle.edu.co