

Básicos de programación en R

Tipos de datos

integer son los números enteros y se denominan con una L al final

```
x <- 2L
```

#**typeof()** indica el tipo de dato que estamos usando

```
typeof(x)
```

#**double** es el tipo de dato por default y abarca enteros y decimales

```
y <- 2.5
```

```
typeof(y)
```

#**complex** son números complejos en R

```
z <- 3 + 2i
```

```
typeof(z)
```

#**character** son datos de tipo texto

```
a <- "hola mundo"
```

```
typeof(a)
```

#**logical** funciona para indicar elementos de tipo booleano como TRUE o FALSE

```
b <- T
```

```
typeof(b)
```

Usos con variables

```
#Primero se definen unas variables
```

```
A <- 5
```

```
B <- 10
```

```
#Nótese que se almacenan los resultado en diferentes variables como C, C1, etc. Posterior a  
eso se puede poner sólo la variable y el valor se despliega en la consola (e.j. C sería [1] 15)
```

```
#suma
```

```
C <- A + B
```

```
#resta
```

```
C1 <- A - B
```

```
#división
```

```
C2 <- A / B
```

```
#multiplicación
```

```
C3 <- A * B
```

```
#raíz cuadrada, aquí se usa la función sqrt para elevar al cuadrado A y se almacena en la  
variable C4
```

```
C4 <- sqrt(A)
```

```
# Ahora usando datos de tipo character. Se definen unas variables y se genera un mensaje con  
ellas en la región de ambiente
```

```
nombre <- "Pau"
```

```
saludo <- "hola"
```

```
pregunta <- "cómo estás?"
```

```
mensaje <- paste(saludo, nombre, pregunta)
```

Variables y operadores lógicos

#Operadores lógicos, aquí se llaman logical y son:

== igual a, ayuda a ver que dos elementos sean iguales

#TRUE

5 == 5

FALSE

5 == 4

!= diferente que, nos ayuda a confirmar que dos elementos sean distintos

#FALSE

5 != 5

#TRUE

5 != 4

< menor que

#TRUE

5 < 6

#FALSE

6 < 4

> mayor que

#FALSE

5 > 6

#TRUE

6 > 5

```
# <= menor o igual que
```

```
#TRUE
```

```
10 <= 10
```

```
#FALSE
```

```
11 <= 10
```

```
# >= mayor o igual que
```

```
#TRUE
```

```
10 >= 10
```

```
#FALSE
```

```
9 >= 10
```

```
# ! negación, niega el elemento al cual se une
```

```
#FALSE
```

```
!(10 > 9)
```

```
#TRUE
```

```
!(10 > 15)
```

```
# | indica disyunción en sentido booleano, si uno u otra TRUE entonces TRUE
```

```
x <- 5 < 6
```

```
y <- 5 > 6
```

```
z <- 5 < 2
```

```
#TRUE
```

```
x | y
```

```
#FALSE
```

```
x | y
```

& indica conjunción, y sólo es TRUE si ambas lo son

#FALSE

x & y

isTRUE(x) es para saber si algo es verdadero o no

a <- 5 < 4

#FALSE

isTRUE(a)

Ciclo while

#Funciona de tal manera que si la condición en () se cumple, entonces se ejecuta lo que está en {}. Esto lo repite de manera cíclica hasta que () es FALSE

```
#ciclo infinito
```

```
while(TRUE){
```

```
    print("hola")}
```

```
while(FALSE){
```

```
    print("hola")}
```

#Este código funciona hasta que contador llega a 12, esto regulado porque cada ciclo contador aumenta +1, por ello sólo puede hacer 11 ciclos

```
contador <- 1
```

```
while(contador < 12){
```

```
    print(contador)
```

```
    contador <- contador +1}
```

```
número <- (1 - 2) * 5
```

```
while(número < 100){
```

```
    print(número)
```

```
    número <- número + 10}
```

Ciclo for

#En esencia lo que está haciendo es poner en () un vector o secuencia de iteraciones y en {} la acción a iterar. Su condición es una secuencia en lugar de una operación lógica como en while

```
#Imprime 5 veces "hola R"
```

```
for(i in 1:5){  
  print("hola R")  
}
```

```
#Imprime 6 veces "hola R :D"
```

```
for(i in 5:10){  
  print("hola R :D")  
}
```

#Aquí el código reitera del 5 al 10 e imprime, alternando, los valores de 5:10 con el valor de i en el ciclo +3. Si 1:5 son 5, 6, 7, 8, 9, 10, entonces 5, 8, 7, 10, 8, 11, 9, 12, 10, 13.

```
for(i in 5:10){  
  print(i)  
  i <- i + 3  
  print(i)  
}
```

Condicional if, else, y else/if

```
#rnorm genera un números aleatorios con una distribución normal, te pide N, media y SD  
x <- rnorm(1)
```

#if es igual a while, donde () es la condición TRUE/FALSE y {} es la acción a ejecutar.La diferencia es que solamente corre una vez

#Para los casos en que queramos que suceda algo cuando () de if no se cumple se usa else

```
x <- rnorm(1)  
if(x > 1){  
  respuesta <- "Mayor que 1"  
}else{  
  if(x < -1){  
    respuesta <- "Entre -1 y 1"  
  }else{  
    respuesta <- "Menor a -1"  
  }  
}  
respuesta
```

#El código anterior genera un número al azar cercano a 1 y luego lo clasifica entre mayor que uno, entre -1 y 1, y menor a -1

#Esto es igual, solamente se redujo el espacio al juntar else if , así se elimina un corchete de al final. Se llaman **condicionales encadenados y los anteriores anidados**

```
x <- rnorm(1)  
if(x > 1){  
  respuesta <- "Mayor que 1"  
}else if(x < -1){  
  respuesta <- "Entre -1 y 1"  
}else{  
  respuesta <- "Menor a -1"  
}
```

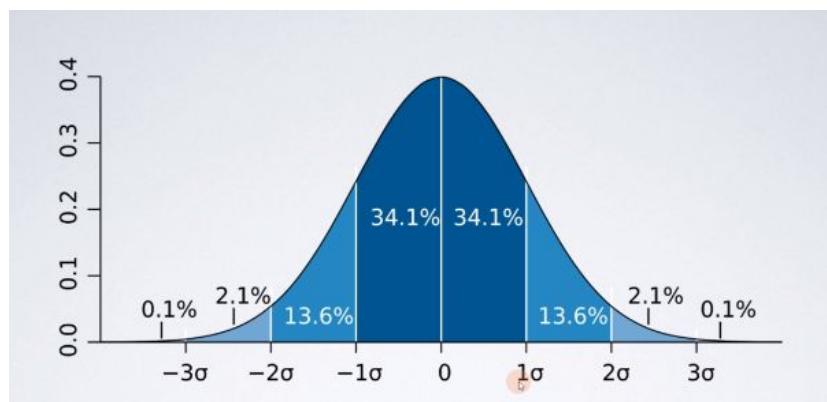
```
respuesta
```

```
#Otro ejemplo
x <- rnorm(1, mean= 20, sd= 10)
if(x > 18){
  respuesta <- "Bienvenido al club de los grandes"
}else{
  respuesta <- "Bienvenido al club de los pequeñines"
}
respuesta
```

Ley de los grandes números

$$\bar{X}_n \rightarrow E(X) \text{ when } n \rightarrow \infty$$

Según esta los valores de la media se aproximan a los esperados conforme el número de casos se acerca al infinito. Si n fuera el tiro de una moneda, entonces a mayor cantidad de tiros de moneda más se acerca la media al valor teórico esperado de $P = .50$ ó 50%.



Aquí se ve la distribución normal que en esencia todo conjunto de datos que se distribuya de esta manera tiende a que sus valores se acomoden alrededor de su media. Dentro del rango -1 a 1 desviaciones estándar. Si esto es el caso, los %s de arriba son ciertos para esa población.

Instrucciones:

Poner a prueba la Ley de los Grandes Números para N números aleatorios con distribución normal, donde la media = 0 y $stdev = 1$

Crear un script en R que cuente cuántas veces cae un número de estos entre -1 y 1 y divide por la cantidad total de observaciones

Sabes que $E(X) = 68.2\%$

Revisa que la Media de $(X_N) \rightarrow E(X)$ conforme corras de nuevo el script mientras incrementas N

Solución:

```
contador <- 0      # el contador inicial, se empieza sin ningún caso entre las SD
N <- 100000        # número de casos o vector
for(i in rnorm(N)){
  if(i > -1 & i < 1){    #la condición a evaluar
    contador <- contador + 1    #se van agregando el número de de casos entre -1 y +1 SD
  }
}
contador/ N  #valor se va acercando al 68.2 de la distribución normal a mayor N
```

Fundamentos de R

Vectores

Es una concatenación de datos de un mismo tipo, es un conjunto de datos en esencia. Por ejemplo:

- Un numérico sería 1, 2, 3, 5, 6, 7
- Uno de caracteres (character) podría ser “a”, “b”, “c”, “d”, “e”

#la función **c()** nos sirve para combinar elementos en un solo vector

```
vector_double <- c(5, 9, 48, 592)  
vector_double
```

#Funciones con **is.algo()** verifican que el elemento sea o no de de dicho tipo

```
is.numeric(vector_double)  
is.integer(vector_double)  
is.double(vector_double)
```

```
vector_integers <- c(5L, 9L, 50L, 60L)
```

```
is.numeric(vector_integers)  
is.integer(vector_integers)  
is.double(vector_integers)
```

#Debido a que numérico es una categoría que abarca a los double e integer es que tenemos que ambos son numeric, pero solamente el primero es double y el segundo es integer

```
vector_character <- c("Hola", "amigo", "cómo estás?")
```

```
is.numeric(vector_character)
```

```
is.character(vector_character)
```

```
#En el siguiente vector se ilustra el principio que dice que no se pueden combinar tipos de  
datos. El 45 se va a almacenar como "".
```

```
vectos_dif_tdatos <- c("jejej", "Fodor", 45, "Dennett")
```

```
#Función seq() para generar secuencias como 1:10. Genera secuencias de números  
seq(1, 10)
```

```
#Si se pone un tercer elemento se modifica la secuencia en función  
#de ese número. De 2 en 2, de 3 en 3, etc.  
seq(1, 10, 2)
```

```
vector_secuencia <- seq (1, 20, 5)  
vector_secuencia
```

```
#Función rep() para repetir un elemento N cantidad de veces. Si yo pongo  
#(5, 10) repito el 5 10 veces  
rep(5, 10)
```

```
vector_rep <- rep(5, 250)  
vector_rep
```

```
#También podemos repetir datos de tipo character, o incluso vectores  
rep(vector_rep, 2)  
rep("jeje", 10)
```

Uso de corchetes

#Dado el siguiente vector

```
W <- c("a", "b", "c", "d", "e")
```

```
W
```

#Corchetes me permiten acceder a los elementos del vector por su orden. Si pongo uno accede al primer elemento, 2 al segundo y así.

```
W[1]
```

```
W[2]
```

```
W[3]
```

#Si coloco números negativos puedo obtener todos los elementos menos el que corresponde a dicha posición. Si -1, entonces todos menos el primero.

```
W[-1]
```

```
W[-3]
```

```
W[-5]
```

#Esto anterior también es un vector en sí mismo

```
vector1 <- W[-1]
```

#Por medio de : podemos delimitar los elementos a los cuales accedemos en sentido

#de un rango

```
W[1:3]
```

```
W[1:2]
```

```
W[1:4]
```

#Si incluimos **c()** dentro de los [] podemos pedir datos específicos

#en el orden que queramos y la cantidad que queramos

```
W[c(1,2,5,3)]
```

```
W[c(5,4,3,2,1,5,5,5)]
```

```
W[c(1,1,1,1,1,1,5,5,5,5,5)]
```

#Al combinar esto con números negativos podemos filtrar los datos

W[c(-1,-2)]

W[c(-1,-3,-4,-5)]

W[c(-1,-5)]

W[c(-1:-2)]

W[c(-1:-4)]

W[c(-2:-4)]

W[c(1:4)]

Operaciones con vectores

Aritmética en R es más simple que en otros lenguajes ya que solamente necesita el símbolo de la operación para realizar cosas como $+$, $-$, $/$, $*$, etc. Lo mismo aplica con las operaciones lógicas. En casos donde el vector 1 es más corto que el vector 2, el vector 1 se recicla para poder ajustarse al vector 2, siempre y cuando 2 sea un múltiplo del vector 1.

Por ejemplo: Si el vector 1 es de 5 elementos, y el vector 2 es de 10 elementos y se quiere hacer una suma entre estos. Entonces, si 1 es [1,2,3,4,5] y 2 [1,2,3,4,5,6,7,8,9,10], lo que va a ocurrir es que del 1 al 5 se sumarán con los elementos de 1, y después del 6 al 10 nuevamente con 1 a 5 del vector 1.

En caso de que no sean múltiplos lo va a realizar pero el reciclado va a estar incompleto y el programa te lo hará saber. En nuestro caso anterior esto sería si por ejemplo el vector 1 fuera del 1 al 6, y el vector 2 se mantuviera igual.

Por último, como ya se ha mostrado anteriormente, los vectores de R también pueden ser indicados como una función de las 2 manera siguientes:

- $f(x)$
- $x \leftarrow f(y)$

#Primero definamos unos vectores

```
x <- c(seq(1:100))  
y <- c(seq(100, 200))
```

#**suma y resta** de vectores. Nota, cuando no son múltiplos se necesita almacenarlos en una variable para después ejecutarla

```
z <- x + y
```

```
z
```

```
z + y
```

```
z2 <- x-y
```

```
z
```

#**división** de vectores

```
a <- z/c(rep(10,100))
```

```
a
```

```
a1 <- a/x
```

```
a1
```

#**multiplicación** de vectores

```
b <- z * a
```

```
b
```

```
b1 <- b*z
```

```
b1
```

#**sqrt()** de vectores

```
sqrt(z+a)
```

```
sqrt(z-a)
```

```
sqrt(b+a)
```

```
sqrt(y+a)
```

```
#Algunas formas de acceder a los valores de los vectores por los medios convencionales de la  
programación
```

```
x1 <- rnorm(5)  
x2 <- rnorm(100, mean = 50, sd = 5 )
```

```
for (i in x1){  
  print(i)  
}
```

```
for(i in 1:5){  
  print(x1[i])  
}
```

```
for(j in x2){  
  print(j)  
}
```

```
for(j in 1: 50){  
  print(x2[j])  
}
```

```
#Esta siguiente por medio de print es posible pero ineficiente porque no es un ciclo  
print(x1[1])  
print(x1[2])  
print(x1[1:3])
```

```
#Operaciones con y sin vectores. Más sencillo con vectores, y más rápido, debido a que son  
menos pasos como se ve a continuación con estas multiplicaciones.
```

```
#Dadas las siguientes definiciones  
N <- 100  
a <- rnorm(N)
```

```
b <- rnorm(N)
d <- rep(NA, N)
```

#Vectores

```
c <- a * b
```

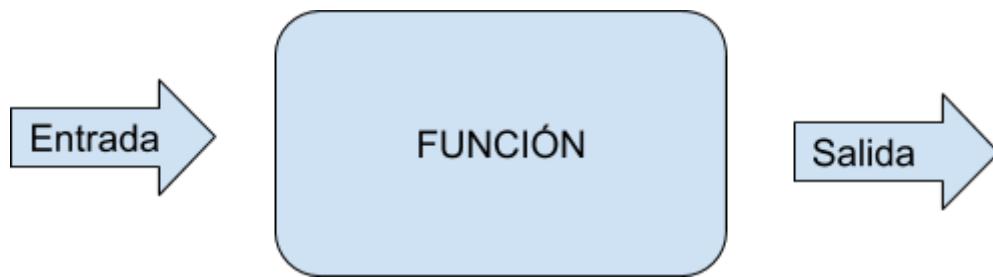
#Convencionales

#A diferencia del anterior, aquí se va pasando por cada uno de los elementos del vector y se van multiplicando y almacenado en otro vector. Es por ello que es más lento, sobretodo cuando más elemento tienen los vectores.

```
for (i in 1:N){
  d[i] <- a[i] * b[i]
}
```

Funciones en R

Una función puede entenderse como una caja a la cual uno le añade una entrada y da una salida. Esta caja tiene una serie de instrucciones o pasos que sigue para manipular la entrada de x tipo y dar una salida de y tipo.



En caso de querer saber qué hace cada función por medio del símbolo ?. Por ejemplo, si uno quiere saber qué hace rnorm(), uno escribe **?rnorm()** y R nos dice en help lo siguiente:

R: The Normal Distribution ▾ Find in Topic

Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

`x, q` vector of quantiles.

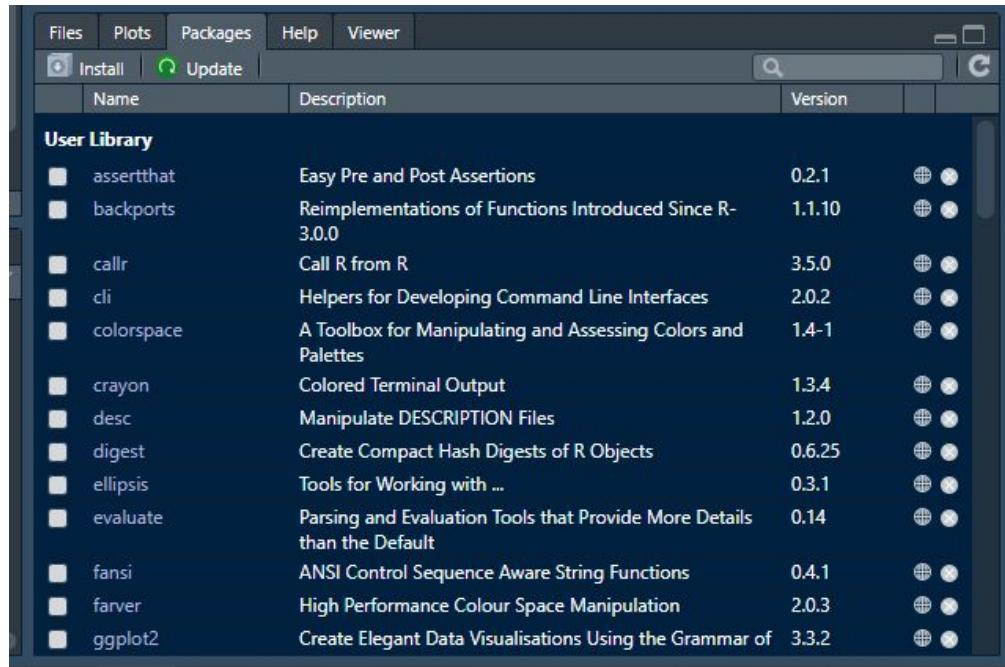
03:12 p.m.

This screenshot shows the R documentation for the 'Normal' distribution. The title is 'The Normal Distribution'. Under 'Description', it says: 'Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`'. Under 'Usage', four functions are listed: `dnorm`, `pnorm`, `qnorm`, and `rnorm`. Each function has parameters: `mean` and `sd` (both default to 0 and 1 respectively), and optional `log` and `lower.tail` (both default to FALSE). Under 'Arguments', it specifies that `x, q` are vectors of quantiles. The bottom right corner shows the time as 03:12 p.m.

Paquetes de R

Los paquetes de R no son más que una colección de funciones, datos, y código compilados. La librería es donde se encuentran almacenados estos paquetes.

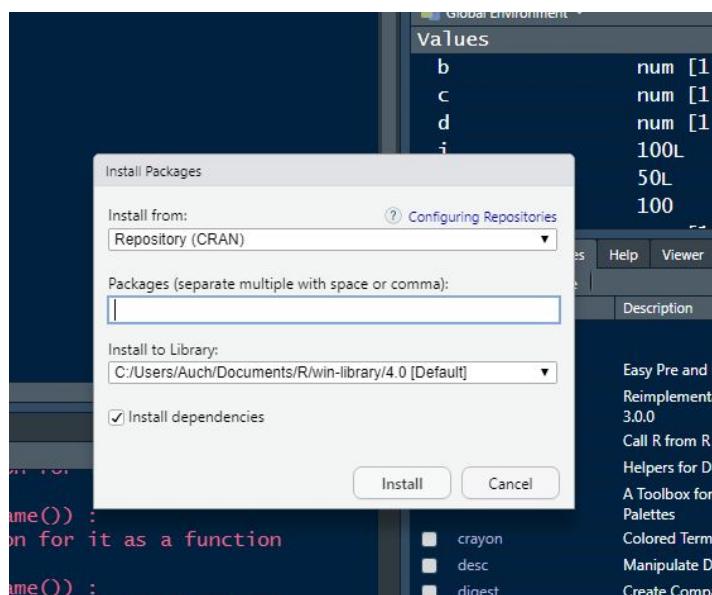
Se encuentran en:



The screenshot shows the RStudio interface with the 'Packages' tab selected in the top menu bar. Below the menu, there are two buttons: 'Install' and 'Update'. The main area displays a table titled 'User Library' with columns for 'Name', 'Description', 'Version', and two circular icons. The table lists various R packages, such as assertthat, backports, callr, cli, colorspace, crayon, desc, digest, ellipsis, evaluate, fansi, farver, and ggplot2, along with their descriptions and versions.

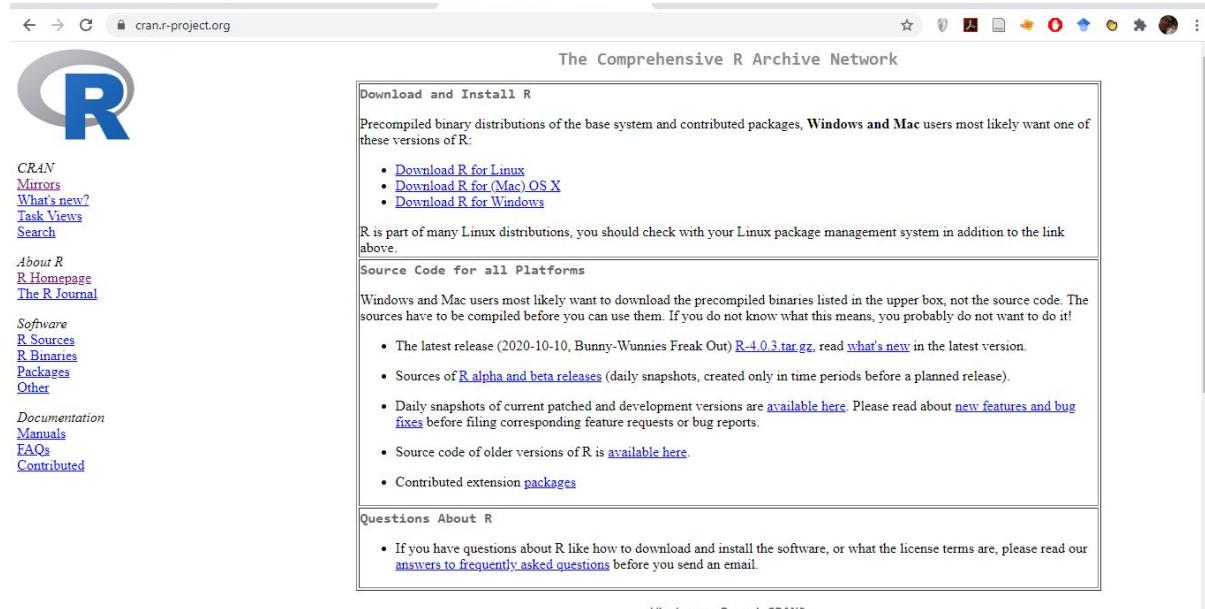
Name	Description	Version		
assertthat	Easy Pre and Post Assertions	0.2.1	●	●
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.10	●	●
callr	Call R from R	3.5.0	●	●
cli	Helpers for Developing Command Line Interfaces	2.0.2	●	●
colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	1.4-1	●	●
crayon	Colored Terminal Output	1.3.4	●	●
desc	Manipulate DESCRIPTION Files	1.2.0	●	●
digest	Create Compact Hash Digests of R Objects	0.6.25	●	●
ellipsis	Tools for Working with ...	0.3.1	●	●
evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.14	●	●
fansi	ANSI Control Sequence Aware String Functions	0.4.1	●	●
farver	High Performance Colour Space Manipulation	2.0.3	●	●
ggplot2	Create Elegant Data Visualisations Using the Grammar of	3.3.2	●	●

Si uno hace click en install aparece lo siguiente



The screenshot shows the RStudio interface with the 'Install Packages' dialog box open. The dialog has several fields: 'Install from:' set to 'Repository (CRAN)', 'Packages (separate multiple with space or comma):' with an empty input field, 'Install to Library:' set to 'C:/Users/Auch/Documents/R/win-library/4.0 [Default]', and a checked 'Install dependencies' checkbox. In the background, the R console shows some code execution, and the Packages tab of the library view is visible.

CRAN es the Comprehensive R Archive Network, que es donde se encuentran los paquetes que van generando la comunidad de R. El link <https://cran.r-project.org/>

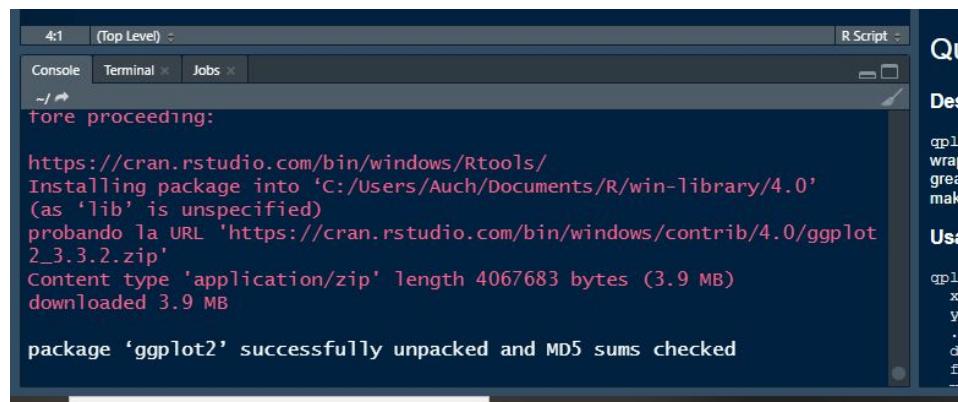


The screenshot shows the homepage of The Comprehensive R Archive Network (CRAN) at cran.r-project.org. The page features a large R logo on the left. On the right, there's a main content area with sections for "Download and Install R", "Source Code for all Platforms", and "Questions About R". The "Download and Install R" section provides links for Linux, Mac OS X, and Windows. The "Source Code for all Platforms" section contains a bulleted list of instructions for users. The "Questions About R" section has one bullet point. At the bottom right, there's a link to "What are R and CRAN?".

Un ejemplo de esto es las siguientes líneas de código que sirven para descargar y activar el paquete ggplot2:

```
install.packages("ggplot2")
```

#Genera que ocurra esto en la consola y se descargue el paquete de CRAN



The screenshot shows an RStudio interface with a dark theme. The console tab is active, displaying the command "install.packages("ggplot2")" and its output. The output shows the package being downloaded from CRAN and successfully unpacked. A tooltip "To proceed:" is visible above the console. To the right, there are tabs for "Qu", "Des", and "Usa", which are partially visible.

```
4:1 (Top Level) R Script
Console Terminal Jobs
-/→ To proceed:
https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/Auch/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
probando la URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/ggplot
2_3.3.2.zip'
Content type 'application/zip' length 4067683 bytes (3.9 MB)
downloaded 3.9 MB

package 'ggplot2' successfully unpacked and MD5 sums checked
```

```
library(ggplot2)
```

Análisis de estado financiero

Caso a resolver

Escenario: Eres un Data Scientist trabajando para una empresa que da consultoría. Uno de tus compañeros del departamento de Auditorías te ha pedido que le ayudes a evaluar los estados financieros de la organización X.

Te han proporcionado dos vectores de datos: Ingresos mensuales y Gastos mensuales del año fiscal en cuestión. Tu trabajo es obtener las siguientes métricas:

- Utilidad para cada mes
- Utilidad Despues de Impuestos (UDI) para cada mes (la tasa es del 30%)
- Margen de Utilidad para cada mes - igual a la UDI dividida entre los ingresos
- Buenos meses - donde la UDI para el mes fue mayor que el promedio del año
- Malos meses - donde la UDI para el mes fue menor que el promedio del año
- Mejor mes - donde la UDI fue la máxima para el año
- Peor mes - donde la UDI fue la mínima para el año

Programación en R: A-Z

© SuperDataScience

Todos los resultados deben de ser presentados como vectores

Los resultados deben de ser calculados usando una precisión de \$0.01, pero deben de ser presentados en unidades de \$1,000 (1 K) sin puntos decimales

Los resultados para el margen de utilidad promedio deben de ser presentados en porcentaje (%) sin puntos decimales

Nota: Tu compañero te ha comentado que está bien que el impuesto para cualquier mes sea negativo (en términos contables, el impuesto negativo será utilizado como impuesto diferido)

Funciones extra a usar, revisar con ?

round()
mean()
max()
min()

NUEVAS FUNCIONES

Resolución del ejercicio

#Datos del ejercicio

```
ingresos <- c(14574.49, 7606.46, 8611.41, 9175.41, 8058.65, 8105.44, 11496.28, 9766.09,
```

```
10305.32, 14379.96, 10713.97, 15433.50)
```

```
gastos <- c(12051.82, 5695.07, 12319.20, 12089.72, 8658.57, 840.20, 3285.73, 5821.12,
```

```
6976.93, 16618.61, 10054.37, 3803.96)
```

?round()

?min()

?max()

?mean()

#Calcular Utilidad como la Diferencia de Ingreso y Gasto

```
utilidad <- ingresos - gastos
```

```
utilidad
```

#Calcular el Impuesto como el 30% de la Utilidad y Redondear a 2 Puntos Decimales

```
impuesto <- round(0.30 * utilidad, 2)
```

```
impuesto
```

#Calcular la Utilidad Despues de Impuestos o UDI

```
utilidad.despues.de.impuestos <- utilidad - impuesto
```

```
utilidad.despues.de.impuestos
```

#Calcular el Margen de Utilidad como Utilidad Despues de Impuestos sobre Ingresos

#Redondear a 2 Puntos Decimales, Luego Multiplicar por 100 para obtener el %

```
margen.de.utilidad <- round(utilidad.despues.de.impuestos/ ingresos, 2) * 100
```

```
margen.de.utilidad
```

#Calcular el promedio para los 12 meses de la Utilidad Despues de Impuestos

```
promedio.utilidad.despues.de.impuestos <- mean(utilidad.despues.de.impuestos)
```

```
promedio.utilidad.despues.de.impuestos
```

```
#Encuentra los Meses Utilidad Despues de Impuestos por Encima de la Media  
meses.buenos <- utilidad.despues.de.impuestos > promedio.utilidad.despues.de.impuestos  
meses.buenos
```

```
#Los Meses Malos son el Opuesto a los Meses Buenos !  
meses.malos <- !meses.buenos
```

```
meses.malos
```

```
#El Mejor Mes es Donde la Utilidad Despues de Impuestos es Igual al Máximo
```

```
#Aquí compara capada valor de utilidad.despues de impuestos con el máximo de ese vector
```

```
#Esto es gracias al reciclado de valores, el valor máximo se repite las 12 veces
```

```
mejor.mes <- utilidad.despues.de.impuestos == max(utilidad.despues.de.impuestos)
```

```
mejor.mes
```

```
#El Peor Mes es Donde la Utilidad Despues de Impuestos es Igual al Mínimo
```

```
#Mismos principios que en el paso anterior
```

```
peor.mes <- utilidad.despues.de.impuestos == min(utilidad.despues.de.impuestos)
```

```
peor.mes
```

```
#Convierte Todos los Cálculos a Unidades de Mil Dólares
```

```
ingresos.1000 <- round(ingresos / 1000, 0)
```

```
gastos.1000 <- round(gastos / 1000, 0)
```

```
utilidad.1000 <- round(utilidad / 1000, 0)
```

```
utilidad.despues.de.impuestos.1000 <- round(utilidad.despues.de.impuestos/1000, 0)
```

```
#Imprime los Resultados
```

```
print(ingresos.1000)
```

```
print(gastos.1000)
```

```
print(utilidad.1000)
```

```
print(utilidad.despues.de.impuestos.1000)
```

```
print(margen.de.utilidad)
```

```
print(meses.buenos)
```

```
print(meses.malos)
```

```
print(mejor.mes)
```

```
print(peor.mes)

#Si se agrega el siguiente código se obtiene una tabla o matriz de los datos
M <- rbind(
  ingresos.1000,
  gastos.1000,
  utilidad.1000,
  utilidad.despues.de.impuestos.1000,
  margen.de.utilidad,
  meses.buenos,
  meses.malos,
  mejor.mes,
  peor.mes
)
```

M

Matrices

Los vectores solamente corren en una dimensión, sin embargo las matrices lo hacen en 2 dimensiones. Como lo muestra la siguiente imagen



Como se ve, arriba tenemos un vector que corre hacia la derecha con valores de 1,2,3,4,5. Sin embargo nuestra matriz lo hace también hacia abajo, generando así un plano, o una tabla de valores.

Luego, si uno quiere acceder a un valor de la matriz uno se topa con un problema, con el hecho de que hay tanto filas como columnas, y por ende uno no puede indicar el valor deseado como se hacía con un vector. Si uno pone $A[3]$, entonces es ambigua la instrucción ya que hay N valores en 3 como columna o 3 como fila; dependiendo de las dimensiones de nuestra matriz. Esto se ve en la siguiente imagen:

	1	2	3	4	5	
1	44	44	44	44	44	A[4]
2	44	44	44	44	44	A[3]
3	44	44	44	0	44	A[3,4]

INDEXACIÓN

Programación en R: A-Z © SuperDataScience

Si uno quiere seleccionar un valor se debe dar dos posiciones, una relativa a las columnas, y otra a las filas. En el caso de arriba se ve que para indicar el cero uno debe dar las coordenadas A[3, 4]. El primero indica la fila, y el segundo la columna, esto se llama indexación.

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	A[4]
[2,]	44	44	44	44	44	A[3]
[3,]	44	44	44	0	44	A[3,4]

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]	
[1,]	44	44	44	44	44	~~A[4]~~
[2,]	44	44	44	44	44	~~A[3]~~

<tbl_r cells="7" ix="4" maxcspan="1" maxr

que las matrices solamente aceptan datos de un solo tipo, ya sea double, integer, character, etc.

Construyendo tu primer matriz

Hay varias formas de generar matrices, una de estas es mediante el uso de la función **matrix()**. Lo que hace es tomar un vector y acomodarlo en sentido de una matriz, lo cual puede no ser muy bueno porque se pierde el orden de nuestros datos. Si tenemos un vector de 1:12, entonces:

1	4	7	10
2	5	8	11
3	6	9	12

Otra manera es mediante la función **rbind()**, r es de row, esta une los vectores de tal manera que un vector se toma como fila y se van apilando formando la matriz. Si fuese más corto un vector que otro se aplica la ley del reciclado de vectores.

Fila 1	Fila 1	Fila 1	Fila 1
Fila 2	Fila 2	Fila 2	Fila 2
Fila 3	Fila 3	Fila 3	Fila 3

Una última es mediante **cbind()**, c es de column, y lo que hace es que une los vectores tomándolos como columnas. Igualmente aplica la ley del reciclado.

Columna 1	Columna 2	Columna 3	Columna 4
Columna 1	Columna 2	Columna 3	Columna 4
Columna 1	Columna 2	Columna 3	Columna 4

```
# matriz usando matrix(), te pide nrow y ncol
```

```
mis.datos <- 1:25
```

```
mis.datos
```

```
#Ojo, es importante que el número de datos sea igual a nrow * ncol
```

```
M <- matrix(mis.datos, nrow = 5, ncol = 5)
```

```
M
```

```
#Si quisiéramos un valor de esta matriz usamos lo siguiente
```

```
M[1,3]
```

```
M[2,4]
```

```
M[1,1]
```

```
M[3,4]
```

```
#matriz usando rbind()
```

```
v1 <- c(1,2,3,4,5)
```

```
v2 <- c(10,11,12,13,14)
```

```
v3 <- c(20,21,22,23,24)
```

```
M2 <- rbind(v1,v2, v3)
```

```
M2
```

```
#matriz con cbind()
```

```
M3 <- cbind(v1,v2,v3)
```

```
M3
```

Nombrando las dimensiones

Uno puede ponerle nombres a las columnas. Supongamos que hacemos lo siguiente

	“A”	“B”	“C”	“D”
“Dinosaurio”	Dino 1	Dino 2	Dino 3	Dino 4
“Ave”	Ave 1	Ave 2	Ave 3	Ave 4
“Reptil”	Reptil 1	Reptil 2	Reptil 3	Reptil 4
“Mamífero”	Mamífero 1	Mamífero 2	Mamífero 3	Mamífero 4

Hemos nombrado las filas y columnas, si uno quisiera obtener el mamífero 3 entonces uno pondría $M[“Mamífero”, C]$. Igual que como se hacía cuando nuestras filas y columnas eran números y un ponía la coordenada entre [X, Y] para obtener x o y elemento de la matriz. Es importante mencionar que si nuestra matriz ya tenía previamente otros nombres o símbolos asignados a sus columnas y filas, entonces se pueden seguir usando. Incluso se pueden combinar con los nuevos, uno podría poner $M[“Mamífero”, 3]$ y sería correcto si previamente teníamos número de en lugar de nuestras letras.

También se puede aplicar a los vectores como el siguiente

“A”	“B”	“C”	“D”
Dino 1	Dino 2	Dino 3	Dino 4

Aquí uno pondría $V[“B”]$ si quisiera acceder al elemento Dino 3

Colnames() y rownames()

```
#Primero hacemos un vector
```

```
Pau <- 1:5
```

```
Pau
```

```
#Aquí se hace que el vector Pau siempre sea acompañado por el vector names()
```

```
names(Pau) <- c("a", "b", "c", "d", "e")
```

```
#Ahora cuando se corre Pau se obtiene que el vector original de 1:5 esté acompañado por  
c("a", "b", "c", "d", "e")
```

```
Pau
```

```
#También podemos quitar los nombres acompañantes mediante NULL
```

```
names(Pau) <- NULL
```

```
Pau
```

```
#Ahora a nombrar matrices
```

```
#Se almacena primero un vector simple en un objeto
```

```
V1 <- rep(c("a", "b", "c", "d", "e"), each = 3)
```

```
#Luego se vuelve este una matriz, se especifican las dimensiones y se almacena como otro  
objeto
```

```
Dennett <- matrix(V1, 3, 3)
```

```
Dennett
```

```
#Posteriormente le ponemos los nombres a las filas y columnas por las funciones rownames()  
y colnames() respectivamente
```

```
rownames(Dennett) <- c("Dinosaurio", "Mamífero", "Reptil")
```

```
colnames(Dennett) <- c("A", "B", "C")
```

```
Dennett
```

#Aplicando el principio que se mencionó antes podemos tomar cosas de la matriz aprovechando todos los nombres que han tenido las filas y columnas

Dennett["Reptil", "A"]

Dennett[3,1]

#Como lo hicimos anteriormente, igualmente se pueden quitar los nombres por medio de NULL

rownames(Dennett) <- NULL

colnames(Dennett) <- NULL

Operaciones con matrices

#En el anexo vienen los datos que se usan en el curso. No son míos y todos los derechos son de www.superdatascience.com.

#Las operaciones en R son muy simples, como con los vectores solamente necesitamos el símbolo

```
tiros_anotados/juegos  
round(tiros_anotados /juegos, 1)
```

#Esto solamente lo hice para ir analizando los datos de la matriz más a fondo

```
mean(tiros_anotados)  
max(tiros_anotados)  
min(tiros_anotados)
```

```
minutos_jugados  
round(minutos_jugados/juegos, 1)
```

```
mean(minutos_jugados)  
max(minutos_jugados)  
min(minutos_jugados)
```

#También, aunque no tiene mucho sentido para estos datos, es posible hacer otro tipo de operaciones

```
minutos_jugados * tiros_anotados  
sqrt(minutos_jugados)  
minutos_jugados + tiros_anotados
```

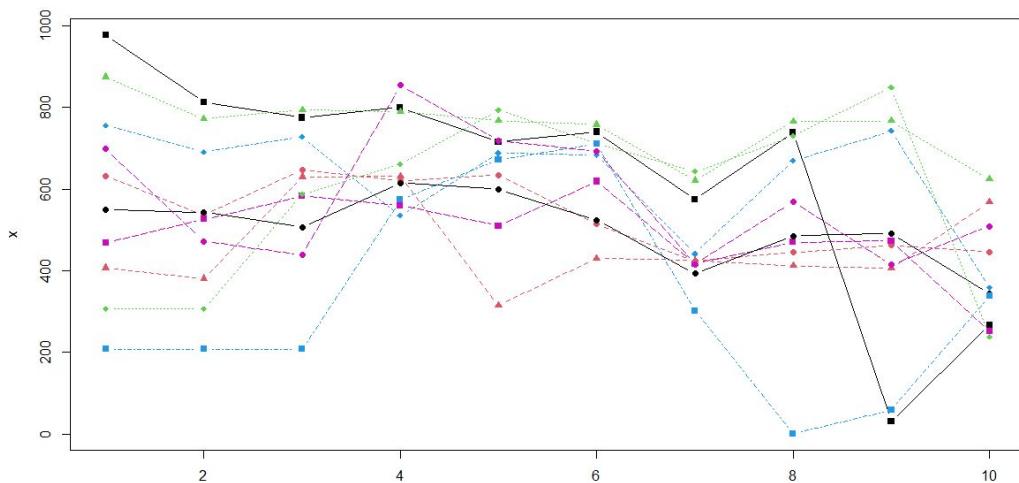
Visualizando con matplotlib()

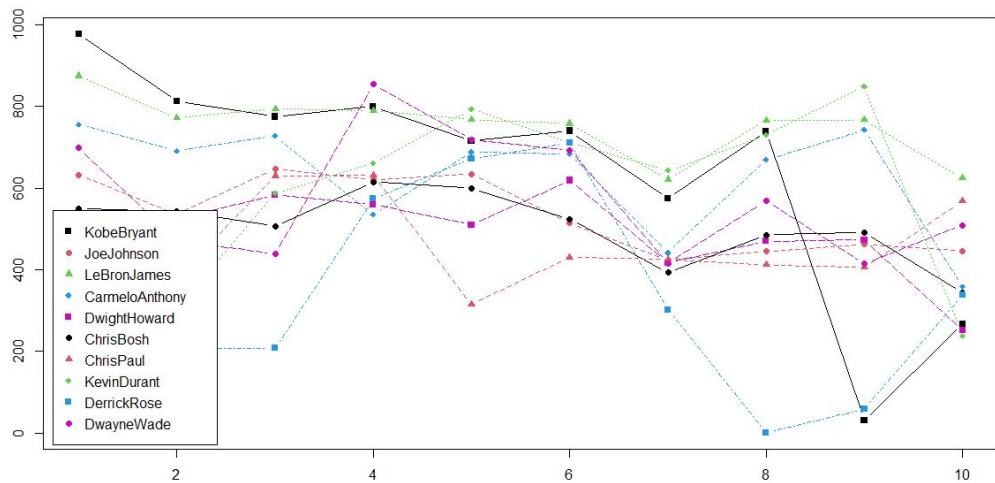
1. Solo tiros anotados

```
# t() nos permite transponer matrices, que no es otra cosa que poner lo que está como filas  
como columnas y viceversa  
x <- t(tiros_anotados)
```

```
#Una vez con la matriz transpuesta podemos usar matplotlib() que la necesita así  
#El type= indica que vamos a usar puntos y líneas  
#pch= indica el tipo puntos a usar  
#col= finalmente nos dicta los colores, donde 1:4 indica el rango de los colores a usar (son  
cíclicos por eso se especifica)  
matplotlib(x, type="b", pch=15:18, col=c(1:4, 6))
```

```
#Luego, como nuestros colores no indican el jugador necesitamos una leyenda que lo marque  
#Primero se pone la posición  
#Posteriormente se pone inset= para indicar la distancia de la posición inicial indicada  
previamente y lo que se quiere que indique la leyenda  
#Luego ponemos los colores de manera idéntica al matplotlib(), además del pch  
#Finalmente solamente se indica la posición  
legend("bottomleft", inset = .01, legend = jugadores, col=c(1:4, 6), pch=15:18, horiz = F)
```





2. Tiros anotados/tiros intentados

t() nos permite transponer matrices, que no es otra cosa que poner lo que está como filas como columnas y viceversa

```
x <- t(tiros_anotados/tiros_intentados)
```

#Una vez con la matriz transpuesta podemos usar matplot() que la necesita así

#El type= indica que vamos a usar puntos y líneas

#pch= indica el tipo puntos a usar

#col= finalmente nos dicta los colores, donde 1:4 indica el rango de los colores a usar (son cílicos por eso se especifica)

```
matplot(x, type="b", pch=15:18, col=c(1:4, 6))
```

#Luego, como nuestros colores no indican el jugador necesitamos una leyenda que lo marque

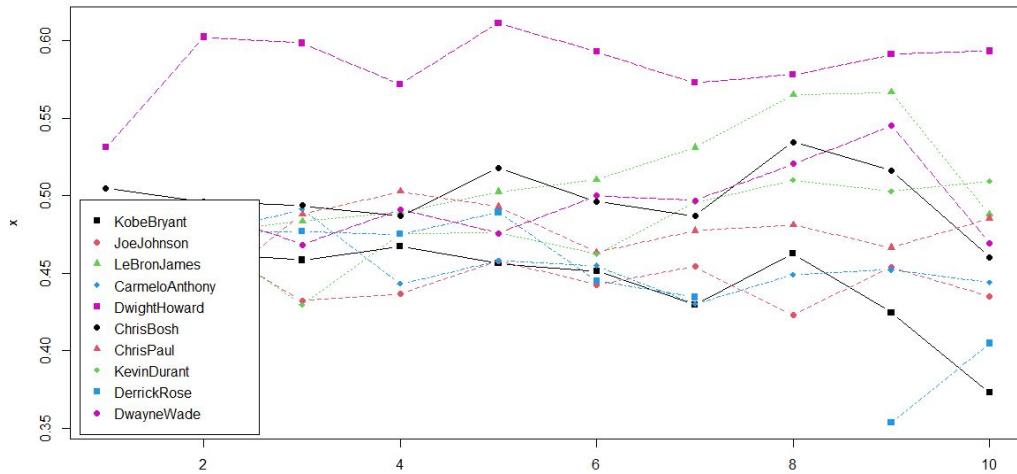
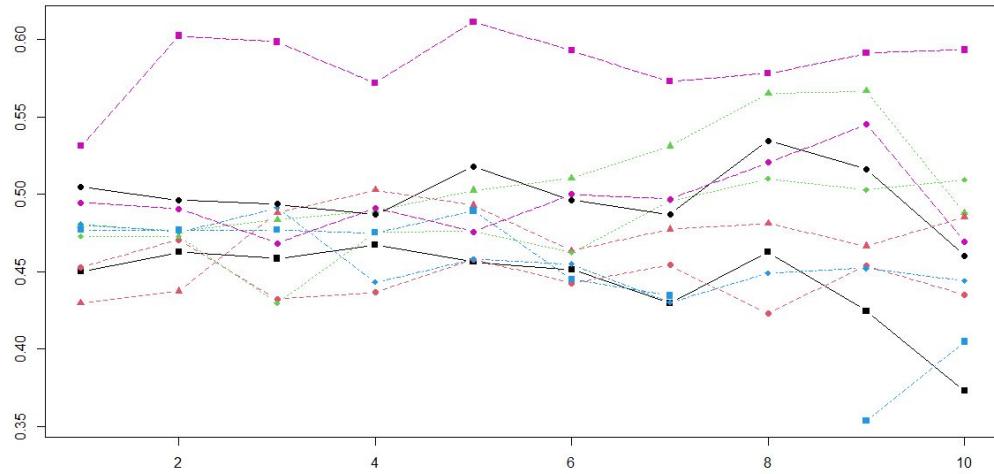
#Primero se pone la posición

#Posteriormente se pone inset= para indicar la distancia de la posición inicial indicada previamente y lo que se quiere que indique la leyenda

#Luego ponemos los colores de manera idéntica al matplot(), además del pch

#Finalmente solamente se indica la posición

```
legend("bottomleft", inset = .01, legend = jugadores, col=c(1:4, 6), pch=15:18, horiz = F)
```



3. Tiros anotados/juegos

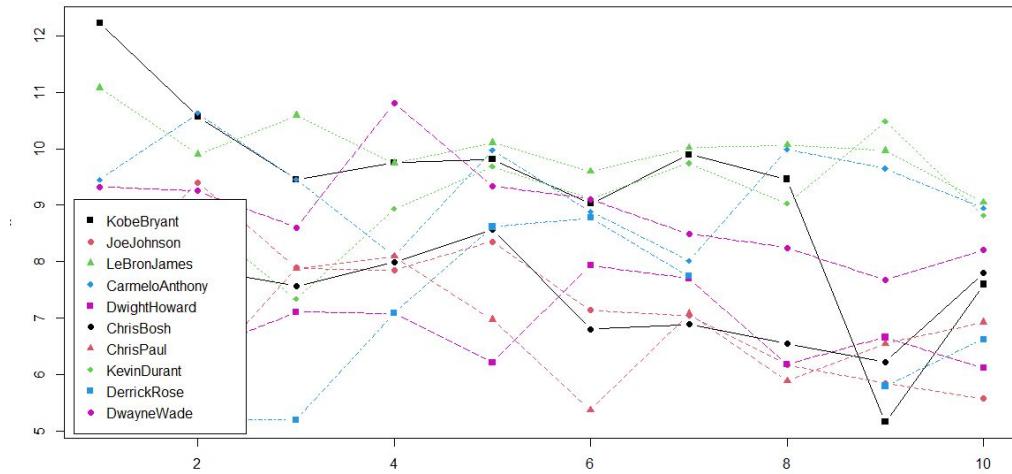
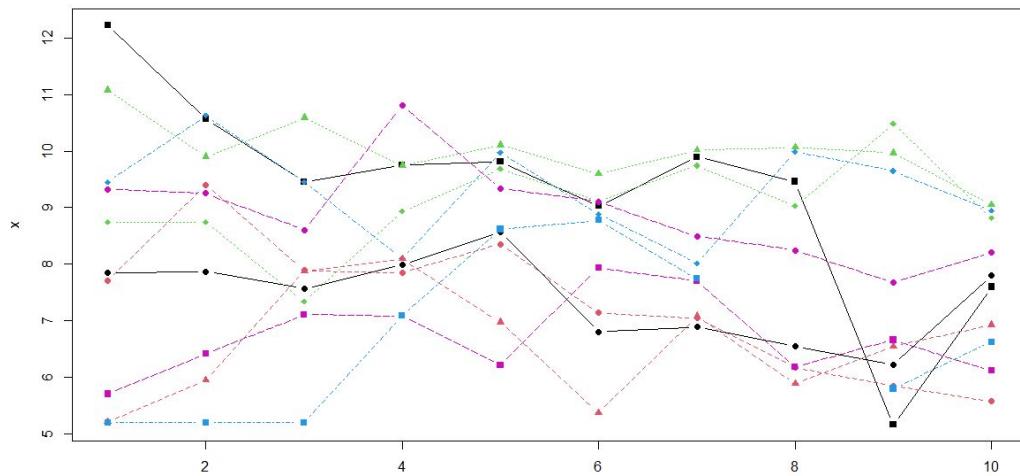
```
# t() nos permite transponer matrices, que no es otra cosa que poner lo que está como filas
como columnas y viceversa
x <- t(tiros_anotados/juegos)
```

```
#Una vez con la matriz transpuesta podemos usar matplotlib() que la necesita así
#El type= indica que vamos a usar puntos y líneas
#pch= indica el tipo puntos a usar
#col= finalmente nos dicta los colores, donde 1:4 indica el rango de los colores a usar (son
cíclicos por eso se especifica)
matplotlib(x, type="b", pch=15:18, col=c(1:4, 6))
```

```

#Luego, como nuestros colores no indican el jugador necesitamos una leyenda que lo marque
#Primero se pone la posición
#Posteriormente se pone inset= para indicar la distancia de la posición inicial indicada
#previamente y lo que se quiere que indique la leyenda
#Luego ponemos los colores de manera idéntica al matplot(), además del pch
#Finalmente solamente se indica la posición
legend("bottomleft", inset = .01, legend = jugadores, col=c(1:4, 6),pch=15:18, horiz = F)

```



Subconjuntos de datos

#Subconjuntos (subsettings), literalmente son conjuntos formados de un conjunto de datos, ya sea un vector o una matriz

```
vector <- c(1,2,3,4,5,6,7,8,9,10)  
vector
```

#Por medio de nombre del vector y las posiciones uno puede generar un subconjunto del vector

```
vector[c(1,4)]  
vector[c(2,3,5,7)]  
vector[1]
```

#Mismo principio aplica para las matrices, pero hay que especificar las filas y columnas

juegos

```
juegos[1:3,5:10]  
juegos[1:2, 1:2]  
juegos[1:5, 6:8]  
juegos[, c("2007", "2010")]  
juegos[,c("2005", "2007")]  
juegos[c("KobeBryant", "LeBronJames"),]  
juegos[c("ChrisPaul", "DwyaneWade", "CarmeloAnthony"),]
```

#Pero si uno hace lo siguiente uno transforma su subconjunto en un vector a partir de una matriz. Todos los anteriores tomaban o la matriz o el vector y el subconjunto era del mismo tipo

```
juegos[1,]
```

```
is.matrix(juegos[1,])  
is.matrix(juegos[1,1])
```

```
#Esto es debido a que damos solamente una dimensión
```

```
#Para evitar esto se hace lo siguiente y nos dará una matriz nuevamente
```

```
juegos[1,,drop=FALSE]
```

```
is.matrix(juegos[1,,drop=FALSE])
```

```
juegos[1,1,,drop=F]
```

```
is.matrix(juegos[1,1,drop=FALSE])
```

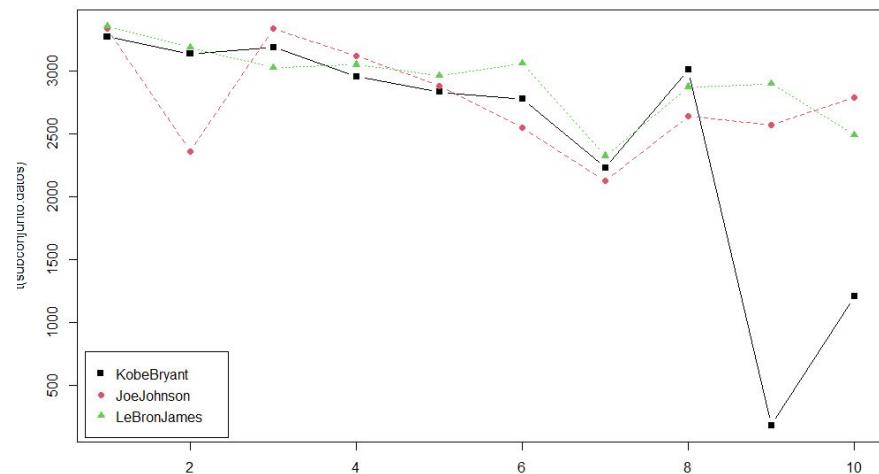
Visualizando los subconjuntos

```
#Visualizar los subconjuntos
```

```
subconjunto.datos <- minutos_jugados[1:3,]
```

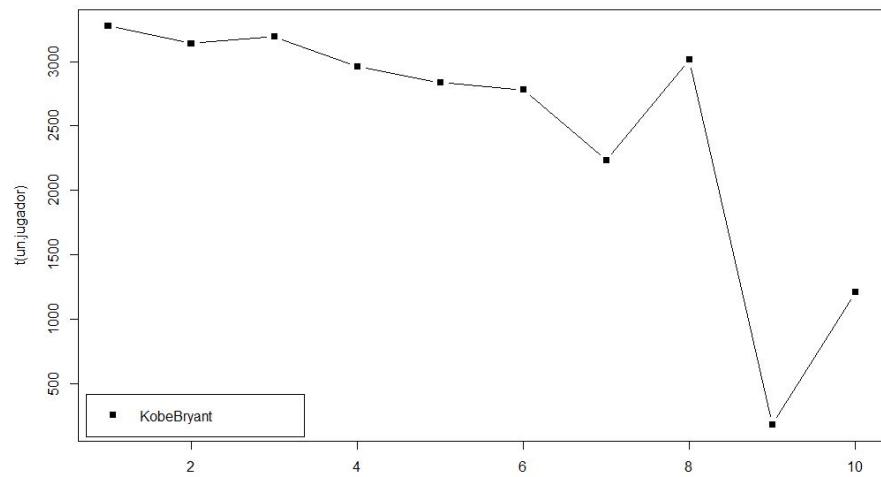
```
matplot(t(subconjunto.datos), type = "b", pch = 15:18, col = c(1:4, 6))
```

```
legend("bottomleft", inset = .01, legend = jugadores[1:3], pch = 15:18, col = c(1:4, 6))
```



```
#Si quisiéramos solamente ver al primer jugador. Es importante aquí indicar por medio de drop que no es un vector sino una matriz, esto debido a que matplot() necesita matrices para funcionar
```

```
un.jugador <- minutos_jugados[1,,drop=FALSE]
matplotlib(t(un.jugador), type = "b", pch = 15:18, col = c(1:4, 6) )
legend("bottomleft",inset = .01, legend = jugadores[1], pch = 15:18, col = c(1:4, 6))
```



Creando funciones

#Una función te permite almacenar todo un código en una frase en que podemos escribir fácilmente.

#Aquí se hace uso de **function()** para generar la función

#Se agregan **parámetros en ()** que permiten flexibilidad a la función. De tal modo que uno puede pedir los datos, y las filas a usar. Una vez que las da estas se acoplan al código dado.

#En nuestro caso se toma data y este se almacena en datos y estos luego son transpuestos para poder ser usados por `matplot()`

#Así mismo se piden filas para que esas sean las tomadas del data y usadas en el código.

Además se pone un valor default tal que se toman automáticamente las fila 1:10 si solamente se da el data

```
mi_función <- function(data, filas=1:10){  
  datos <- data[filas,,drop=FALSE]  
  matplot(t(datos), type = "b", pch = 15:18, col = c(1:4, 6) )  
  legend("bottomleft",inset = .01, legend = jugadores[filas], pch = 15:18, col = c(1:4, 6))  
}
```

```
mi_función(sueldos/juegos, 1)
```

#Esta nos permite hacer cualquiera de las gráficas que hemos hecho antes de manera rápida y sencilla

Práctica basketball de tiros libres

Te han proveído datos de dos estadísticas más del juego

- Tiros Libres
- Tiros Libres Intentados

Tienes que crear tres gráficos que muestren los siguientes insights:

- Tiros Libres Intentados por juego
- Precisión en Tiros Libres
- Estilo de Juego del Jugador (preferencia de 2 vs 3 puntos) excluyendo los Tiros Libres*

*Cada Tiro Libre cuenta como 1 punto

Los datos han sido suministrados en forma de vectores. Vas a tener que crear dos matrices antes de proceder con el análisis

#Primero debemos generar las matrices de tiros libres y tiros libres intentados

#Matriz tiros libres

```
tiros_libres <- rbind(KobeBryant_TL, JoeJohnson_TL, LeBronJames_TL,  
CarmeloAnthony_TL, DwightHoward_TL,  
ChrisBosh_TL, ChrisPaul_TL, KevinDurant_TL, DerrickRose_TL,  
DwyaneWade_TL)  
rm (KobeBryant_TL, JoeJohnson_TL, LeBronJames_TL, CarmeloAnthony_TL,  
DwightHoward_TL,  
ChrisBosh_TL, ChrisPaul_TL, KevinDurant_TL, DerrickRose_TL, DwyaneWade_TL)  
colnames(tiros_libres) <- temporadas  
rownames(tiros_libres) <- jugadores  
tiros_libres
```

#Matriz tiros libres intentados

```
tiros_libres_intentados <- rbind(KobeBryant_TLI, JoeJohnson_TLI, LeBronJames_TLI,  
CarmeloAnthony_TLI, DwightHoward_TLI, ChrisBosh_TLI,  
ChrisPaul_TLI, KevinDurant_TLI,  
DerrickRose_TLI, DwyaneWade_TLI)  
rm(KobeBryant_TLI, JoeJohnson_TLI, LeBronJames_TLI,
```

```
CarmeloAnthony_TLI, DwightHoward_TLI, ChrisBosh_TLI, ChrisPaul_TLI,  
KevinDurant_TLI,
```

```
DerrickRose_TLI, DwyaneWade_TLI)
```

```
colnames(tiros_libres_intentados) <- temporadas
```

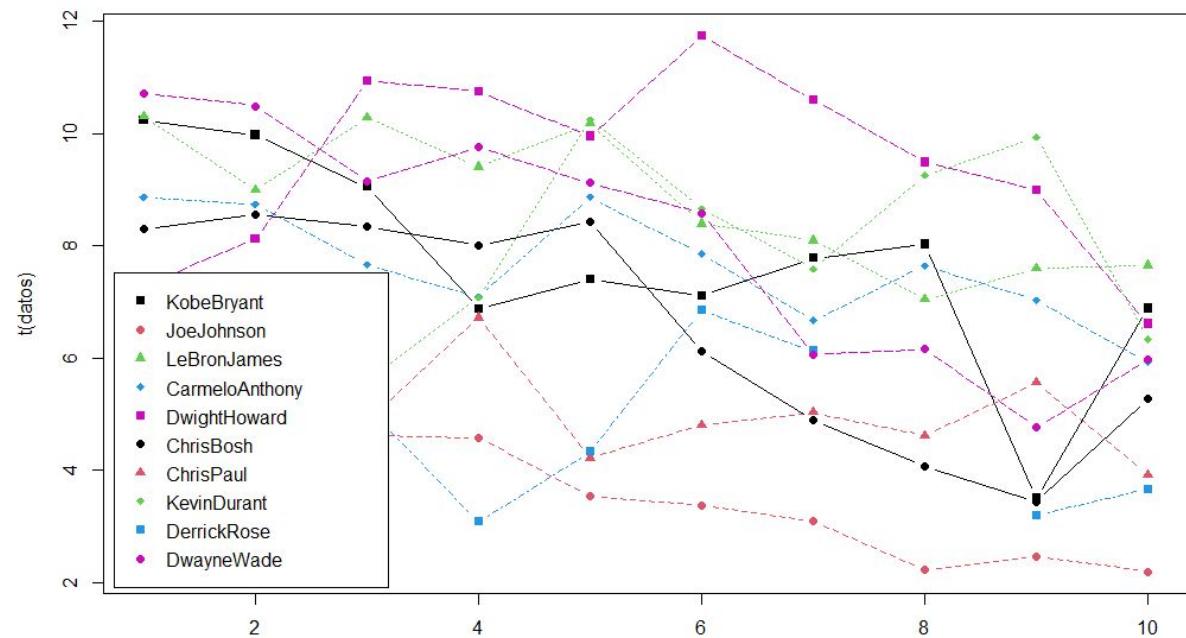
```
rownames(tiros_libres_intentados) <- jugadores
```

```
tiros_libres_intentados
```

```
#Tiros libres intentados por juego
```

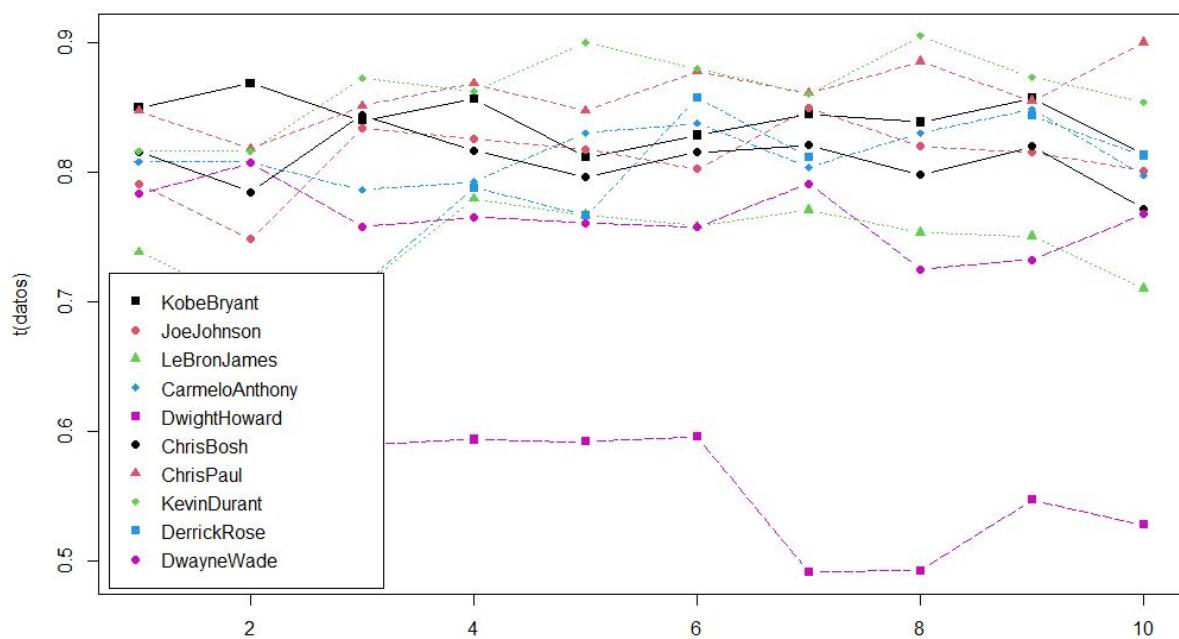
```
mi_función(tiros_libres_intentados/juegos)
```

```
##
```

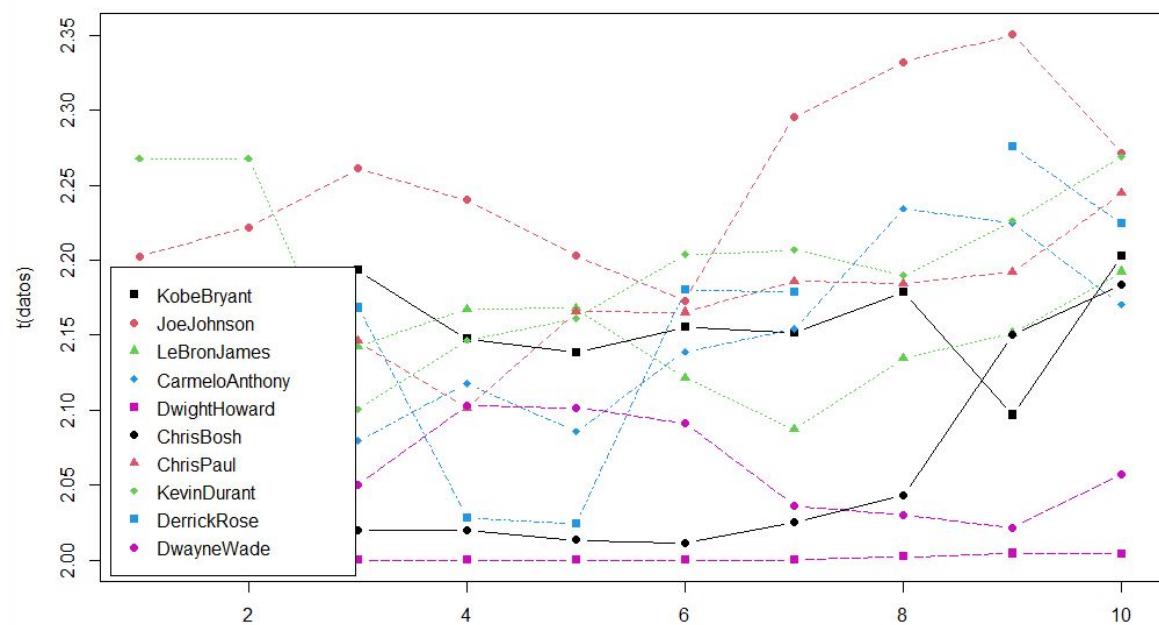


```
#Precisión en tiros libre
```

```
mi_función(tiros_libres/tiros_libres_intentados)
```



#Estilo del jugador excluyendo tiros libres
 $\text{mi_función}((\text{puntos} - \text{tiros_libres}) / \text{tiros_anotados})$



Anexos

Esta base de datos no es mía y los scripts pertenecen a www.superdatascience.com.

Solamente se deben correr para poder hacer las actividades de las secciones por si alguien quisiera hacerlo. De todos modos es muy recomendable tomar el curso en Udemy (o en la página de <http://www.superdatascience.com>), es excelente, y no solamente revisar estas notas.

#Comentarios:

#Las temporadas están etiquetadas con el primer año de la temporada

#Ejemplo: la temporada 2012-2013 es presentada como 2012

#

Notas adicionales:

#Kevin Durant: 2006 - Datos de temporada universitaria

#Kevin Durant: 2005 - Creada con datos del 2006

#Derrick Rose: 2012 - No jugó

#Derrick Rose: 2007 - Datos de temporada universitaria

#Derrick Rose: 2006 - Creada con datos de temporada 2007

#Derrick Rose: 2005 - Creada con datos de temporada 2007

#Temporadas

```
temporadas <- c("2005","2006","2007","2008","2009","2010","2011","2012","2013","2014")
```

#Jugadores

```
jugadores <-
```

```
c("KobeBryant","JoeJohnson","LeBronJames","CarmeloAnthony","DwightHoward","ChrisB  
osh","ChrisPaul","KevinDurant","DerrickRose","DwyaneWade")
```

#Sueldo

```
KobeBryant_sueldos <-
```

```
c(15946875,17718750,19490625,21262500,23034375,24806250,25244493,27849149,30453  
805,23500000)
```

```

JoeJohnson_sueldos <-
c(12000000,12744189,13488377,14232567,14976754,16324500,18038573,19752645,21466
718,23180790)

LeBronJames_sueldos <-
c(4621800,5828090,13041250,14410581,15779912,14500000,16022500,17545000,1906750
0,20644400)

CarmeloAnthony_sueldos <-
c(3713640,4694041,13041250,14410581,15779912,17149243,18518574,19450000,2240747
4,22458000)

DwightHoward_sueldos <-
c(4493160,4806720,6061274,13758000,15202590,16647180,18091770,19536360,20513178,
21436271)

ChrisBosh_sueldos <-
c(3348000,4235220,12455000,14410581,15779912,14500000,16022500,17545000,1906750
0,20644400)

ChrisPaul_sueldos <-
c(3144240,3380160,3615960,4574189,13520500,14940153,16359805,17779458,18668431,2
0068563)

KevinDurant_sueldos <-
c(0,0,4171200,4484040,4796880,6053663,15506632,16669630,17832627,18995624)

DerrickRose_sueldos <-
c(0,0,0,4822800,5184480,5546160,6993708,16402500,17632688,18862875)

DwayneWade_sueldos <-
c(3031920,3841443,13041250,14410581,15779912,14200000,15691000,17182000,1867300
0,15000000)

#Matriz

sueldos <- rbind(KobeBryant_sueldos, JoeJohnson_sueldos, LeBronJames_sueldos,
CarmeloAnthony_sueldos, DwightHoward_sueldos, ChrisBosh_sueldos, ChrisPaul_sueldos,
KevinDurant_sueldos, DerrickRose_sueldos, DwayneWade_sueldos)

rm(KobeBryant_sueldos, JoeJohnson_sueldos, CarmeloAnthony_sueldos,
DwightHoward_sueldos, ChrisBosh_sueldos, LeBronJames_sueldos, ChrisPaul_sueldos,
DerrickRose_sueldos, DwayneWade_sueldos, KevinDurant_sueldos)

colnames(sueldos) <- temporadas
rownames(sueldos) <- jugadores

```

```
#Juegos
```

```
KobeBryant_j <- c(80,77,82,82,73,82,58,78,6,35)  
JoeJohnson_j <- c(82,57,82,79,76,72,60,72,79,80)  
LeBronJames_j <- c(79,78,75,81,76,79,62,76,77,69)  
CarmeloAnthony_j <- c(80,65,77,66,69,77,55,67,77,40)  
DwightHoward_j <- c(82,82,82,79,82,78,54,76,71,41)  
ChrisBosh_j <- c(70,69,67,77,70,77,57,74,79,44)  
ChrisPaul_j <- c(78,64,80,78,45,80,60,70,62,82)  
KevinDurant_j <- c(35,35,80,74,82,78,66,81,81,27)  
DerrickRose_j <- c(40,40,40,81,78,81,39,0,10,51)  
DwyaneWade_j <- c(75,51,51,79,77,76,49,69,54,62)
```

```
#Matriz
```

```
juegos = rbind(KobeBryant_j, JoeJohnson_j, LeBronJames_j, CarmeloAnthony_j,  
DwightHoward_j, ChrisBosh_j, ChrisPaul_j, KevinDurant_j, DerrickRose_j,  
DwyaneWade_j)  
rm(KobeBryant_j, JoeJohnson_j, CarmeloAnthony_j, DwightHoward_j, ChrisBosh_j,  
LeBronJames_j, ChrisPaul_j, DerrickRose_j, DwyaneWade_j, KevinDurant_j)  
colnames(juegos) <- temporadas  
rownames(juegos) <- jugadores
```

```
#Minutos Jugados
```

```
KobeBryant_mj <- c(3277,3140,3192,2960,2835,2779,2232,3013,177,1207)  
JoeJohnson_mj <- c(3340,2359,3343,3124,2886,2554,2127,2642,2575,2791)  
LeBronJames_mj <- c(3361,3190,3027,3054,2966,3063,2326,2877,2902,2493)  
CarmeloAnthony_mj <- c(2941,2486,2806,2277,2634,2751,1876,2482,2982,1428)  
DwightHoward_mj <- c(3021,3023,3088,2821,2843,2935,2070,2722,2396,1223)  
ChrisBosh_mj <- c(2751,2658,2425,2928,2526,2795,2007,2454,2531,1556)  
ChrisPaul_mj <- c(2808,2353,3006,3002,1712,2880,2181,2335,2171,2857)  
KevinDurant_mj <- c(1255,1255,2768,2885,3239,3038,2546,3119,3122,913)  
DerrickRose_mj <- c(1168,1168,1168,3000,2871,3026,1375,0,311,1530)  
DwyaneWade_mj <- c(2892,1931,1954,3048,2792,2823,1625,2391,1775,1971)
```

```
#Matriz
```

```
minutos_jugados <- rbind(KobeBryant_mj, JoeJohnson_mj, LeBronJames_mj,
CarmeloAnthony_mj, DwightHoward_mj, ChrisBosh_mj, ChrisPaul_mj, KevinDurant_mj,
DerrickRose_mj, DwyaneWade_mj)
rm(KobeBryant_mj, JoeJohnson_mj, CarmeloAnthony_mj, DwightHoward_mj,
ChrisBosh_mj, LeBronJames_mj, ChrisPaul_mj, DerrickRose_mj, DwyaneWade_mj,
KevinDurant_mj)
colnames(minutos_jugados) <- temporadas
rownames(minutos_jugados) <- jugadores
```

#Tiros Anotados

```
KobeBryant_ta <- c(978,813,775,800,716,740,574,738,31,266)
JoeJohnson_ta <- c(632,536,647,620,635,514,423,445,462,446)
LeBronJames_ta <- c(875,772,794,789,768,758,621,765,767,624)
CarmeloAnthony_ta <- c(756,691,728,535,688,684,441,669,743,358)
DwightHoward_ta <- c(468,526,583,560,510,619,416,470,473,251)
ChrisBosh_ta <- c(549,543,507,615,600,524,393,485,492,343)
ChrisPaul_ta <- c(407,381,630,631,314,430,425,412,406,568)
KevinDurant_ta <- c(306,306,587,661,794,711,643,731,849,238)
DerrickRose_ta <- c(208,208,208,574,672,711,302,0,58,338)
DwyaneWade_ta <- c(699,472,439,854,719,692,416,569,415,509)
```

#Matriz

```
tiros_anotados <- rbind(KobeBryant_ta, JoeJohnson_ta, LeBronJames_ta,
CarmeloAnthony_ta, DwightHoward_ta, ChrisBosh_ta, ChrisPaul_ta, KevinDurant_ta,
DerrickRose_ta, DwyaneWade_ta)
rm(KobeBryant_ta, JoeJohnson_ta, CarmeloAnthony_ta, DwightHoward_ta, ChrisBosh_ta,
LeBronJames_ta, ChrisPaul_ta, DerrickRose_ta, DwyaneWade_ta, KevinDurant_ta)
colnames(tiros_anotados) <- temporadas
rownames(tiros_anotados) <- jugadores
```

#Tiros Intentados

```
KobeBryant_ti <- c(2173,1757,1690,1712,1569,1639,1336,1595,73,713)
JoeJohnson_ti <- c(1395,1139,1497,1420,1386,1161,931,1052,1018,1025)
LeBronJames_ti <- c(1823,1621,1642,1613,1528,1485,1169,1354,1353,1279)
```

```

CarmeloAnthony_tí <- c(1572,1453,1481,1207,1502,1503,1025,1489,1643,806)
DwightHoward_tí <- c(881,873,974,979,834,1044,726,813,800,423)
ChrisBosh_tí <- c(1087,1094,1027,1263,1158,1056,807,907,953,745)
ChrisPaul_tí <- c(947,871,1291,1255,637,928,890,856,870,1170)
KevinDurant_tí <- c(647,647,1366,1390,1668,1538,1297,1433,1688,467)
DerrickRose_tí <- c(436,436,436,1208,1373,1597,695,0,164,835)
DwyaneWade_tí <- c(1413,962,937,1739,1511,1384,837,1093,761,1084)
#Matriz
tiros_intentados <- rbind(KobeBryant_tí, JoeJohnson_tí, LeBronJames_tí,
CarmeloAnthony_tí, DwightHoward_tí, ChrisBosh_tí, ChrisPaul_tí, KevinDurant_tí,
DerrickRose_tí, DwyaneWade_tí)
rm(KobeBryant_tí, JoeJohnson_tí, LeBronJames_tí, CarmeloAnthony_tí, DwightHoward_tí,
ChrisBosh_tí, ChrisPaul_tí, KevinDurant_tí, DerrickRose_tí, DwyaneWade_tí)
colnames(tiros_intentados) <- temporadas
rownames(tiros_intentados) <- jugadores

```

```

#Puntos
KobeBryant_puntos <- c(2832,2430,2323,2201,1970,2078,1616,2133,83,782)
JoeJohnson_puntos <- c(1653,1426,1779,1688,1619,1312,1129,1170,1245,1154)
LeBronJames_puntos <- c(2478,2132,2250,2304,2258,2111,1683,2036,2089,1743)
CarmeloAnthony_puntos <- c(2122,1881,1978,1504,1943,1970,1245,1920,2112,966)
DwightHoward_puntos <- c(1292,1443,1695,1624,1503,1784,1113,1296,1297,646)
ChrisBosh_puntos <- c(1572,1561,1496,1746,1678,1438,1025,1232,1281,928)
ChrisPaul_puntos <- c(1258,1104,1684,1781,841,1268,1189,1186,1185,1564)
KevinDurant_puntos <- c(903,903,1624,1871,2472,2161,1850,2280,2593,686)
DerrickRose_puntos <- c(597,597,597,1361,1619,2026,852,0,159,904)
DwyaneWade_puntos <- c(2040,1397,1254,2386,2045,1941,1082,1463,1028,1331)
#Matriz
puntos <- rbind(KobeBryant_puntos, JoeJohnson_puntos, LeBronJames_puntos,
CarmeloAnthony_puntos, DwightHoward_puntos, ChrisBosh_puntos, ChrisPaul_puntos,
KevinDurant_puntos, DerrickRose_puntos, DwyaneWade_puntos)

```

```
rm(KobeBryant_puntos, JoeJohnson_puntos, LeBronJames_puntos,  
CarmeloAnthony_puntos, DwightHoward_puntos, ChrisBosh_puntos, ChrisPaul_puntos,  
KevinDurant_puntos, DerrickRose_puntos, DwayneWade_puntos)  
colnames(puntos) <- temporadas  
rownames(puntos) <- jugadores
```

Marcos de datos

Son casi iguales a las matrices, pero su diferencia crucial es que pueden manejar más de un tipo de datos a diferencia de las matrices. Es decir, que podemos tener datos de tipo double con datos de tipo integer, o con logical, o incluso character.

```
#Los siguientes son dos métodos para importar los marcos de datos a R
```

```
?read.csv()
```

```
#Método 1 o por selección manual del archivo
```

```
datos <- read.csv(file.choose())
```

```
datos
```

```
#Método 2 o por directorio de trabajo (working directory)
```

```
getwd()
```

```
?setwd()
```

```
#NOTA, se debe reemplazar \ por \\, si no es incapaz de encontrar la dirección!!!!
```

```
setwd("C:\\\\Users\\\\Auch\\\\Downloads\\\\Datos de trabajo")
```

```
getwd()
```

```
rm(datos)
```

```
datos <- read.csv("DatosDemograficos.csv")
```

Explorando los datos

```
#Explorando los datos
```

```
datos
```

```
#Número de filas y columnas del archivo mediante nrow() y ncol()
```

```
nrow(datos)
```

```
ncol(datos)
```

```
#Para ir visualizando cómo van a estar acomodados los datos, o querer ver el nombre  
de las columnas rápidamente
```

```
#Se puede usar head() que da las primeras 6 filas, o tail() que da las últimas 6.
```

```
head(datos)
```

```
tail(datos)
```

```
#Como se dijo, por default te da 6 filas, pero se puede decidir el N de filas
```

```
head(datos, 15)
```

```
head(datos, 40)
```

```
tail(datos, 4)
```

```
tail(datos, 2)
```

```
#Para saber la estructura del marco de datos. Esto nos indica el tipo de datos, los  
códigos de las variables,
```

```
#si hay factores (variables con N niveles), el # de filas y columnas, etc.
```

```
str(datos)
```

```
#Este hace un resumen por columnas de los datos, incluse te da el min, max, 1º y 3º  
intercuartil, mediana, y media
```

```
summary(datos)
```

```
#Uso del símbolo de $  
head(datos)  
  
#Se pueden seleccionar datos como en una matriz  
datos[4,3]  
datos[4,4]  
datos[4,5]  
datos[4, "Tasa.Natalidad"]  
datos[2, "Tasa.Natalidad"]  
  
#El $ regresa un vector de alguna de las columnas de los datos  
datos$Nombre.Pais  
datos$Codigo.Pais  
datos$Tasa.Natalidad  
datos$Penetracion.Internet  
datos$Grupo.Ingresos  
  
#También nos puede ayudar a conocer los niveles de una variable cuando se junta con  
levels()  
str(datos)  
levels(datos$Grupo.Ingresos)
```

Operaciones básicas con un marco de datos

```
#Operaciones básicas con un marco de datos
```

```
#Subconjuntos. Primero se especifica las filas y luego las columnas  
datos[1:10, ]
```

```
#Si usamos c() dentro del subconjunto podemos seleccionar varias filas discontinuas  
datos[c(1,4,6),]
```

```
#A diferencia de las matrices si se selecciona una sola fila no tenemos el cambio a vector. Se  
mantiene como marco de datos
```

```
datos[1,]  
is.data.frame(datos[1,])
```

```
#Sin embargo al extraer una columna si tenemos la conversión a vector
```

```
datos[,1]  
is.data.frame(datos[,1])
```

```
#Si se quiere mantener como marco de datos se aplica drop=. Esto evita que caiga la  
dimensión de plano a línea
```

```
datos[, 1, drop= FALSE]  
is.data.frame(datos[, 1, drop= FALSE])
```

```
#Lo siguiente no tiene mucho sentido para los datos que uso, pero permite demostrar cómo  
hacer operaciones aritméticas
```

```
head(datos)  
datos$Tasa.Natalidad * datos$Penetracion.Internet  
datos$Tasa.Natalidad / datos$Penetracion.Internet  
datos$Tasa.Natalidad + datos$Penetracion.Internet  
datos$Tasa.Natalidad - datos$Penetracion.Internet  
sqrt(datos$Tasa.Natalidad)
```

```
#También se pueden agregar columnas nuevas al marco de datos  
head(datos)  
datos$columna.extra <- datos$Tasa.Natalidad * datos$Penetracion.Internet  
head(datos)
```

```
#Podemos aprovechar el reciclado de vectores para agregar columnas  
datos$xyz <- 1:3  
head(datos)
```

```
#Si queremos quitar columnas podemos hacerlo mediante NULL  
datos$xyz <- NULL  
datos$columna.extra <- NULL  
head(datos)
```

Filtrando un marco de datos

```
#Filtrando marcos de datos
```

```
head(datos)
```

```
filtro <- datos$Penetracion.Internet < 2
```

```
datos[filtro, ]
```

```
#Una forma alternativa sin generar el objeto filtro es poniéndolo directamente entre []
```

```
datos[datos$Tasa.Natalidad > 40, ]
```

```
#Si quisiéramos agregar más de una condición entonces podemos usar &, si ambas son
```

```
TRUE entonces se imprime en la consola el marco de datos deseado
```

```
datos[datos$Tasa.Natalidad > 40 & datos$Penetracion.Internet < 2, ]
```

```
#También se puede hacer el filtrado con variables categóricas
```

```
datos[datos$Grupo.Ingresos == "Ingreso alto", ]
```

```
datos[datos$Grupo.Ingresos == "Ingreso bajo", ]
```

```
# Lo mismo se puede hacer con elementos individuales de las columnas, si estas tienen un código único como en el caso de los países
```

```
datos[datos == "Malta", ]
```

```
datos[datos== "Portugal", ]
```

Introducción a qplot()

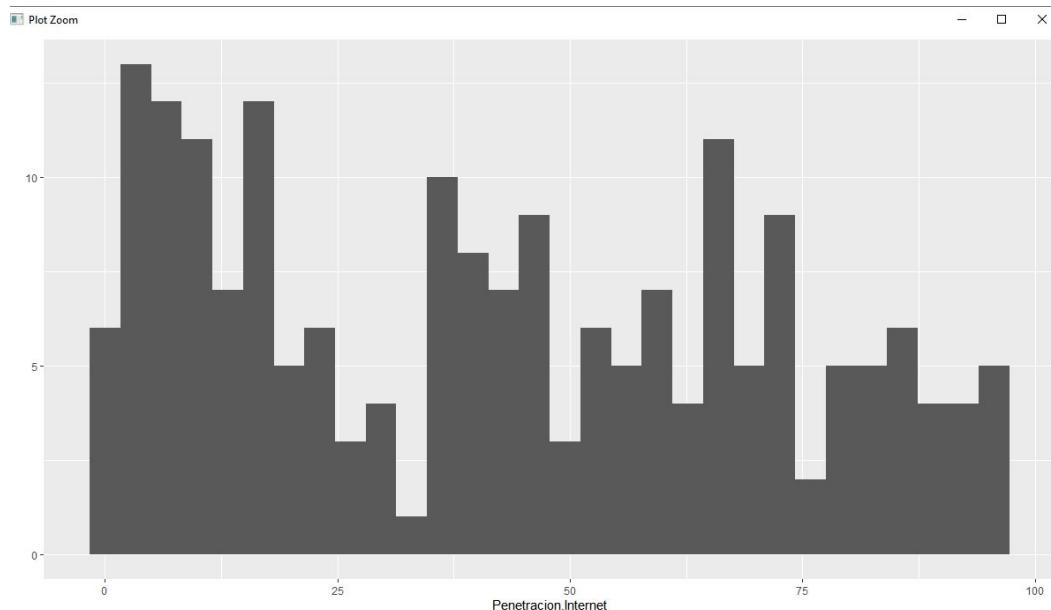
```
#INTRODUCCIÓN A qplot()
```

```
install.packages("ggplot2")
```

```
qplot(data=datos, x= Penetracion.Internet)
```

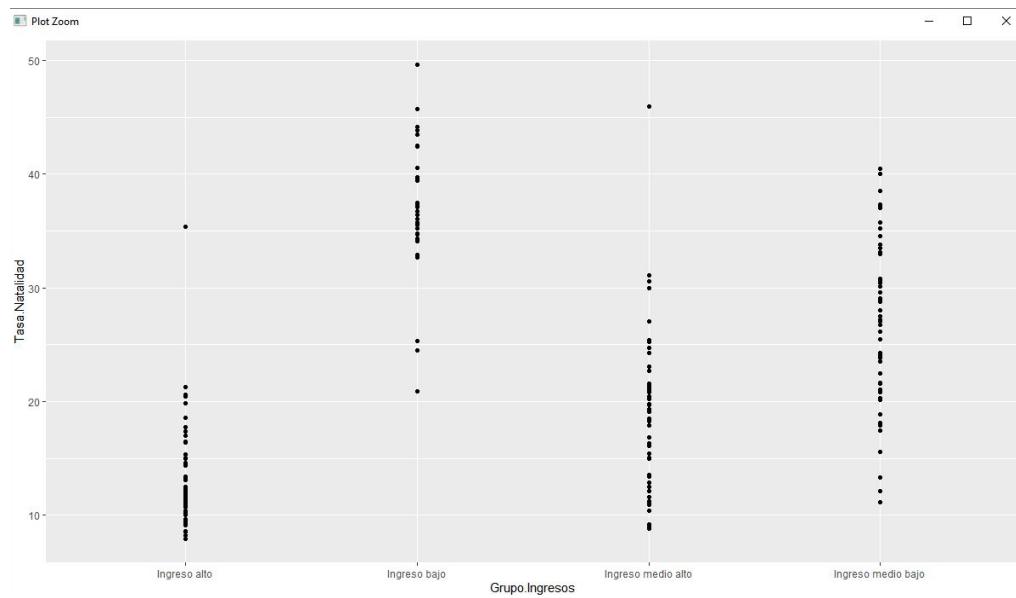
```
#Podemos graficar una variable categórica en eje x contra una numérica en y
```

```
qplot(data=datos, x=Grupo.Ingresos, y= Tasa.Natalidad)
```



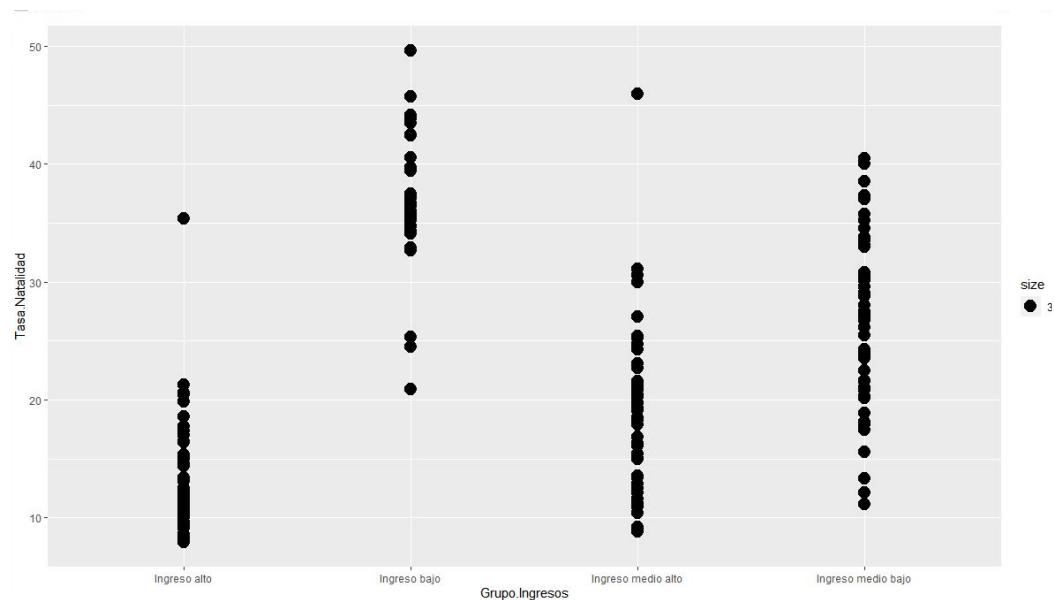
```
#Podemos aumentar el tamaño de los puntos de la gráfica para hacerlos más visibles
```

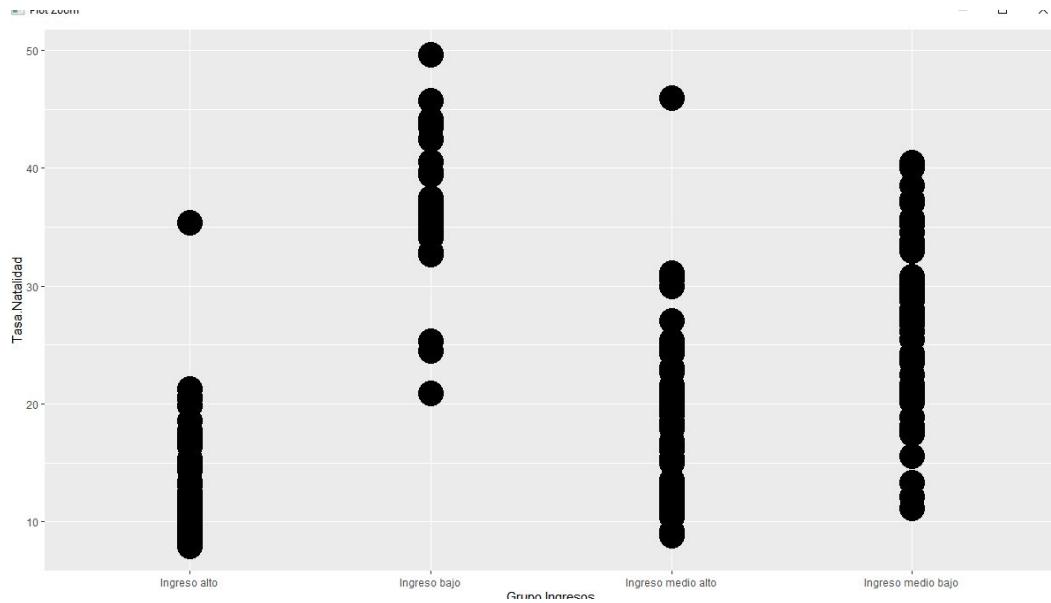
```
qplot(data=datos, x= Grupo.Ingresos, y = Tasa.Natalidad, size = 3)
```



#Así mismo se debe especificar con I() o "as is", qué tamaño queremos, o qué color queremos para el gráfico

```
qplot(data=datos, x= Grupo.Ingresos, y= Tasa.Natalidad, size= I(10))
```



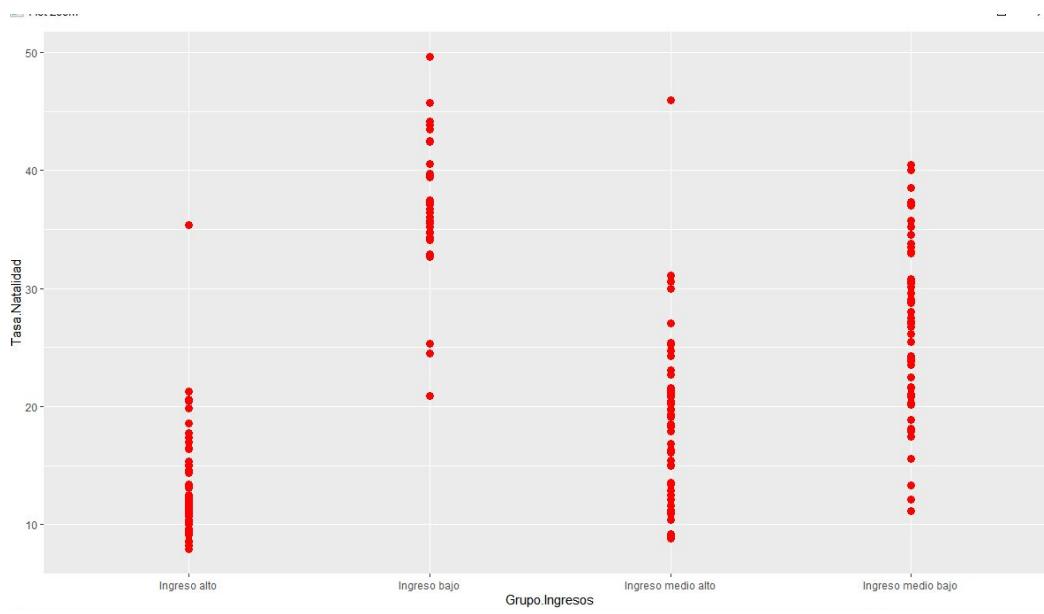
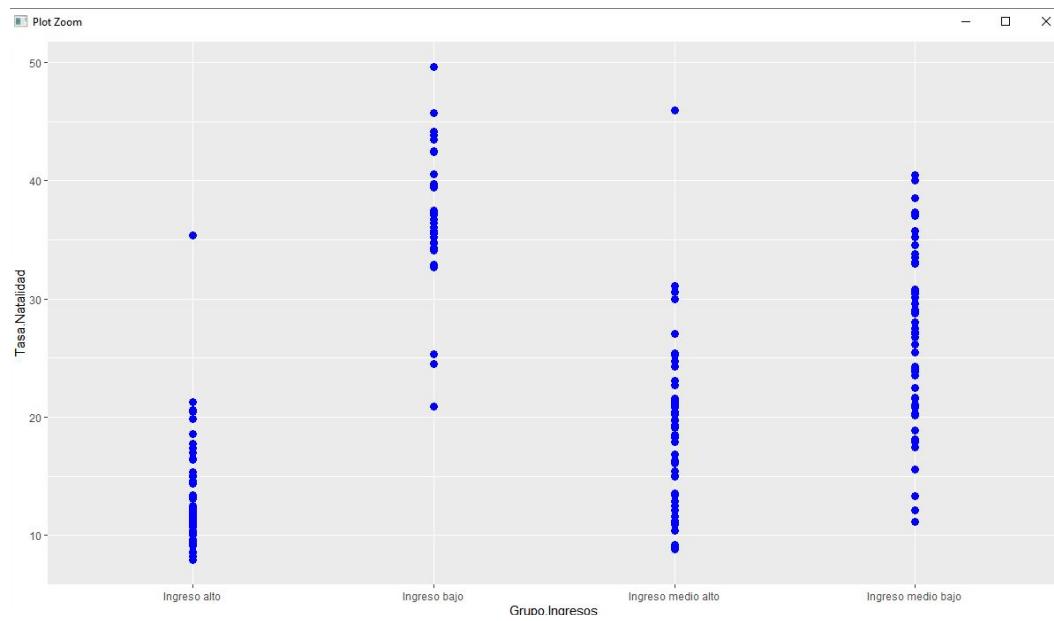


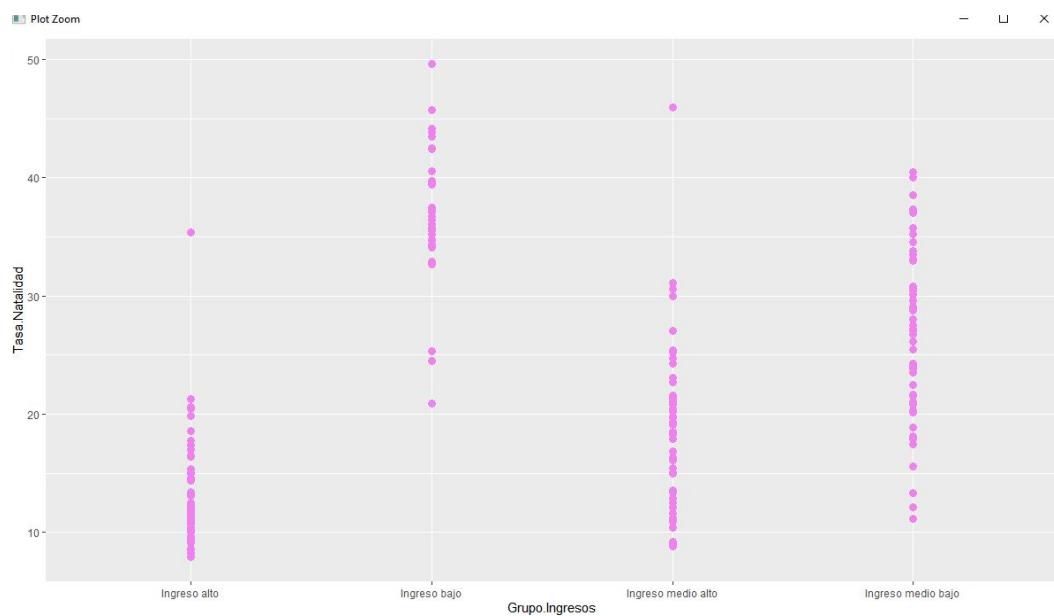
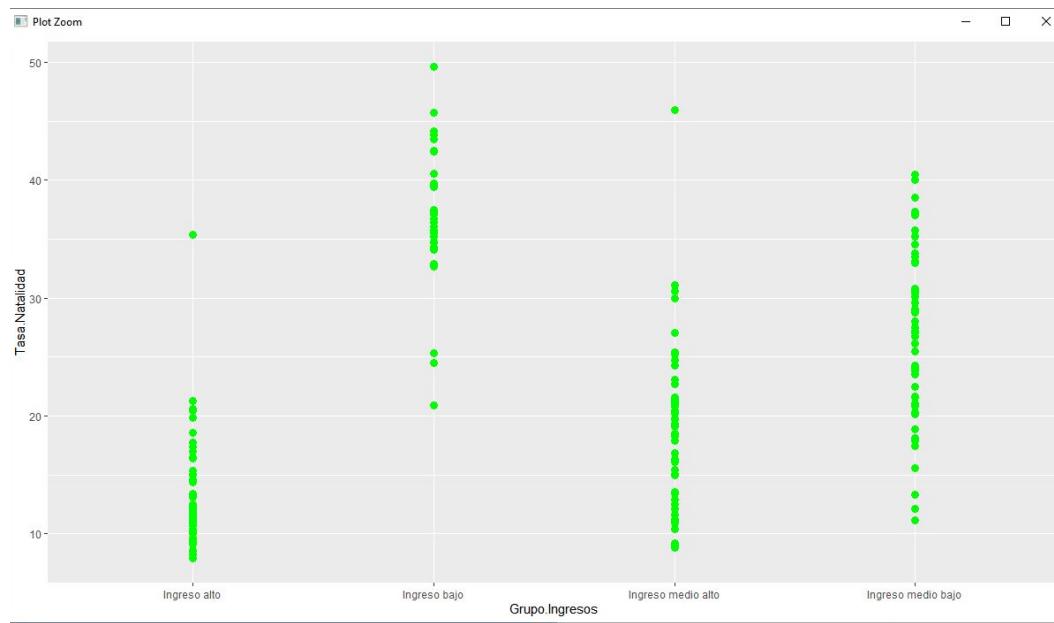
```

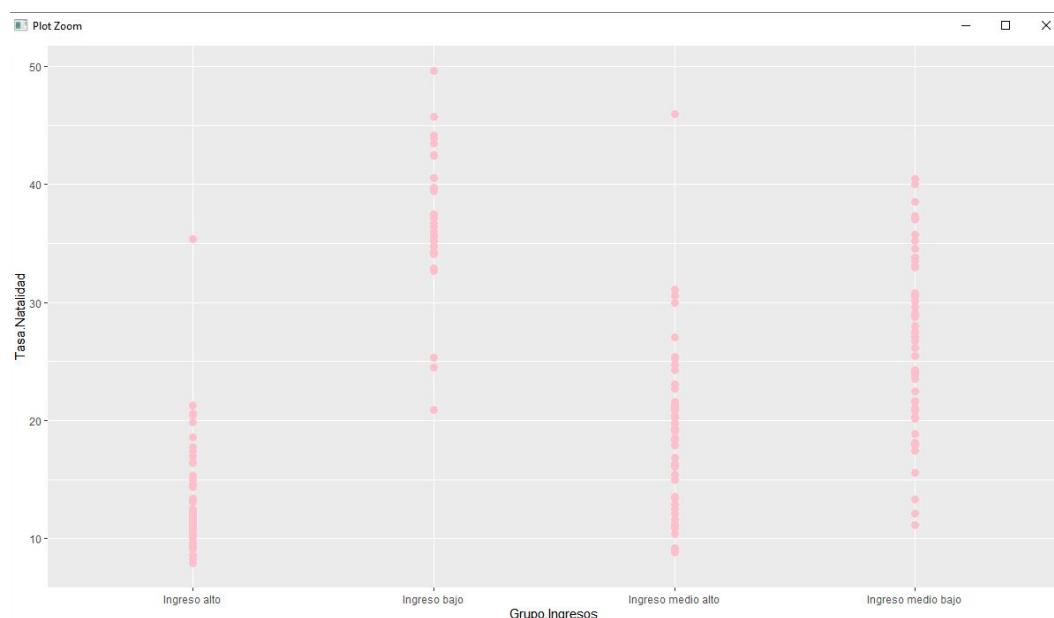
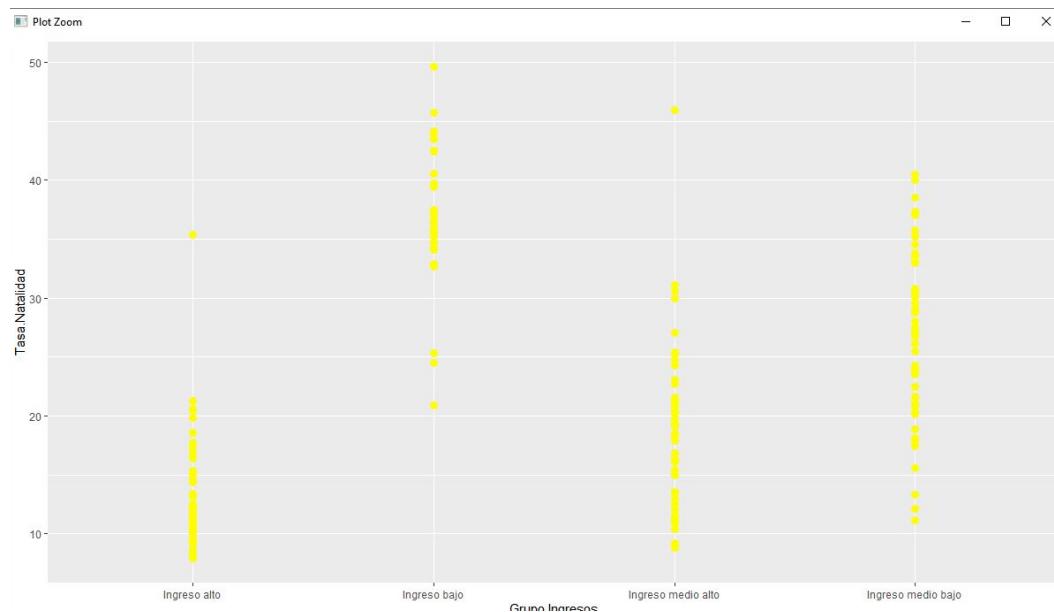
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("blue"))
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("red"))
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("green"))
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("violet"))
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("yellow"))
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, size = I(3),
      color= I("pink"))

```

#A continuación se muestran varios colores

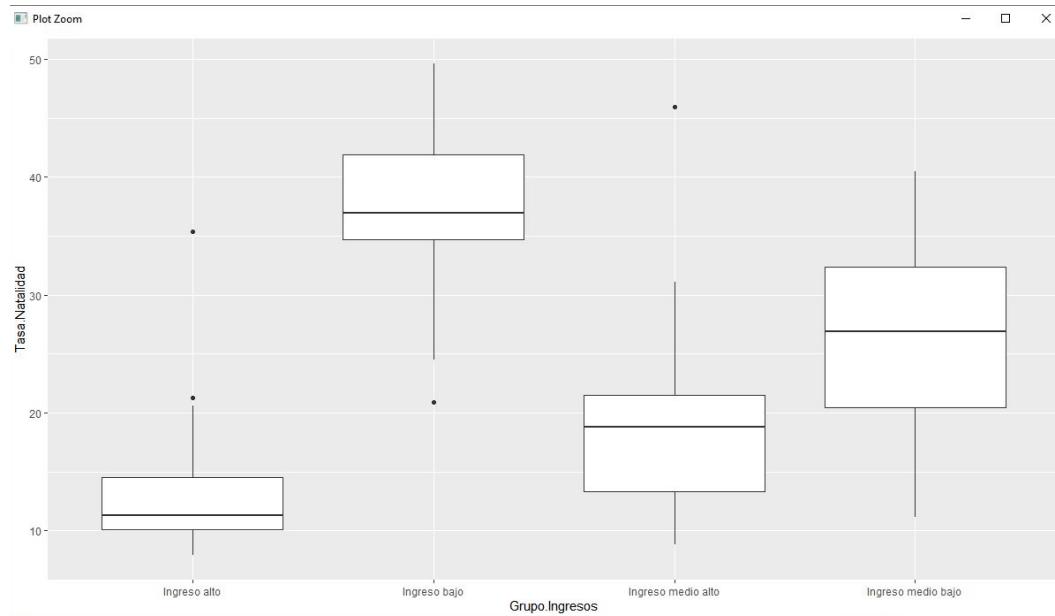






#Otra cosa que podemos alterar es la geometría de la gráfica...como un boxplot

```
qplot(data=datos, x=Grupo.Ingresos, y=Tasa.Natalidad, geom = "boxplot")
```



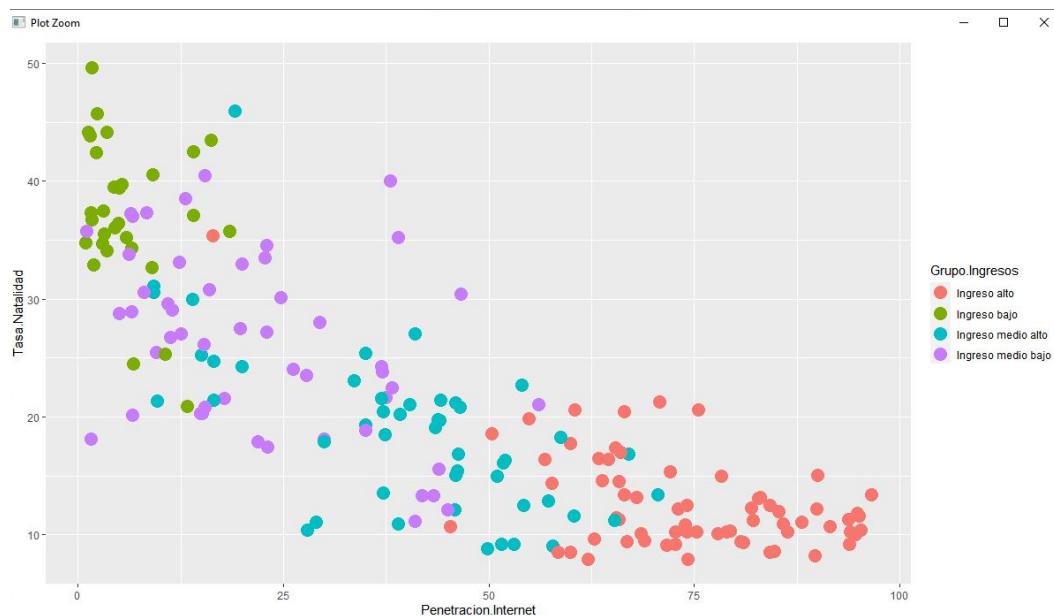
Reto de sección

Se nos pidió al inicio de esta sección de curso la generación de un gráfico donde se contrastaran los datos de penetración del internet con los de la tasa de natalidad dados los ingresos de la población. Para hacer esto se realiza lo siguiente

#Visualizando la solicitud

```
qplot(data=datos, x= Penetracion.Internet, y= Tasa.Natalidad)  
qplot(data=datos, x= Penetracion.Internet, y= Tasa.Natalidad, size= I(5))  
qplot(data=datos, x= Penetracion.Internet, y= Tasa.Natalidad, size= I(5),  
      color= Grupo.Ingresos)
```

#En eje x queda la penetración, en y la tasa de natalidad, y el color nos indica el grado de ingresos



Has recibido una actualización urgente de tu gerente.

Te está solicitando que generes un segundo diagrama de dispersión ilustrando la Tasa de Natalidad y la Penetración de Internet por País.

Sin embargo, ahora deberás de categorizar los datos por Región del País.

Los datos adicionales te los han proveído en la forma de vectores de R.

Programación en R: A-Z © SuperDataScience

#Primero se debe de crear un marco de datos con los vectores que se encuentran como anexos en el apéndice (Pais_dataset_p2, Código_Pais_dataset_p2, Region_dataset_p2). Estos no son míos y pertenecen a <http://www.superdatascience.com>

Creando marcos de datos

#Creando marcos de datos por medio de **data.frame()** que une vectores en marcos de datos, así como **cbind()** une vectores en matrices

```
mi_data_frame <- data.frame(Pais_dataset_p2, Código_Pais_dataset_p2, Region_dataset_p2)
head(mi_data_frame)
colnames(mi_data_frame) <- c("Pais", "Código", "Región")
head(mi_data_frame)

rm(mi_data_frame)
```

```
#Una forma más directa de hacer el marco de datos pero ya con los nombres que queremos
mi_data_frame <- data.frame(Pais=Pais_dataset_p2, Código=Código_Pais_dataset_p2,
                           Región=Region_dataset_p2)
head(mi_data_frame)
tail(mi_data_frame)
summary(mi_data_frame)
```

Combinando marcos de datos

```
#Combinando marcos de datos merge()
```

```
head(datos)
```

```
head(mi_data_frame)
```

```
#Nota, el by.x= y by.y= indican a merge() por que puntos unir los dataframes y es necesario que se escriban todo junto
```

```
data_frame_combinado <- merge(datos, mi_data_frame, by.x= "Codigo.Pais", by.y= "Codigo")
```

```
head(data_frame_combinado)
```

```
#Como se puede ver tenemos columnas repetidas y hay que eliminarlas por medio de $ y NULL
```

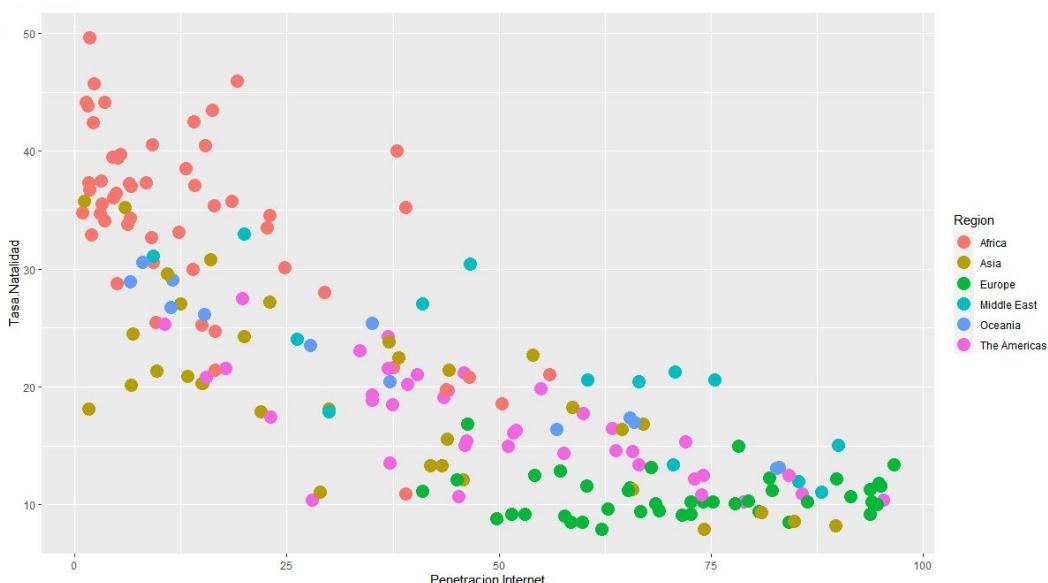
```
data_frame_combinado$Pais <- NULL
```

```
head(data_frame_combinado)
```

```
#Visualizando con qplot() parte 2. O sea con el nuevo marco de datos creado previamente
```

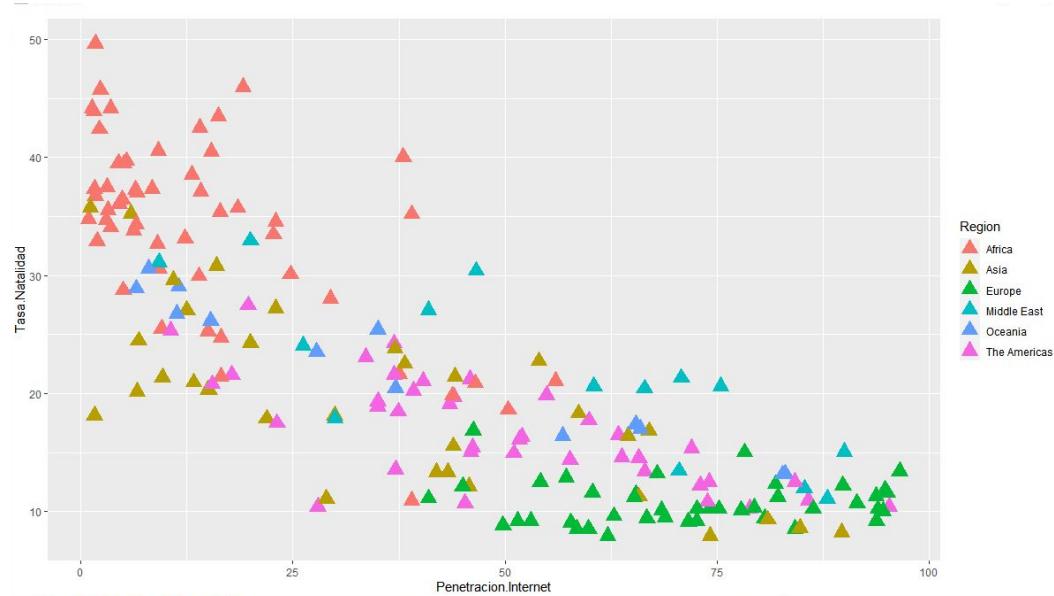
```
#Tasa de natalidad con penetración del internet en función del país
```

```
qplot(data=data_frame_combinado, x = Penetracion.Internet, y= Tasa.Natalidad, color = Region, size= I(5))
```



#Lo mismo pero con un cambio de figura

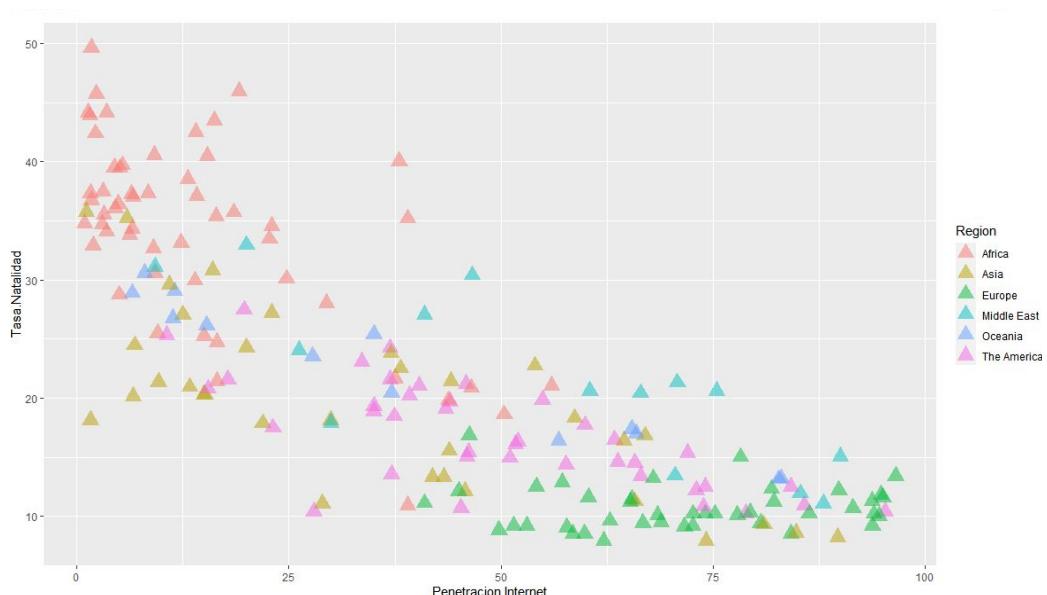
```
qplot(data=data_frame_combinado, x = Penetracion.Internet, y= Tasa.Natalidad,  
color = Region, size= I(5), shape= I(17))
```



#Se pueden transparentar las figuras y así poder mejor las conglomeraciones

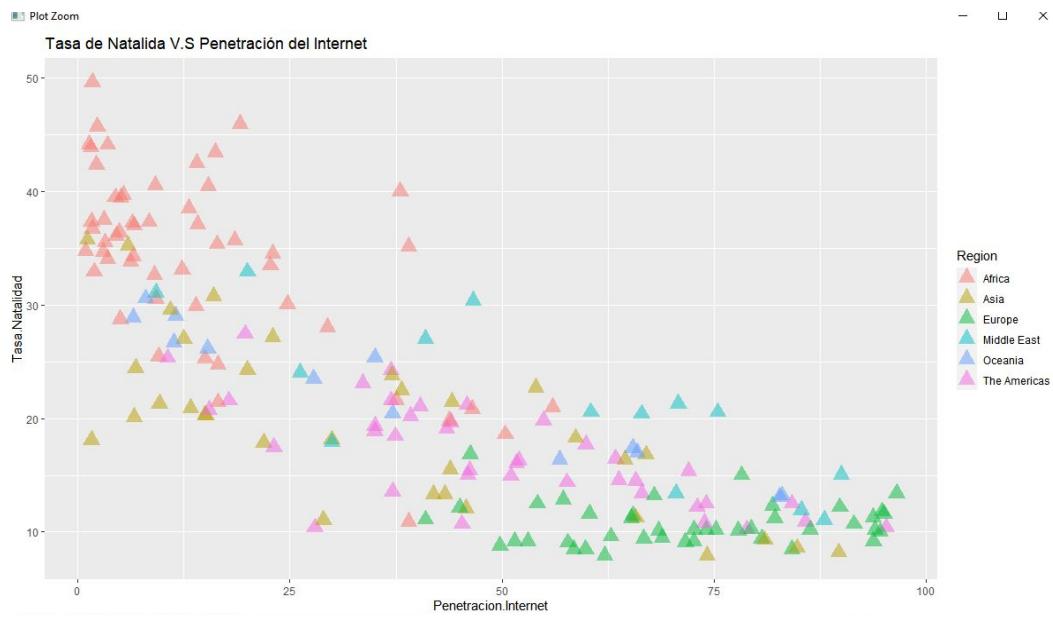
#Alpha= va de 0 a 1, donde 0 es completamente transparente y uno es opaco

```
qplot(data=data_frame_combinado, x = Penetracion.Internet, y= Tasa.Natalidad,  
color = Region, size= I(5), shape= I(17), alpha= I(.5))
```



#Aregar título en gráfico

```
qplot(data=data_frame_combinado, x = Penetracion.Internet, y= Tasa.Natalidad,  
      color = Region, size= I(5), shape= I(17), alpha= I(.5),  
      main= "Tasa de Natalida V.S Penetración del Internet")
```



Ejercicio de la sección

El Banco Mundial quedó muy impresionado con tus entregables del reto pasado y tienen un nuevo proyecto para ti.

Te han solicitado que crees un diagrama de dispersión graficando la Expectativa de Vida (eje Y) y la Tasa de Fertilidad (eje X) por País.

El diagrama de dispersión debe de estar categorizado por Región de los Países.

Te han dado datos con información de los años 1960 y 2013 y debes de crear una visualización para cada uno de estos años.

Algunos datos te los han dado en un archivo csv, otros en vectores R. El archivo csv contiene datos combinados para los dos años. Todos las manipulaciones se tienen que hacer en R (no en Excel) porque este proyecto puede ser auditado más adelante.

También te han pedido que des tus insights en cómo se comparan los dos períodos.

```
#Establece el Directorio de Trabajo
```

```
datos <- read.csv(file.choose())
```

```
datos
```

```
#Importa los datos en el archivo csv
```

```
datos <-_("Sección 5 - Práctica.csv")
```

```
#Exploramos los datos
```

```
datos
```

```
head(datos) #crevisa las 6 filas superiores
```

```
tail(datos, 7) #revisa la últimas 7 filas
str(datos)    #revisa la estructura del marco de datos
summary(datos) #revisa el resumen de los datos

#Filtra el marco de datos en 2 años diferente, uno de 2013 y otro de 1960
datos1960 <- datos[datos$Año==1960,]
datos2013 <- datos[datos$Año==2013,]

#Revisa el número de filas
summary(datos1960) #187 filas
summary(datos2013) #187 filas. Misma cantidad

#Creamos los marcos de datos adicionales
adicional1960 <- data.frame(Codigo=código_país,
Expectativa.Vida=expectativa_vida_al_nacer_1960)
adicional2013 <- data.frame(Codigo=código_país,
Expectativa.vida=expectativa_vida_al_nacer_2013)

#Revisamos los resúmenes
summary(adicional1960)
summary(adicional2013)

#Combinamos los marcos de datos de ambas fechas
combinado1960 <- merge(datos1960, adicional1960, by.x="Código.País", by.y="Codigo")
combinado2013 <- merge(datos2013, adicional2013, by.x="Código.País", by.y="Codigo")

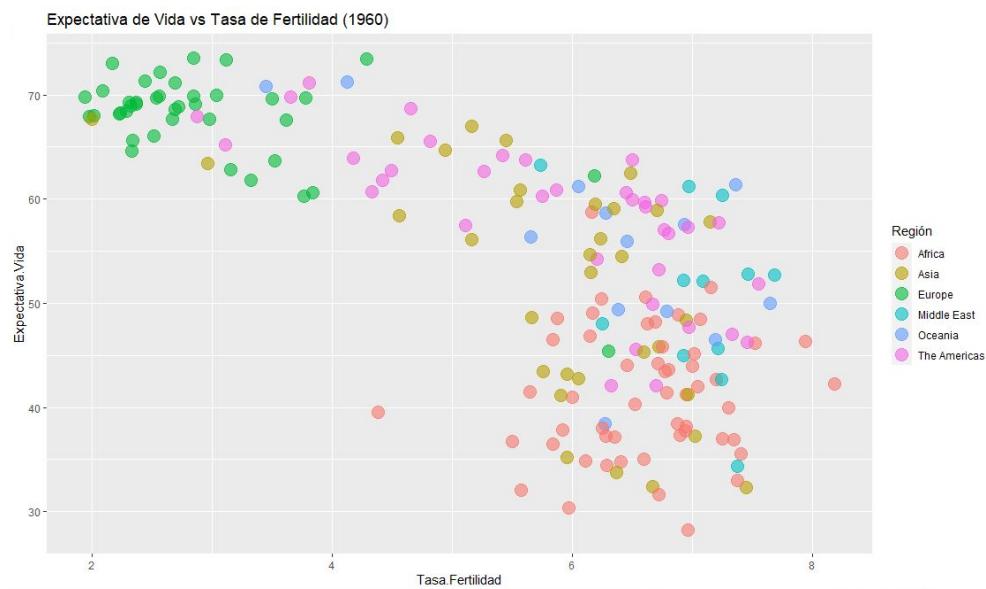
#Se revisa las nuevas estructuras
str(combinado1960)
str(combinado2013)

#Se observa que el año es obsoleto una vez que tenemos dos marcos de datos para cada año
combinado1960$Año <- NULL
combinado2013$Año <- NULL
```

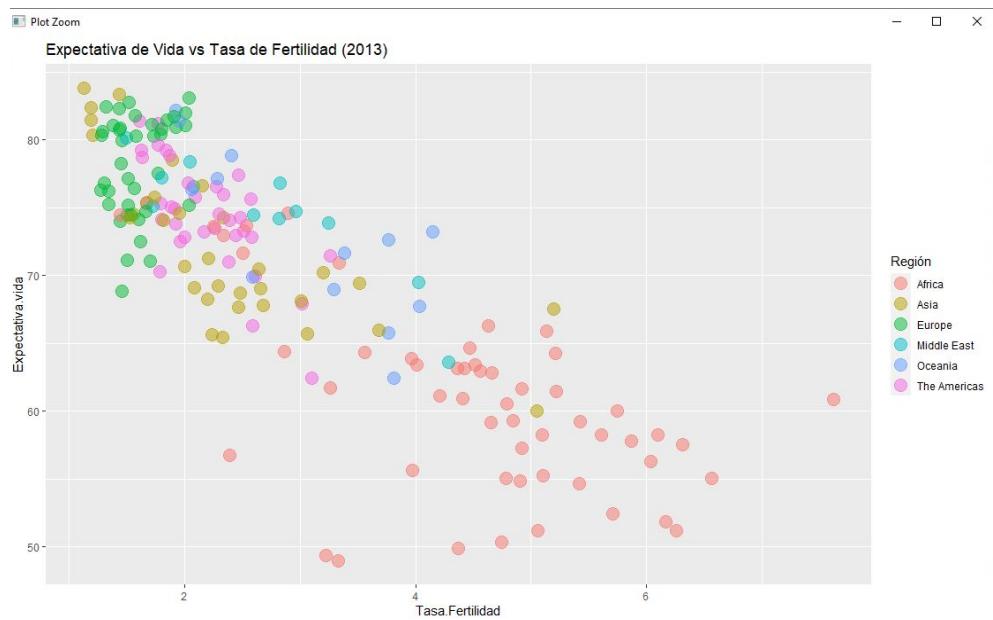
```
#Revisamos las estructuras nuevamente y ya están bien
str(combinado1960)
str(combinado2013)
```

```
#Activamos ggplot2
library(ggplot2)
```

```
#Visualizamos el marco de datos de 1960
qplot(data=combinado1960, x=Tasa.Fertilidad, y=Expectativa.Vida,
       color=Región,                                     #color
       size=I(5),                                         #size
       alpha=I(0.6),                                       #transparencia
       main="Expectativa de Vida vs Tasa de Fertilidad (1960)" #título
       )
```



```
#Visualizamos el de 2013
qplot(data=combinado2013, x=Tasa.Fertilidad, y=Expectativa.vida,
       color=Región,
       size=I(5),
       alpha=I(.5),
       main="Expectativa de Vida vs Tasa de Fertilidad (2013)")
```



#Se pudo ver que la tendencia es a aumentar la expectativa de vida y a reducir la tasa de natalidad. A excepción de África todos han seguido dicha tendencia.

Anexos

Datos siguientes no son de mi autoría y pertenecen a www.superdatascience.com

Anexo 1

```
Pais_dataset_p2 <- c("Aruba", "Afghanistan", "Angola", "Albania", "United Arab  
Emirates", "Argentina", "Armenia", "Antigua and  
Barbuda", "Australia", "Austria", "Azerbaijan", "Burundi", "Belgium", "Benin", "Burkina  
Faso", "Bangladesh", "Bulgaria", "Bahrain", "Bahamas, The", "Bosnia and  
Herzegovina", "Belarus", "Belize", "Bermuda", "Bolivia", "Brazil", "Barbados", "Brunei  
Darussalam", "Bhutan", "Botswana", "Central African  
Republic", "Canada", "Switzerland", "Chile", "China", "Cote d'Ivoire", "Cameroon", "Congo,  
Rep.", "Colombia", "Comoros", "Cabo Verde", "Costa Rica", "Cuba", "Cayman  
Islands", "Cyprus", "Czech Republic", "Germany", "Djibouti", "Denmark", "Dominican  
Republic", "Algeria", "Ecuador", "Egypt, Arab  
Rep.", "Eritrea", "Spain", "Estonia", "Ethiopia", "Finland", "Fiji", "France", "Micronesia, Fed.  
Sts.", "Gabon", "United Kingdom", "Georgia", "Ghana", "Guinea", "Gambia,  
The", "Guinea-Bissau", "Equatorial  
Guinea", "Greece", "Grenada", "Greenland", "Guatemala", "Guam", "Guyana", "Hong Kong  
SAR, China", "Honduras", "Croatia", "Haiti", "Hungary", "Indonesia", "India", "Ireland", "Iran,  
Islamic  
Rep.", "Iraq", "Iceland", "Israel", "Italy", "Jamaica", "Jordan", "Japan", "Kazakhstan", "Kenya", "K  
yrgyz Republic", "Cambodia", "Kiribati", "Korea, Rep.", "Kuwait", "Lao  
PDR", "Lebanon", "Liberia", "Libya", "St. Lucia", "Liechtenstein", "Sri  
Lanka", "Lesotho", "Lithuania", "Luxembourg", "Latvia", "Macao SAR,  
China", "Morocco", "Moldova", "Madagascar", "Maldives", "Mexico", "Macedonia,  
FYR", "Mali", "Malta", "Myanmar", "Montenegro", "Mongolia", "Mozambique", "Mauritania", "  
Mauritius", "Malawi", "Malaysia", "Namibia", "New  
Caledonia", "Niger", "Nigeria", "Nicaragua", "Netherlands", "Norway", "Nepal", "New  
Zealand", "Oman", "Pakistan", "Panama", "Peru", "Philippines", "Papua New  
Guinea", "Poland", "Puerto Rico", "Portugal", "Paraguay", "French  
Polynesia", "Qatar", "Romania", "Russian Federation", "Rwanda", "Saudi
```

Arabia", "Sudan", "Senegal", "Singapore", "Solomon Islands", "Sierra Leone", "El Salvador", "Somalia", "Serbia", "South Sudan", "Sao Tome and Principe", "Suriname", "Slovak Republic", "Slovenia", "Sweden", "Swaziland", "Seychelles", "Syrian Arab Republic", "Chad", "Togo", "Thailand", "Tajikistan", "Turkmenistan", "Timor-Leste", "Tonga", "Trinidad and

Tobago", "Tunisia", "Turkey", "Tanzania", "Uganda", "Ukraine", "Uruguay", "United States", "Uzbekistan", "St. Vincent and the Grenadines", "Venezuela, RB", "Virgin Islands (U.S.)", "Vietnam", "Vanuatu", "West Bank and Gaza", "Samoa", "Yemen, Rep.", "South Africa", "Congo, Dem. Rep.", "Zambia", "Zimbabwe")

```
Codigo_Pais_dataset_p2 <-
```

c("ABW", "AFG", "AGO", "ALB", "ARE", "ARG", "ARM", "ATG", "AUS", "AUT", "AZE", "BDI",
", "BEL", "BEN", "BFA", "BGD", "BGR", "BHR", "BHS", "BIH", "BLR", "BLZ", "BMU", "BOL",
"BRA", "BRB", "BRN", "BTN", "BWA", "CAF", "CAN", "CHE", "CHL", "CHN", "CIV", "CMR",
"COG", "COL", "COM", "CPV", "CRI", "CUB", "CYM", "CYP", "CZE", "DEU", "DJI", "DNK", "
DOM", "DZA", "ECU", "EGY", "ERI", "ESP", "EST", "ETH", "FIN", "FJI", "FRA", "FSM", "GAB
", "GBR", "GEO", "GHA", "GIN", "GMB", "GNB", "GNQ", "GRC", "GRD", "GRL", "GTM", "GU
M", "GUY", "HKG", "HND", "HRV", "HTI", "HUN", "IDN", "IND", "IRL", "IRN", "IRQ", "ISL", "
ISR", "ITA", "JAM", "JOR", "JPN", "KAZ", "KEN", "KGZ", "KHM", "KIR", "KOR", "KWT", "LA
O", "LBN", "LBR", "LBY", "LCA", "LIE", "LKA", "LSO", "LTU", "LUX", "LVA", "MAC", "MAR
", "MDA", "MDG", "MDV", "MEX", "MKD", "MLI", "MLT", "MMR", "MNE", "MNG", "MOZ", "
MRT", "MUS", "MWI", "MYS", "NAM", "NCL", "NER", "NGA", "NIC", "NLD", "NOR", "NPL",
"NZL", "OMN", "PAK", "PAN", "PER", "PHL", "PNG", "POL", "PRI", "PRT", "PRY", "PYF", "Q
AT", "ROU", "RUS", "RWA", "SAU", "SDN", "SEN", "SGP", "SLB", "SLE", "SLV", "SOM", "SR
B", "SSD", "STP", "SUR", "SVK", "SVN", "SWE", "SWZ", "SYC", "SYR", "TCD", "TGO", "THA"
, "TJK", "TKM", "TLS", "TON", "TTO", "TUN", "TUR", "TZA", "UGA", "UKR", "URY", "USA", "
UZB", "VCT", "VEN", "VIR", "VNM", "VUT", "PSE", "WSM", "YEM", "ZAF", "COD", "ZMB", "
ZWE")

```
Region_dataset_p2 <- c("The Americas","Asia","Africa","Europe","Middle East","The Americas","Asia","The
```

Americas", "Oceania", "Europe", "Asia", "Africa", "Europe", "Africa", "Africa", "Asia", "Europe", "Middle East", "The Americas", "Europe", "Europe", "The Americas", "The Americas", "The Americas", "The Americas", "The Americas", "Asia", "Asia", "Africa", "Africa", "Africa", "The Americas", "Europe", "The Americas", "Asia", "Africa", "Africa", "Africa", "The Americas", "Africa", "Africa", "The Americas", "The Americas", "The

Datos siguientes no son de mi autoría y pertenecen a www.superdatascience.com

Anexo 2

```
expectativa_vida_al_nacer_1960 <-
c(65.5693658536586,32.328512195122,32.9848292682927,62.2543658536585,52.24321951
21951,65.2155365853659,65.8634634146342,61.7827317073171,70.8170731707317,68.585
6097560976,60.836243902439,41.2360487804878,69.7019512195122,37.2782682926829,3
4.4779024390244,45.8293170731707,69.2475609756098,52.0893658536585,62.729048780
4878,60.2762195121951,67.7080975609756,59.9613658536585,42.1183170731707,54.2054
634146342,60.7380487804878,62.5003658536585,32.3593658536585,50.5477317073171,3
6.4826341463415,71.1331707317073,71.3134146341463,57.4582926829268,43.465804878
0488,36.8724146341463,41.523756097561,48.5816341463415,56.716756097561,41.442439
0243903,48.8564146341463,60.5761951219512,63.9046585365854,69.5939268292683,70.3
487804878049,69.3129512195122,44.0212682926829,72.1765853658537,51.845268292682
9,46.1351219512195,53.215,48.0137073170732,37.3629024390244,69.1092682926829,67.9
059756097561,38.4057073170732,68.819756097561,55.9584878048781,69.8682926829268
,57.5865853658537,39.5701219512195,71.1268292682927,63.4318536585366,45.83146341
46342,34.8863902439024,32.0422195121951,37.8404390243902,36.7330487804878,68.163
9024390244,59.8159268292683,45.5316341463415,61.2263414634146,60.2787317073171,
66.9997073170732,46.2883170731707,64.6086585365854,42.1000975609756,68.00317073
17073,48.6403170731707,41.1719512195122,69.691756097561,44.945512195122,48.03068
29268293,73.4286585365854,69.1239024390244,64.1918292682927,52.6852682926829,67.
6660975609756,58.3675853658537,46.3624146341463,56.1280731707317,41.23202439024
39,49.2159756097561,53.0013170731707,60.3479512195122,43.2044634146342,63.280121
9512195,34.7831707317073,42.6411951219512,57.303756097561,59.7471463414634,46.51
07073170732,69.8473170731707,68.4463902439024,69.7868292682927,64.6609268292683
,48.4466341463415,61.8127804878049,39.9746829268293,37.2686341463415,57.06563414
63415,60.6228048780488,28.2116097560976,67.6017804878049,42.7363902439024,63.705
6097560976,48.3688048780488,35.0037073170732,43.4830975609756,58.7452195121951,
37.7736341463415,59.4753414634146,46.8803902439024,58.6390243902439,35.51504878
04878,37.1829512195122,46.9988292682927,73.3926829268293,73.549756097561,35.1708
292682927,71.2365853658537,42.6670731707317,45.2904634146342,60.8817073170732,4
7.6915853658537,57.8119268292683,38.462243902439,67.6804878048781,68.7196097560
```

976,62.8089268292683,63.7937073170732,56.3570487804878,61.2060731707317,65.64243
90243903,66.0552926829268,42.2492926829268,45.6662682926829,48.1876341463415,38.
206,65.6598292682927,49.3817073170732,30.3315365853659,49.9479268292683,36.96587
80487805,31.6767073170732,50.4513658536585,59.6801219512195,69.9759268292683,68.
9780487804878,73.0056097560976,44.2337804878049,52.768243902439,38.016121951219
5,40.2728292682927,54.6993170731707,56.1535365853659,54.4586829268293,33.7271219
512195,61.3645365853659,62.6575853658537,42.009756097561,45.3844146341463,43.653
8780487805,43.9835609756098,68.2995365853659,67.8963902439025,69.7707317073171,
58.8855365853659,57.7238780487805,59.2851219512195,63.7302195121951,59.06702439
02439,46.4874878048781,49.969512195122,34.3638048780488,49.0362926829268,41.0180
487804878,45.1098048780488,51.5424634146342)
expectativa_vida_al_nacer_2013 <-
c(75.3286585365854,60.0282682926829,51.8661707317073,77.537243902439,77.19563414
63415,75.9860975609756,74.5613658536585,75.7786585365854,82.1975609756098,80.890
243902439,70.6931463414634,56.2516097560976,80.3853658536585,59.3120243902439,5
8.2406341463415,71.245243902439,74.4658536585366,76.5459512195122,75.0735365853
659,76.2769268292683,72.4707317073171,69.9820487804878,67.9134390243903,74.12243
90243903,75.3339512195122,78.5466585365854,69.1029268292683,64.3608048780488,49.
8798780487805,81.4011219512195,82.7487804878049,81.1979268292683,75.35302439024
39,51.2084634146342,55.0418048780488,61.6663902439024,73.8097317073171,62.932170
7317073,72.9723658536585,79.2252195121951,79.2563902439025,79.9497804878049,78.2
780487804878,81.0439024390244,61.6864634146342,80.3024390243903,73.319902439024
4,74.5689512195122,75.648512195122,70.9257804878049,63.1778780487805,82.42682926
82927,76.4243902439025,63.4421951219512,80.8317073170732,69.9179268292683,81.968
2926829268,68.9733902439024,63.8435853658537,80.9560975609756,74.079512195122,6
1.1420731707317,58.216487804878,59.9992682926829,54.8384146341464,57.2908292682
927,80.6341463414634,73.1935609756098,71.4863902439024,78.872512195122,66.310024
3902439,83.8317073170732,72.9428536585366,77.1268292682927,62.4011463414634,75.2
682926829268,68.7046097560976,67.6604146341463,81.0439024390244,75.125975609756
1,69.4716829268293,83.1170731707317,82.290243902439,73.4689268292683,73.90141463
41463,83.3319512195122,70.45,60.9537804878049,70.2024390243902,67.7720487804878,
65.7665853658537,81.459756097561,74.462756097561,65.687243902439,80.12887804878
05,60.5203902439024,71.6576829268293,74.9127073170732,74.2402926829268,49.331463
4146342,74.1634146341464,81.7975609756098,73.9804878048781,80.3391463414634,73.7

090487804878,68.811512195122,64.6739024390244,76.6026097560976,76.5326585365854
,75.1870487804878,57.5351951219512,80.7463414634146,65.6540975609756,74.75836585
36585,69.0618048780488,54.641512195122,62.8027073170732,74.46,61.466,74.567512195
122,64.3438780487805,77.1219512195122,60.8281463414634,52.4421463414634,74.51475
6097561,81.1048780487805,81.4512195121951,69.222,81.4073170731707,76.84104878048
78,65.9636829268293,77.4192195121951,74.2838536585366,68.1315609756097,62.449170
7317073,76.8487804878049,78.7111951219512,80.3731707317073,72.7991707317073,76.3
340731707317,78.4184878048781,74.4634146341463,71.0731707317073,63.394829268292
7,74.1776341463415,63.1670487804878,65.878756097561,82.3463414634146,67.71892682
92683,50.3631219512195,72.4981463414634,55.0230243902439,55.2209024390244,66.259
512195122,70.99,76.2609756097561,80.2780487804878,81.7048780487805,48.9379268292
683,74.7157804878049,51.1914878048781,59.1323658536585,74.2469268292683,69.40017
07317073,65.4565609756098,67.5223658536585,72.6403414634147,70.3052926829268,73.
6463414634147,75.1759512195122,64.2918292682927,57.7676829268293,71.15951219512
2,76.8361951219512,78.8414634146341,68.2275853658537,72.8108780487805,74.0744146
341464,79.6243902439024,75.756487804878,71.669243902439,73.2503902439024,63.5835
12195122,56.7365853658537,58.2719268292683,59.2373658536585,55.633)

Visualizaciones en ggplot2

Reto de la sección

Te han contactado como analista de datos por una página web de críticas de películas. Están escribiendo un artículo donde analizan los ratings de los críticos y de la audiencia, así como los presupuestos para las películas de los años 2007-2011.

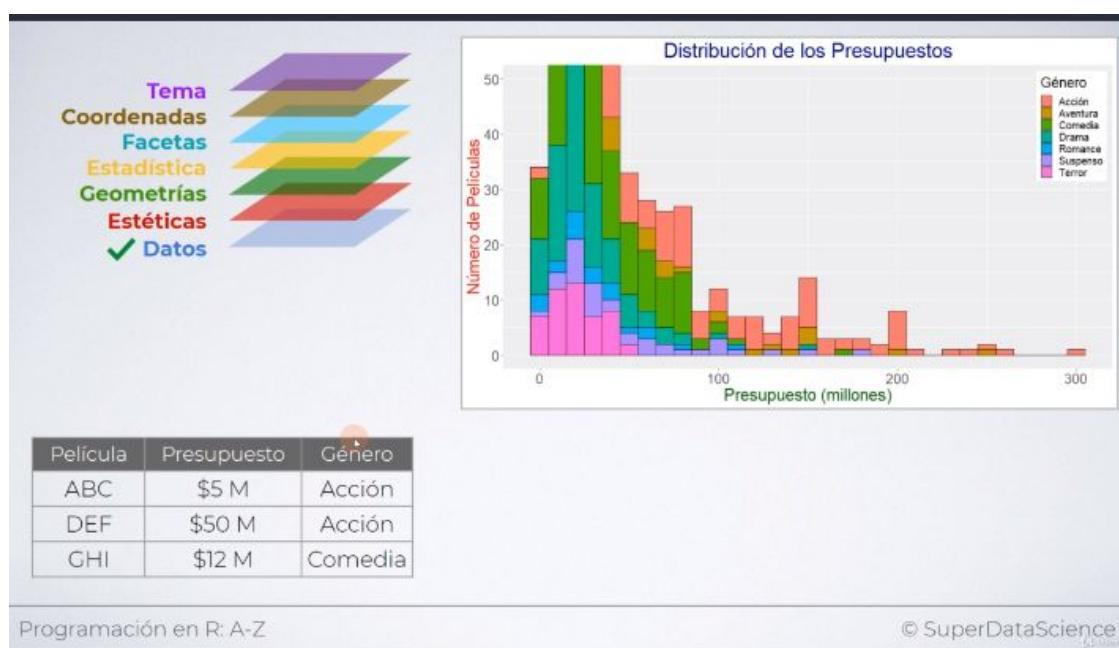
Esta es la primera vez que están haciendo este análisis y no saben exactamente qué necesitan. Te han pedido que veas los datos y que les generes 5 gráficas que cuenten una historia de los datos.

Sin embargo, hay un gráfico que el CEO pidió específicamente - un diagrama que muestre cómo ha ido evolucionando a través de los años la correlación entre los ratings de la audiencia y de los críticos, por género. Esto es adicional a los otros 5 gráficos.

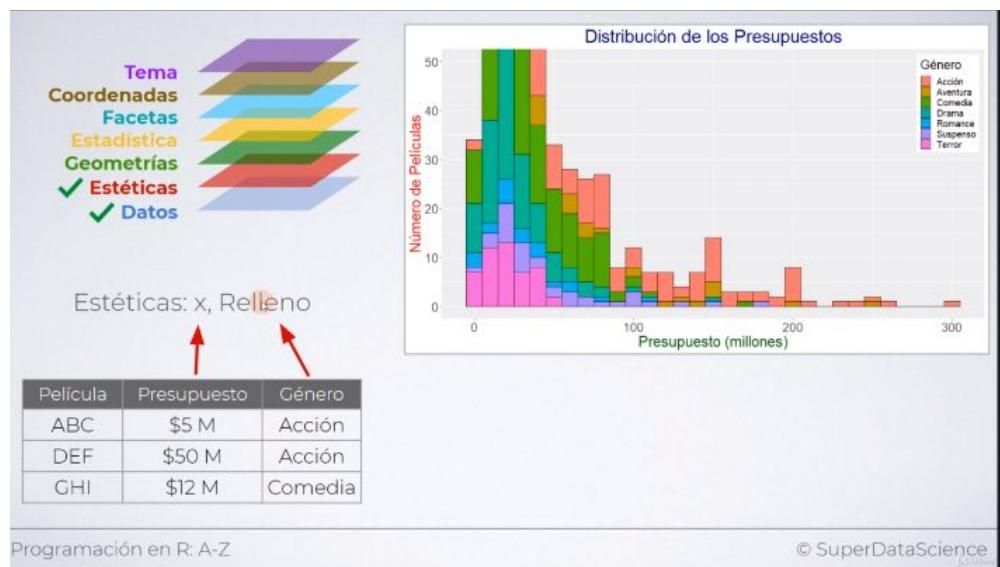
Gramática de Datos



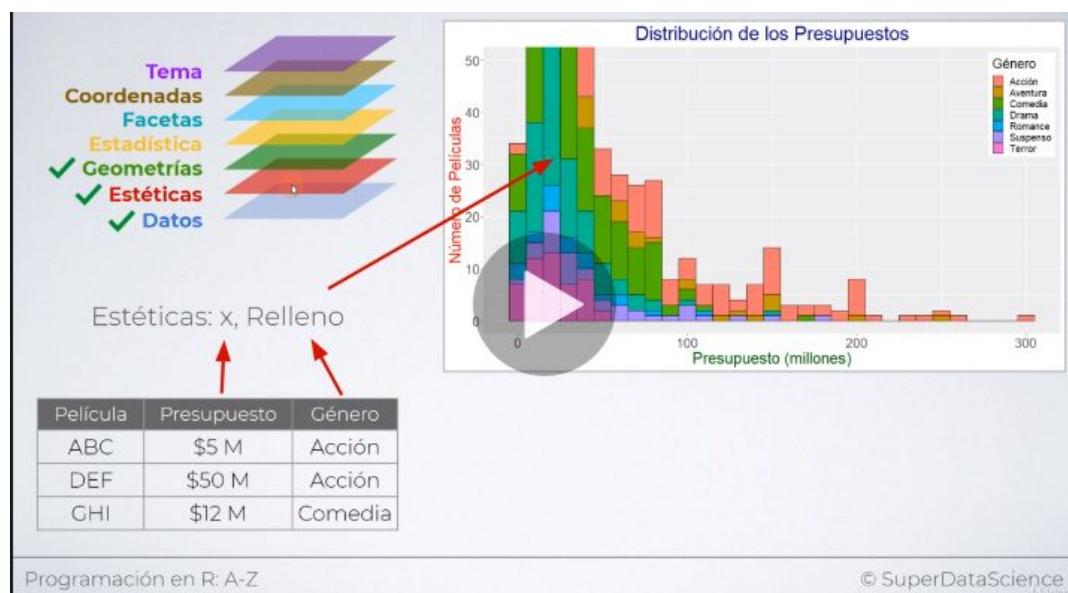
• Datos



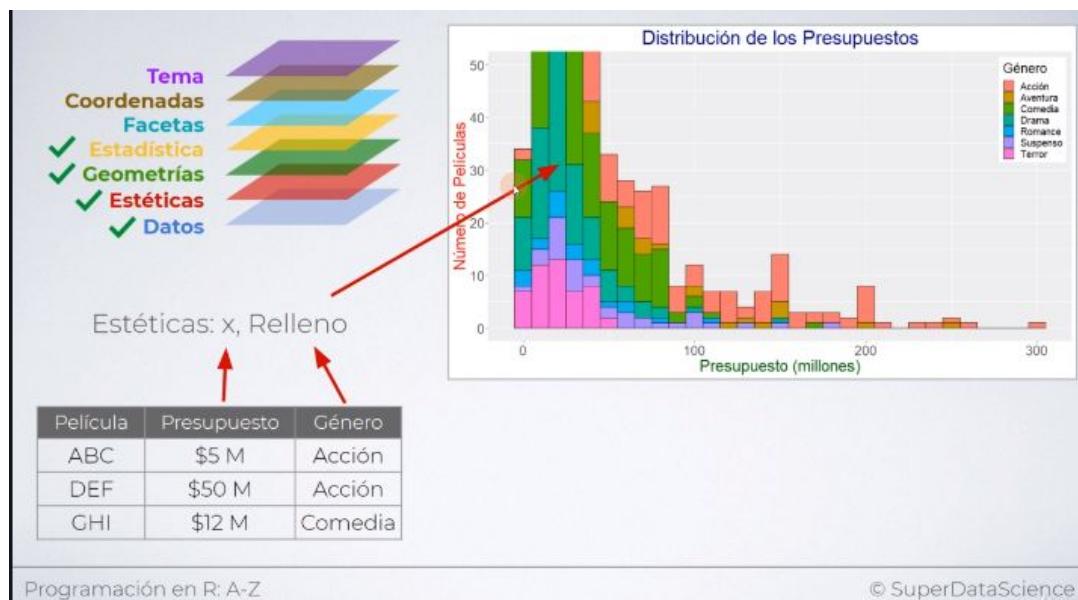
- **Estéticas** es un elemento representado en el gráfico, no es el número propiamente sino la relación de este dato a algo como el color, el tamaño, etc.



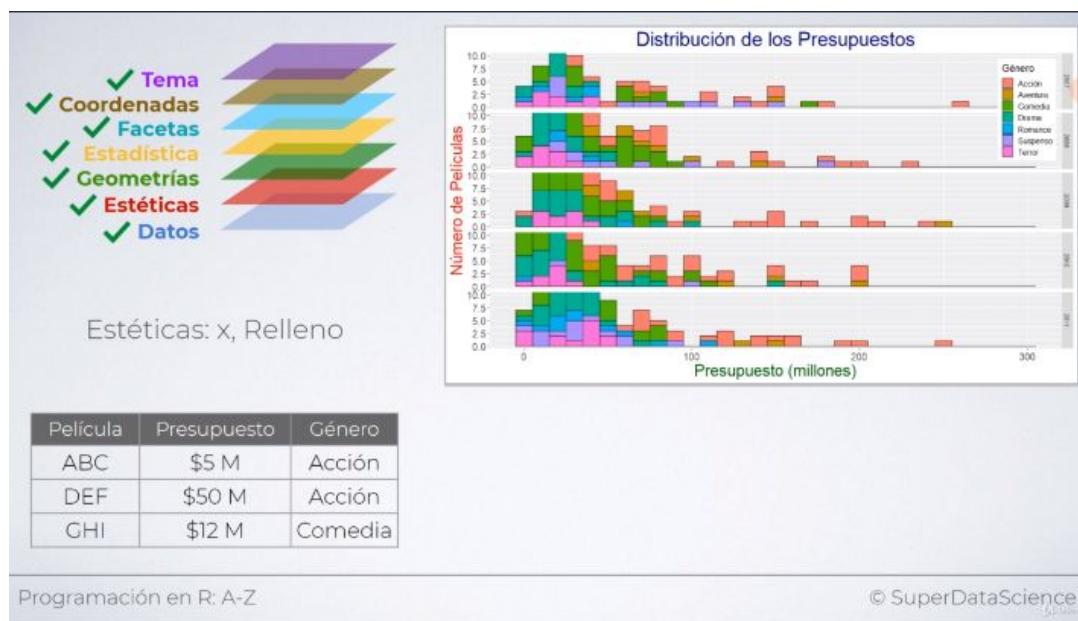
- **Geometría** define la forma de la capa anterior, algo así como el boxplot, diagramas de tallos, gráficos de dispersión, barras, pie, etc.



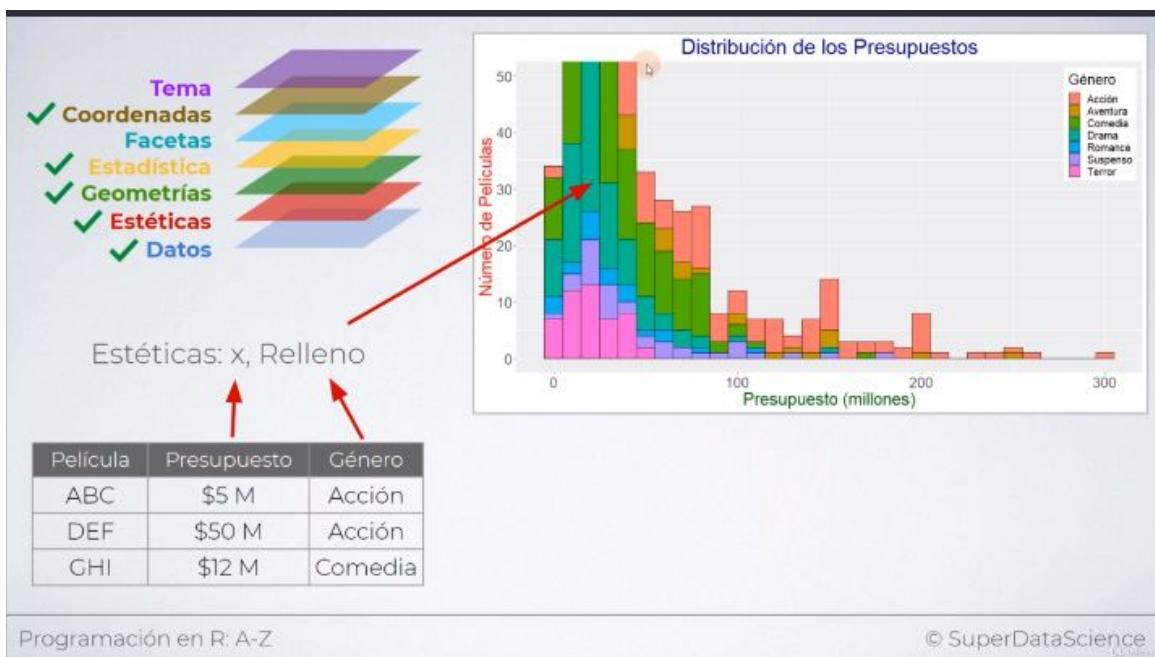
- **Estadística**, es la capa de manipulación de los datos por medio de modelos estadísticos que permite nuevas formas de comprender los datos. En el ejemplo de abajo vemos esto en el mismo histograma que genera una distribución de los datos, la cual no es normal sin sesgada.



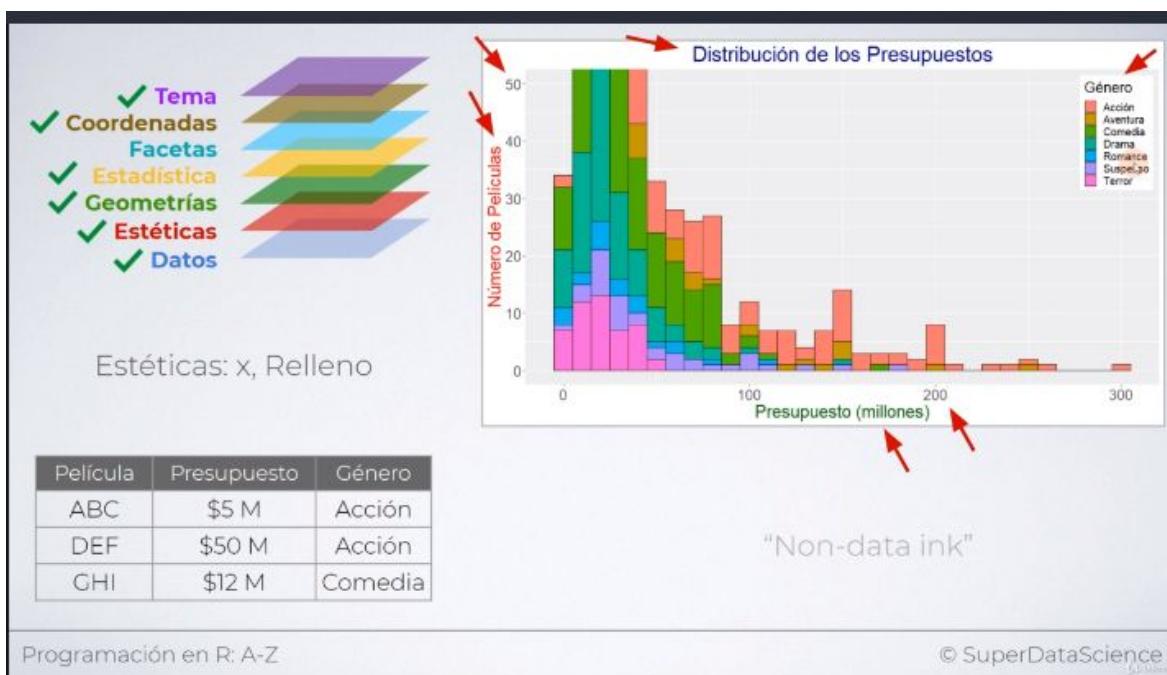
- **Facetas**, son cuando un gráfico X se divide de N maneras dada N categorías distintas; como los años el el ejemplo de abajo.



- **Coordenadas**, ya sean (x,y), polares, (x,y,z), etc. En el ejemplo de abajo se ve en los ejes (x, y) y en el zoom que tienen los datos



- **Tema**, el resto del gráfico que no está asociado con los datos en manera directa. En el caso de abajo tenemos nombres, letras, título, etc.



Factores (capa datos)

```
#Primero seleccionamos nuestros datos  
datos.peliculas <- read.csv(file.choose())  
head(datos.peliculas)
```

```
#Revisamos los datos  
colnames(datos.peliculas) <- c("Pelicula", "Genero", "RatingCriticos", "RatingAudienicia",  
"PresupuestoMillones", "Año")  
head(datos.peliculas)  
tail(datos.peliculas)  
str(datos.peliculas)
```

#Esto no es del curso, pero fue necesario abrir el paquete **dplyr**, un editor de gramática de datos, para así poder poner como factores los datos character del documento proporcionado.
#Lo que hace es usar la función **mutate_at()** para poder cambiar el tipo de dato de character a factor.

```
library(dplyr)  
datos.peliculas <- mutate_at(datos.peliculas, vars(Genero), as.factor)  
datos.peliculas <- mutate_at(datos.peliculas, vars(Pelicula), as.factor)  
str(datos.peliculas)  
summary(datos.peliculas)
```

#Ahora al parecer nos están enseñando cómo alterar esto pero sin necesidad del tidyverse.

#Esto mediante **factor()**

```
factor(datos.peliculas$Año)  
datos.peliculas$Año <- factor(datos.peliculas$Año)  
summary(datos.peliculas)  
str(datos.peliculas)
```

#Por esto anterior, podemos ver que pudimos haber hecho lo siguiente de en lugar de alterar los datos mediante dplyr

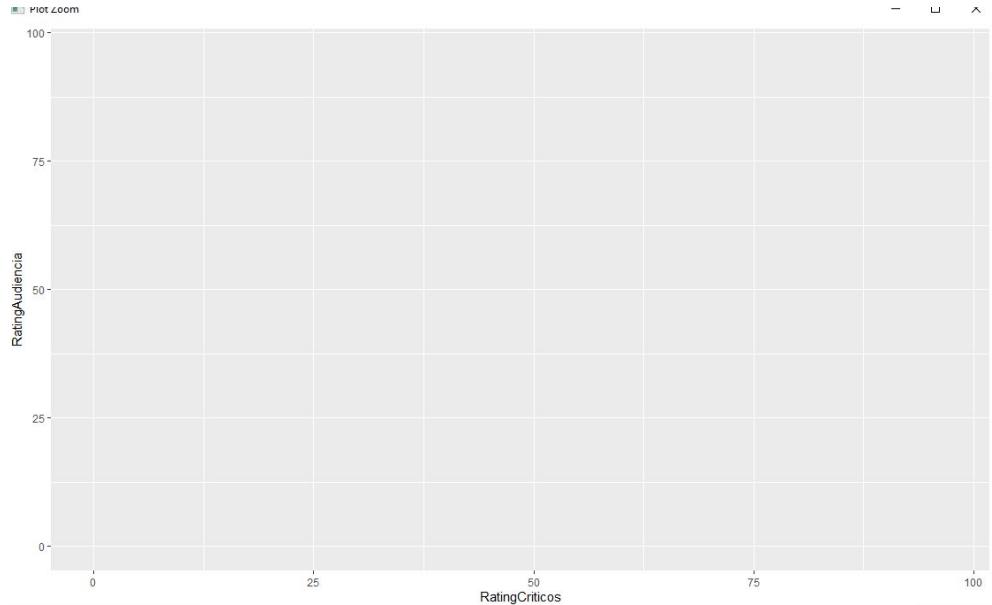
```
datos.peliculas$Año <- factor(datos.peliculas$Año)
```

Estéticas y Geometrías

#Estéticas

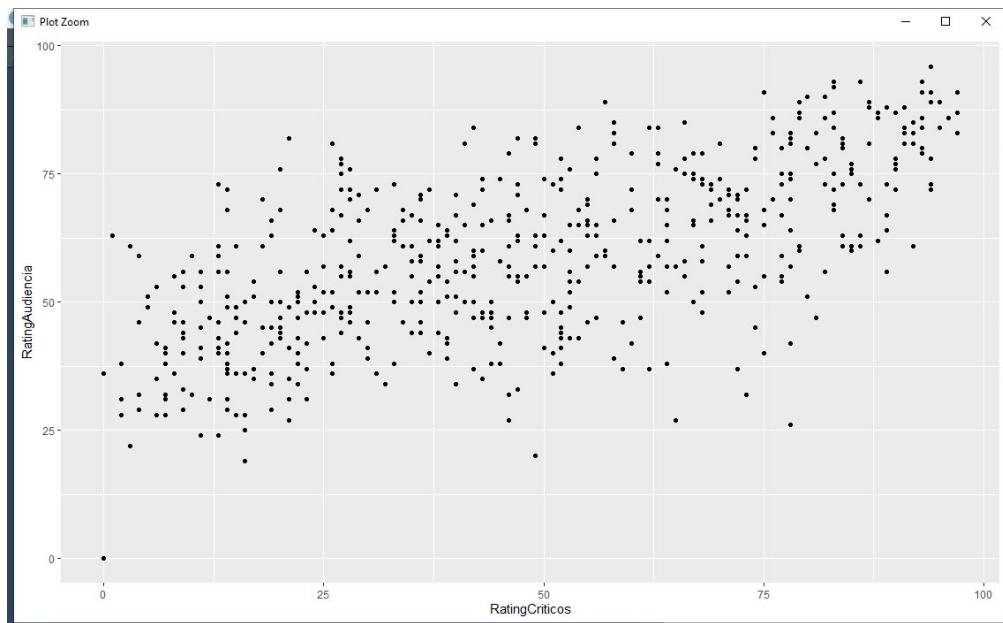
```
library(ggplot2)  
ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudien  
cia))
```

#Al aplicar esto anterior solamente se obtiene el eje (x,y), pero no obtenemos nada más
#Esto se debe a que las estéticas solo van a generar la relación entre los datos y el gráfico
pero no el gráfico en sí. Para esto se necesitan geometrías



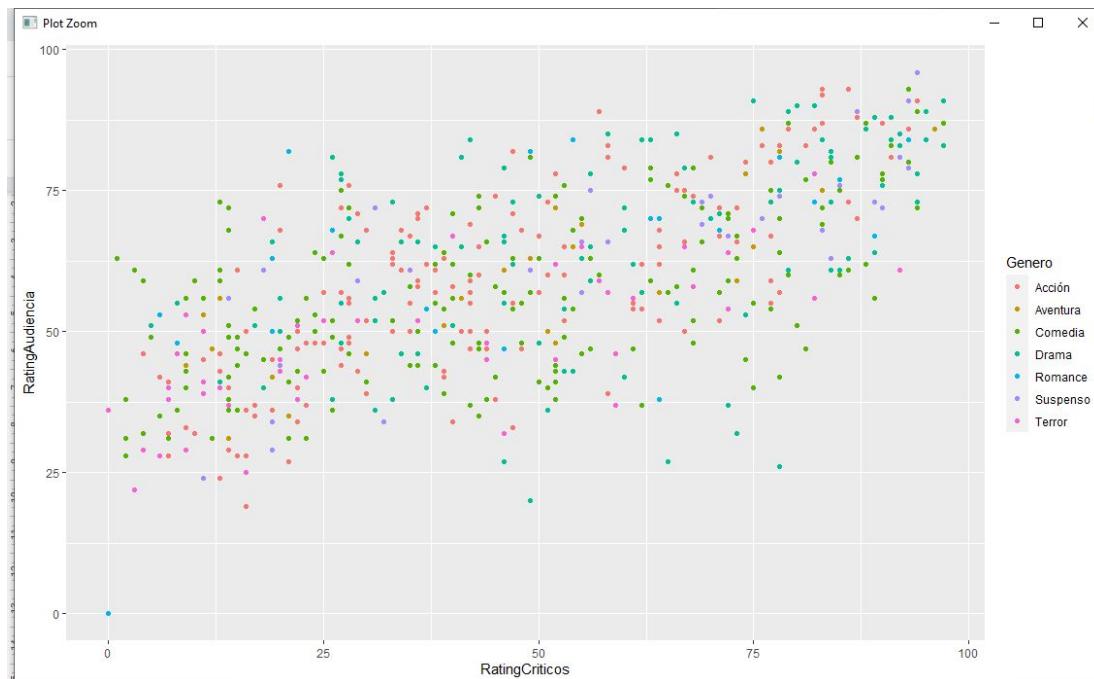
#Geometrías

```
ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudien  
cia))+  
  geom_point()
```



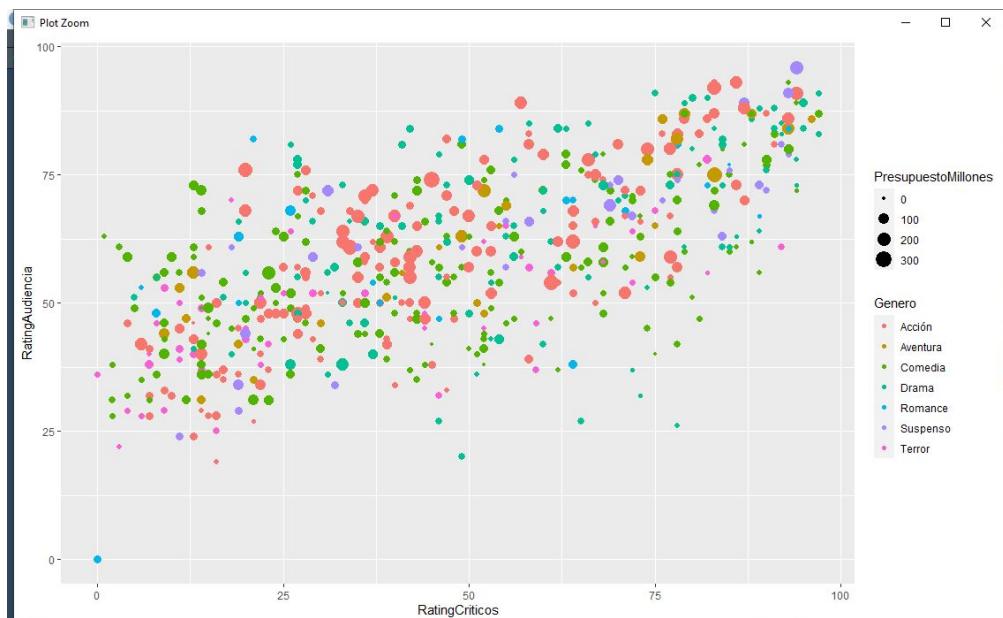
```
#Para agregar color usamos aes()
```

```
ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudencia, color=Genero))+  
  geom_point()
```



#Para poder alterar el tamaño de los puntos

```
ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudiencia, color=Genero, size=PresupuestoMillones))+  
  geom_point()
```



Graficando con Capas (geometrías)

#Graficando con Capas

#Primero vamos a convertir nuestro código de ggplot en un objeto

```
A <- ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudien  
cia, color=Genero,  
size= PresupuestoMillones))
```

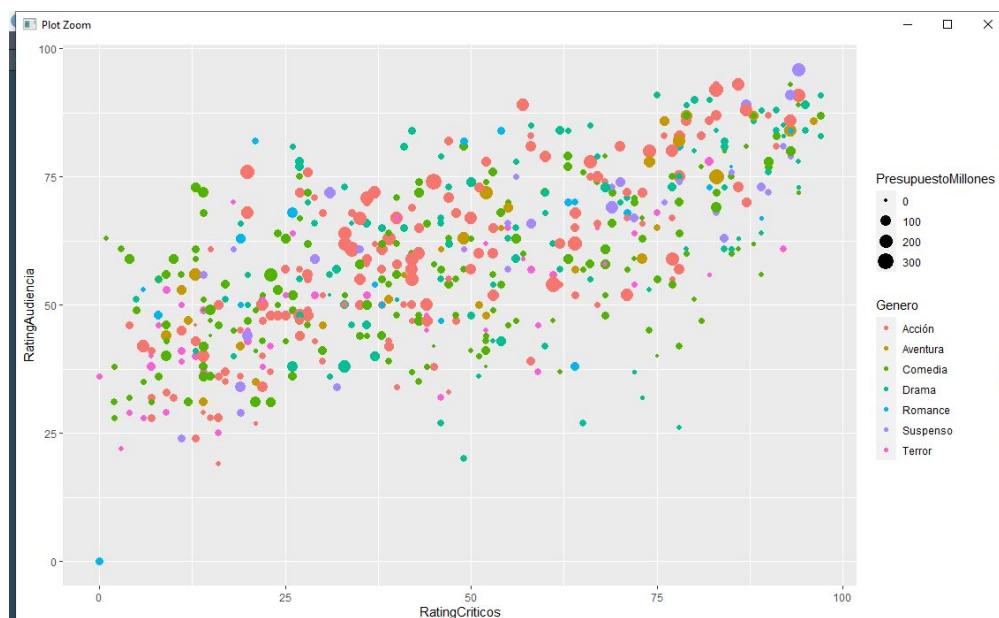
#Si corremos A tendremos solamente un gráfico sin puntos

```
A
```

#Así tendremos de nuevo el último gráfico de la sección anterior

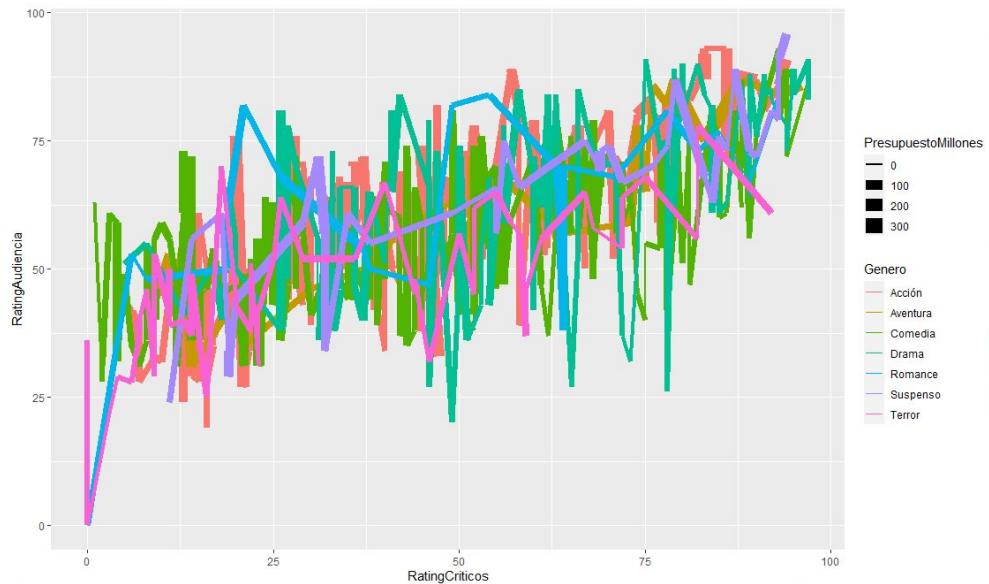
**#Estas son las capas, primero el objeto que corresponde al ggplot(estética) y luego una
capa de tipo geométrica con geom_point()**

```
A + geom_point()
```



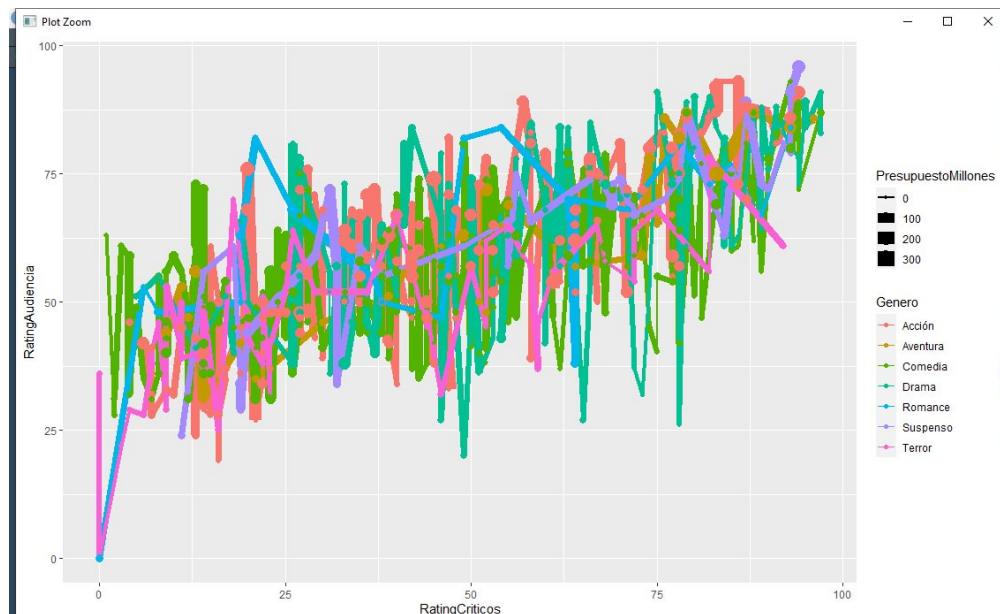
#Lo mismo pero con líneas

```
A + geom_line()
```



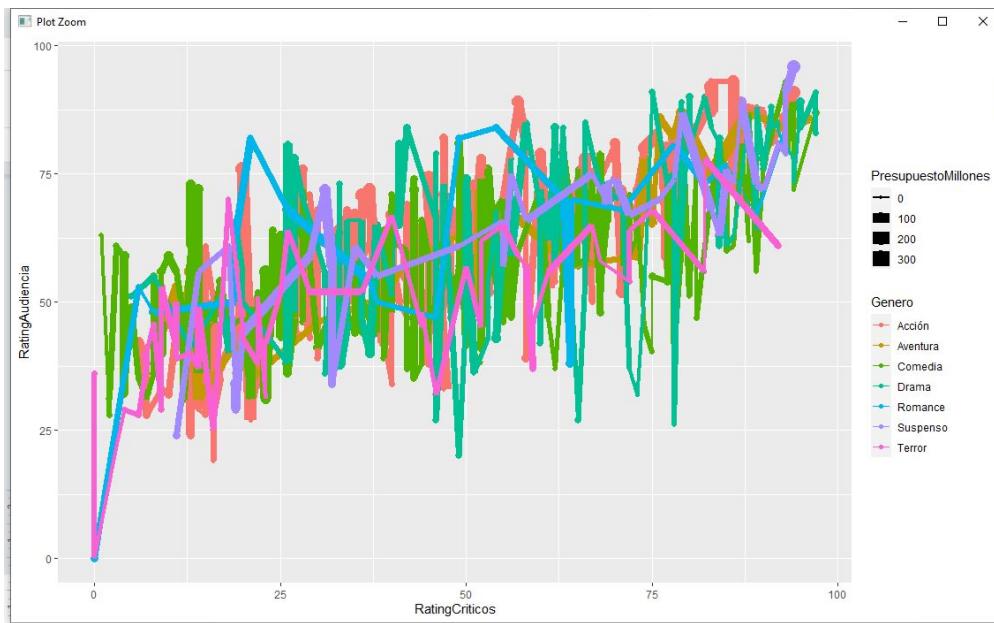
#Podemos combinar las capas, en este caso las línea y los puntos

A + geom_line() + geom_point()



#El orden de las capas es importante como lo ilustra la siguiente gráfica donde no se ven los puntos por ponerlos primero

A + geom_point() + geom_line()



Sobreescribiendo estéticas

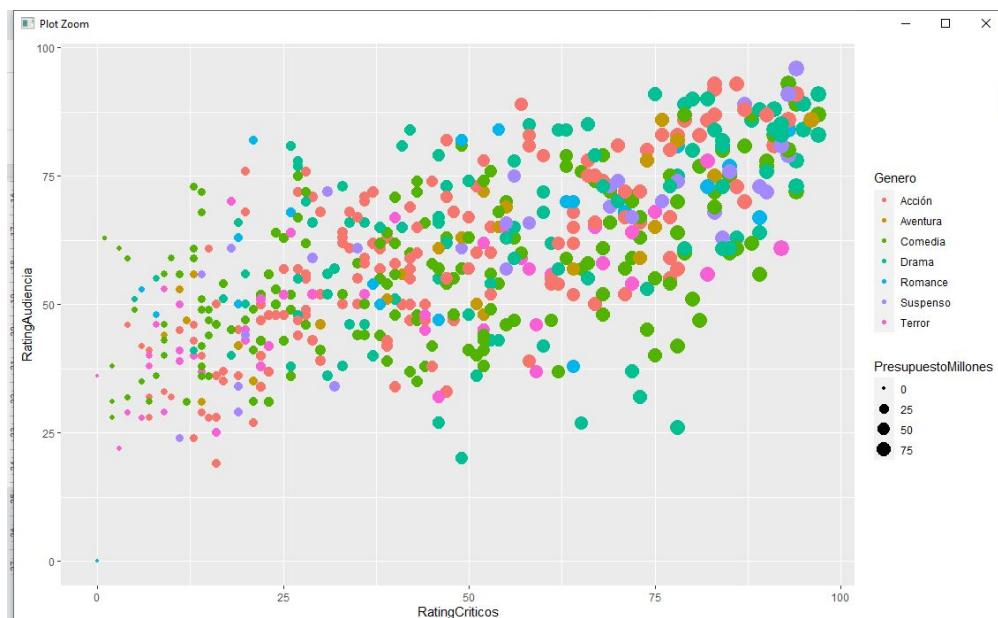
```
#Sobreescribiendo estéticas
```

```
B <- ggplot(data= datos.peliculas, aes(x = RatingCriticos, y = RatingAudien  
cia,  
size= PresupuestoMillones, color=Genero))  
B + geom_point()
```

```
#Debido a que geom_point es una capa sobre el objeto, nosotros podemos sobreescibir las  
estéticas derivadas del objeto
```

```
#Caso 1
```

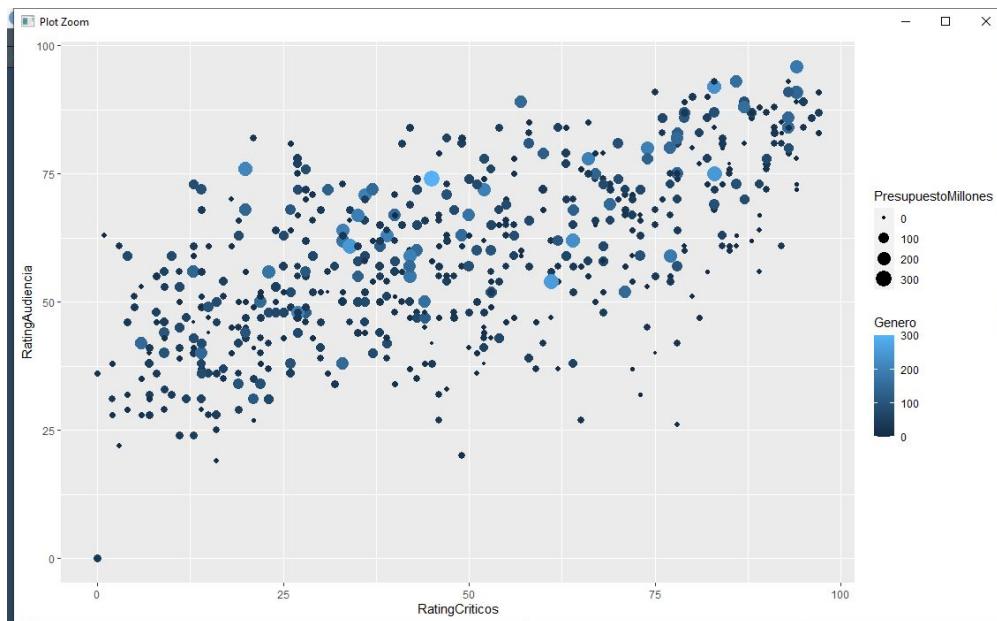
```
B + geom_point(aes(size=RatingCriticos))
```



```
#Caso 2
```

```
B + geom_point(aes(color=PresupuestoMillones))
```

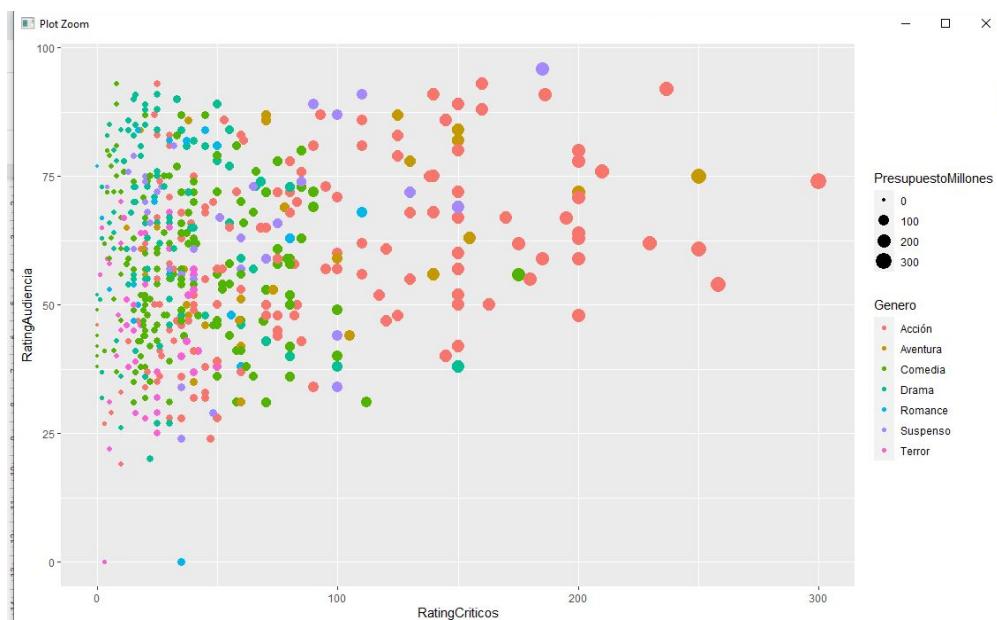
```
#Como se ve, la etiqueta de género que pertenecía a B se mantiene aunque hayamos  
cambiado el color en la capa. Esto es porque el objeto se mantiene y como una capa de  
pintura esta no altera al objeto
```



#Este mismo fenómeno se observa en el siguiente ejemplo, nuestra capa original de estética se mantendrá.

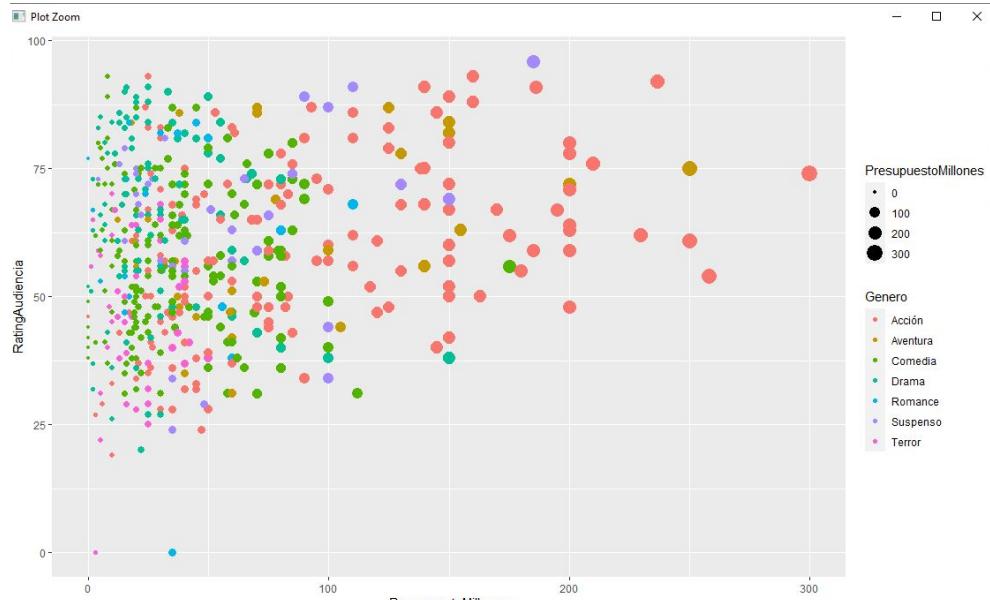
#Se mantuvieron los RatingCriticos, a pesar de que la geometría dice lo contrario

```
B + geom_point(aes(x= PresupuestoMillones))
```



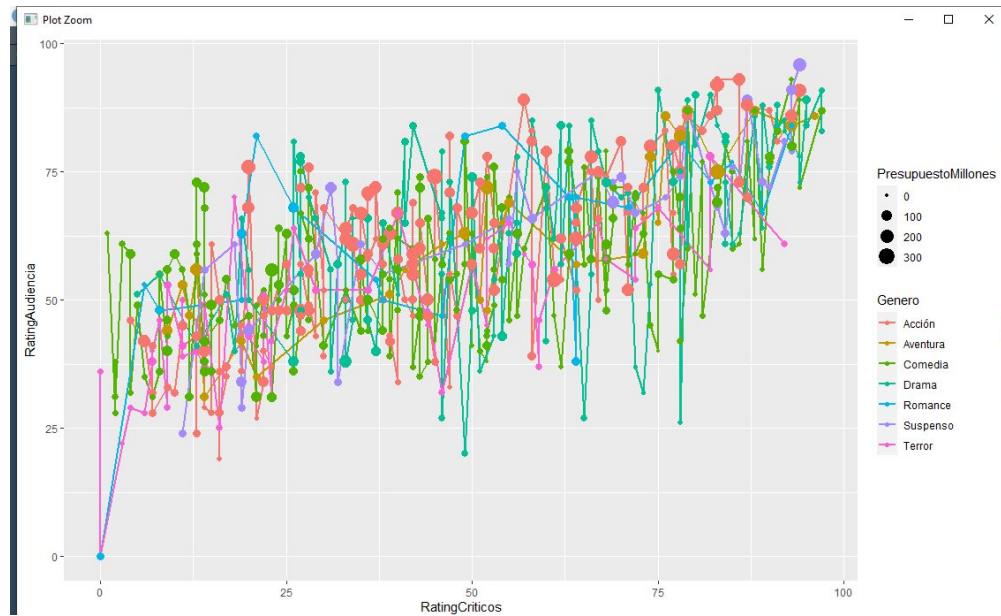
#Una solución a esto es por medio de **xlab()**

```
B + geom_point(aes(x= PresupuestoMillones)) + xlab("PresupuestoMillones")
```



#Volviendo al ejemplo de las líneas, podemos modificar su tamaño para que se puedan ver los puntos fácilmente

```
B + geom_line(size= 1) + geom_point()
```



Mapeando y Estableciendo

#Mapear contra Establecer

```
C <- ggplot(data= datos.peliculas, aes(x=RatingCriticos, y=RatingAudien  
cia))
```

```
C + geom_point()
```

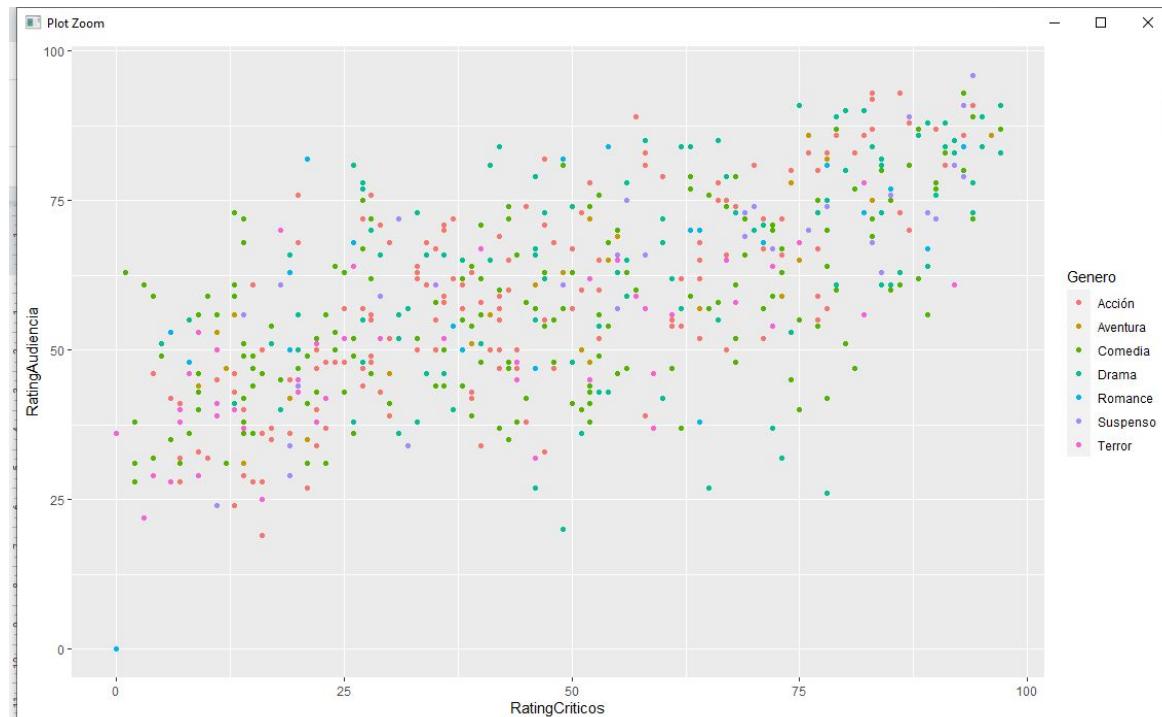
#Agregado de color por mapeo y estableciendo

#Mapeo qua involucra el cambio de algo en función de una variable y depende de aes()

```
C <- ggplot(data= datos.peliculas, aes(x=RatingCriticos, y=RatingAudien  
cia))
```

```
C + geom_point(aes(color=Genero)) #gráfico es de este
```

```
C + geom_point(aes(size= PresupuestoMillones))
```

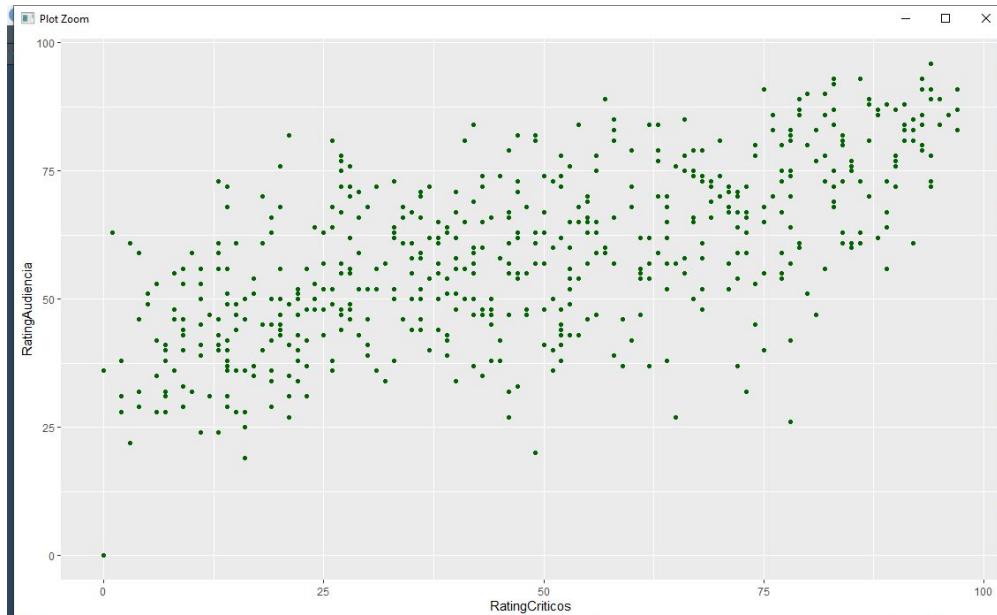


#Estableciendo, es directo y no depende de la variable

```
C + geom_point(color="Darkgreen") #gráfico es de este
```

```
C + geom_point(color= "Blue")
```

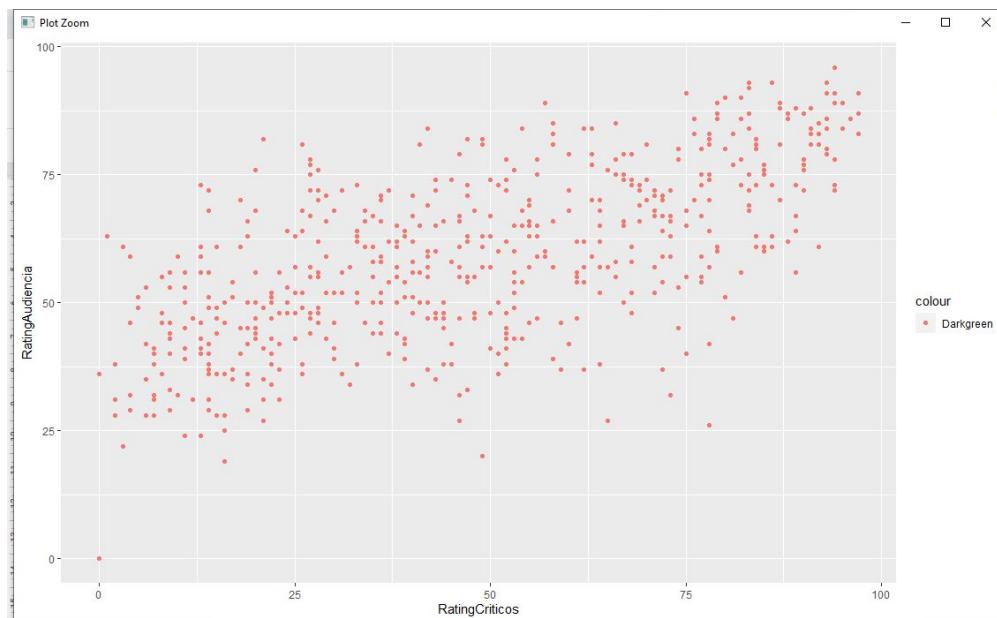
```
C + geom_point(size= 4)
```



```
#ERROR
```

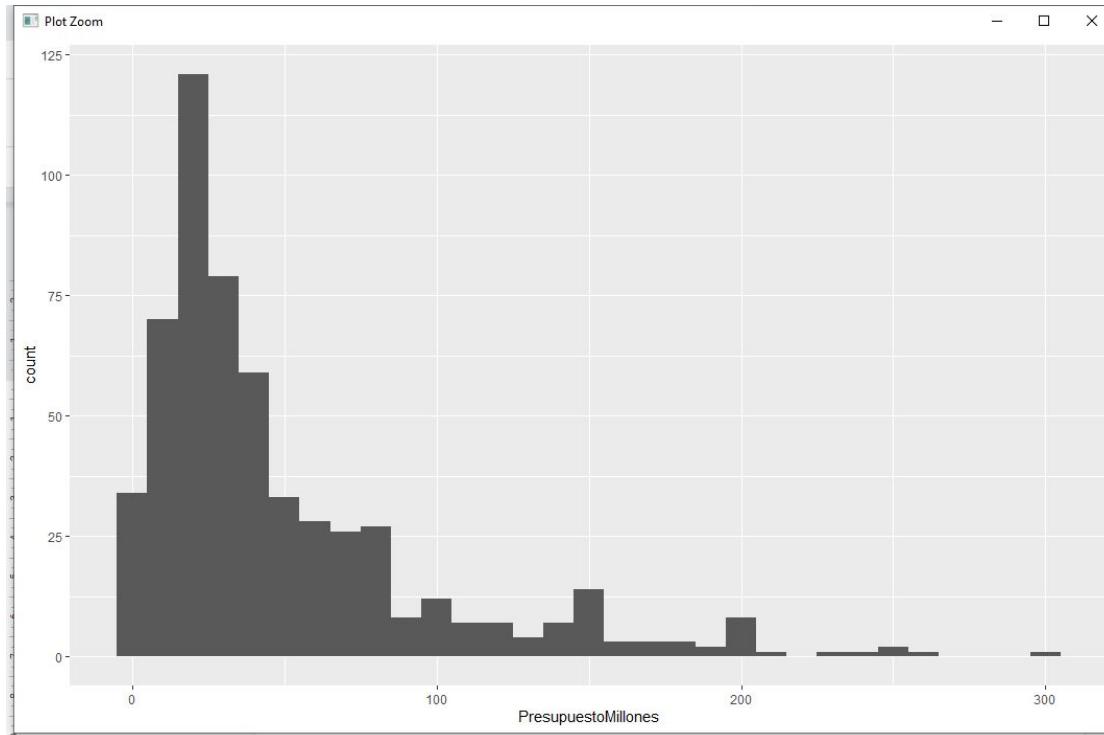
```
C + geom_point(aes(color="Darkgreen")) #gráfico es de este
```

```
C + geom_point(aes(size= 20))
```

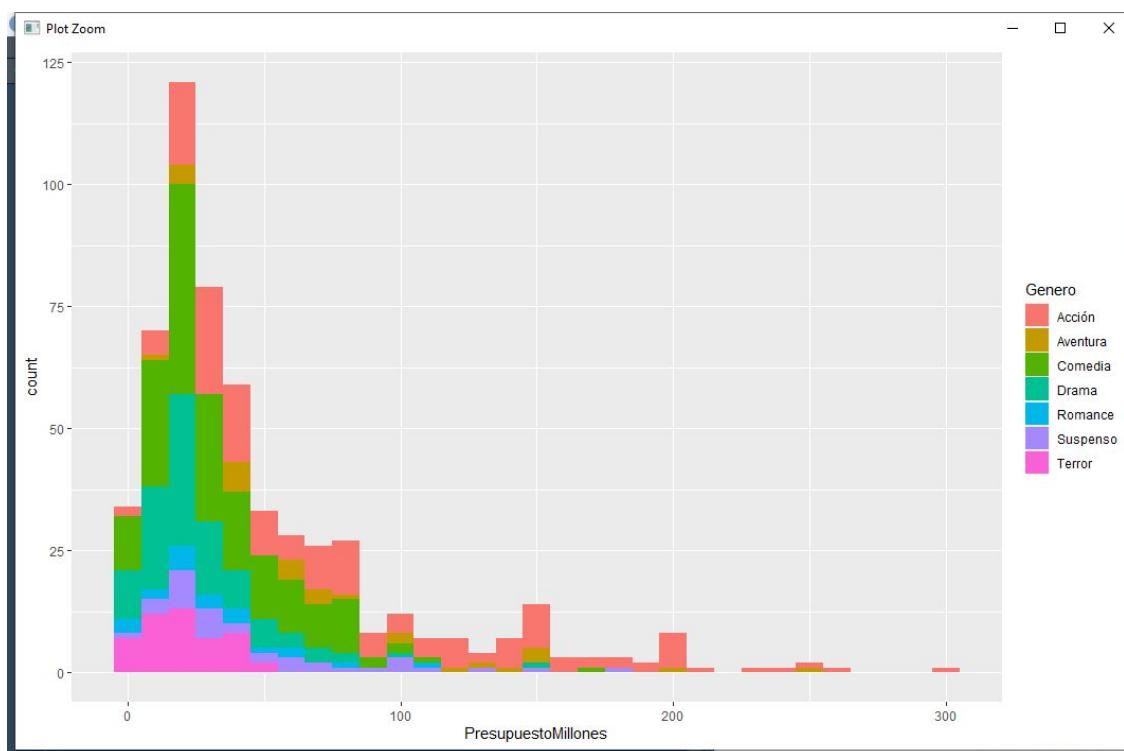
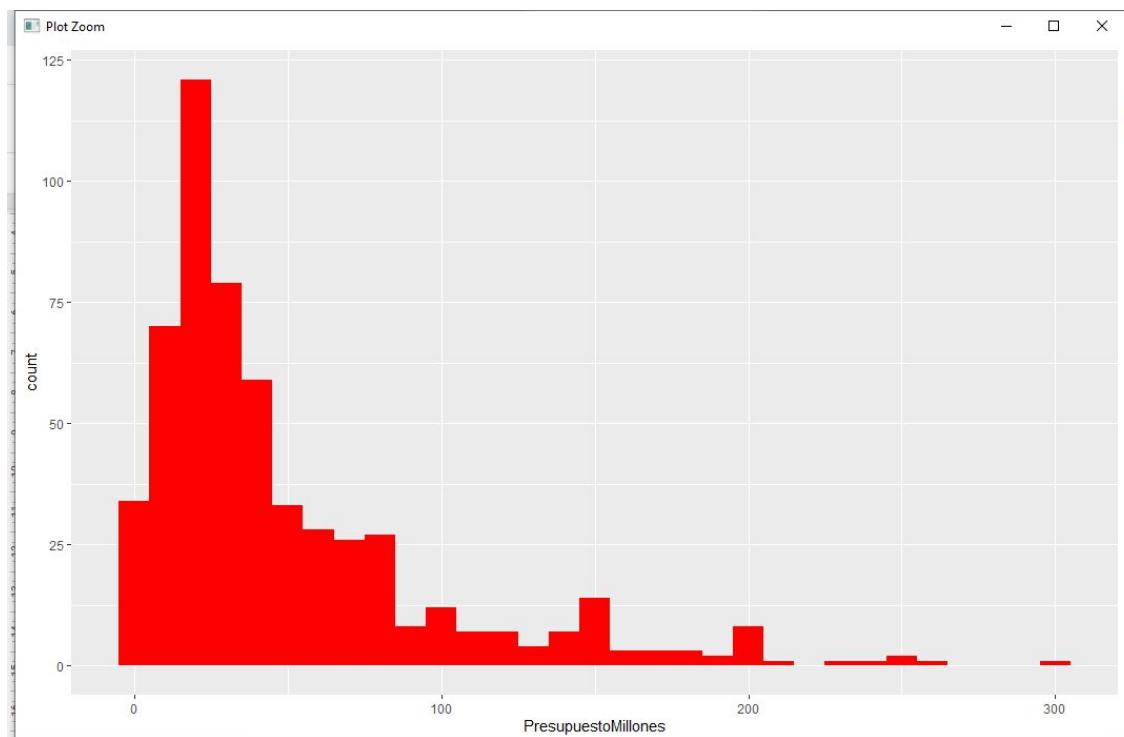


Histogramas y diagramas de densidad

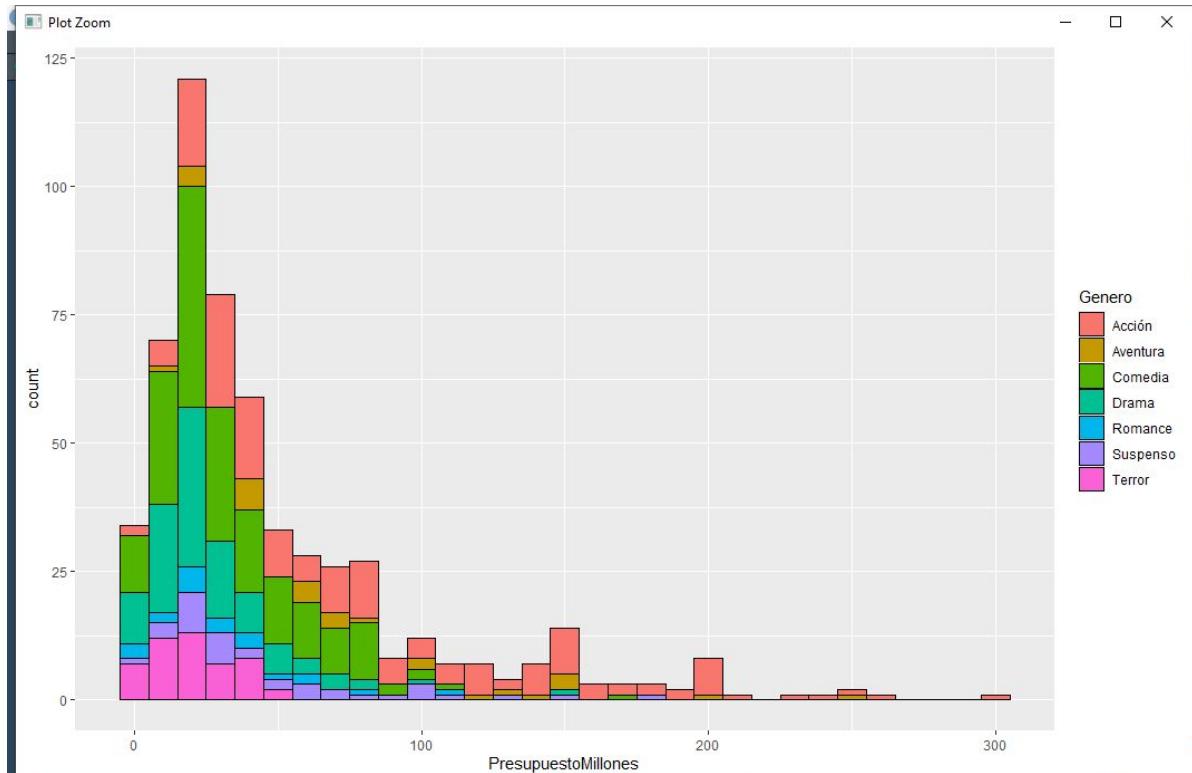
```
#Histogramas y diagramas de densidad  
D <- ggplot(data=datos.peliculas, aes(x=PresupuestoMillones))  
D + geom_histogram(binwidth = 10)  
#binwidth da el ancho del histograma
```



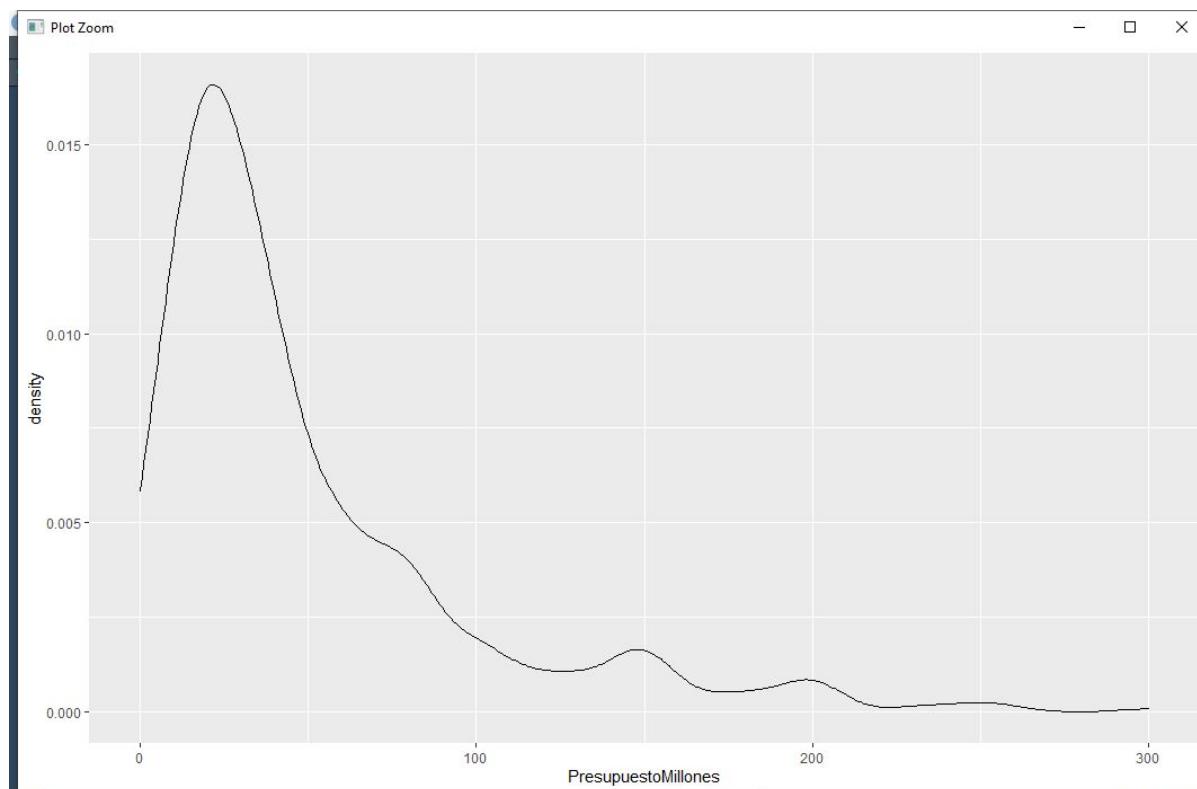
```
#Aregar color  
D + geom_histogram(binwidth = 10, fill ="red") #Establecer  
D + geom_histogram(binwidth = 10, aes(fill = Genero)) #Mapear
```



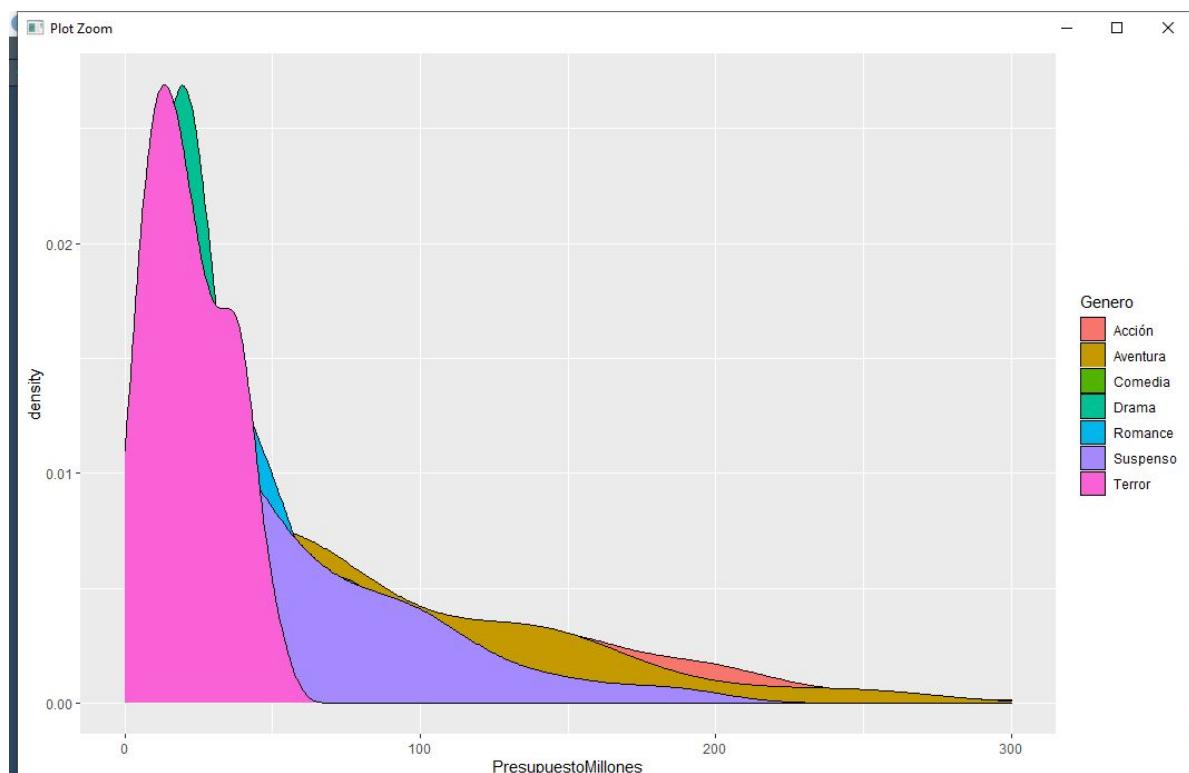
```
#Aregar borde, aquí color da bordes de en lugar del color propiamente  
D + geom_histogram(binwidth = 10, aes(fill = Genero), color = "black")
```



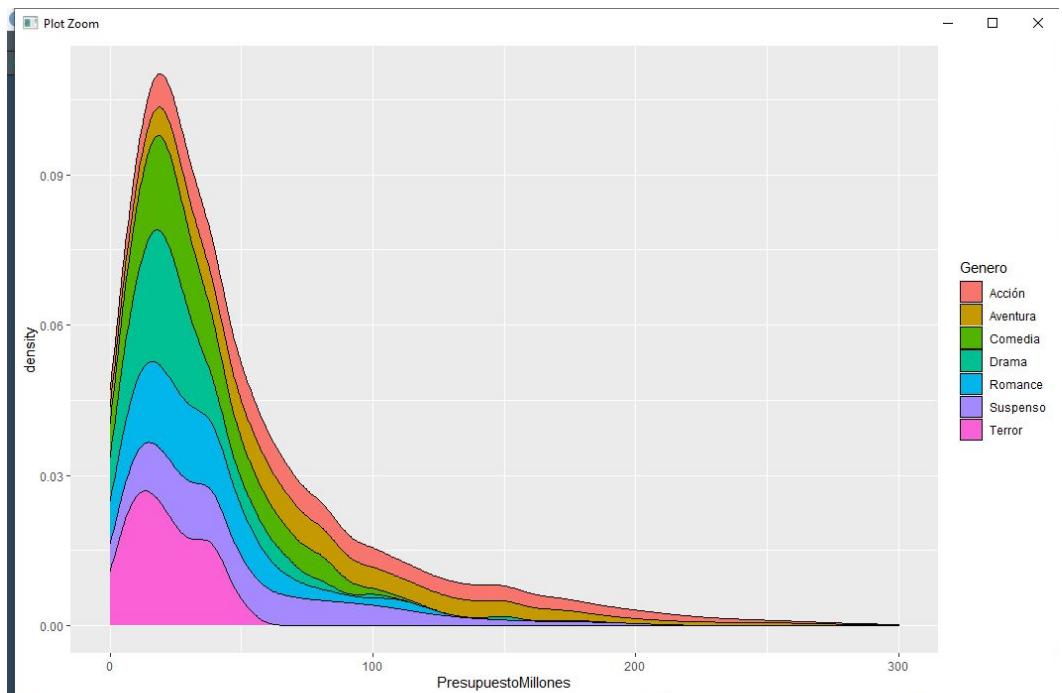
```
#Gráficos de densidad  
D +geom_density()
```



```
D +geom_density(aes(fill= Genero))
```



```
D +geom_density(aes(fill= Genero), position = "stack")
```

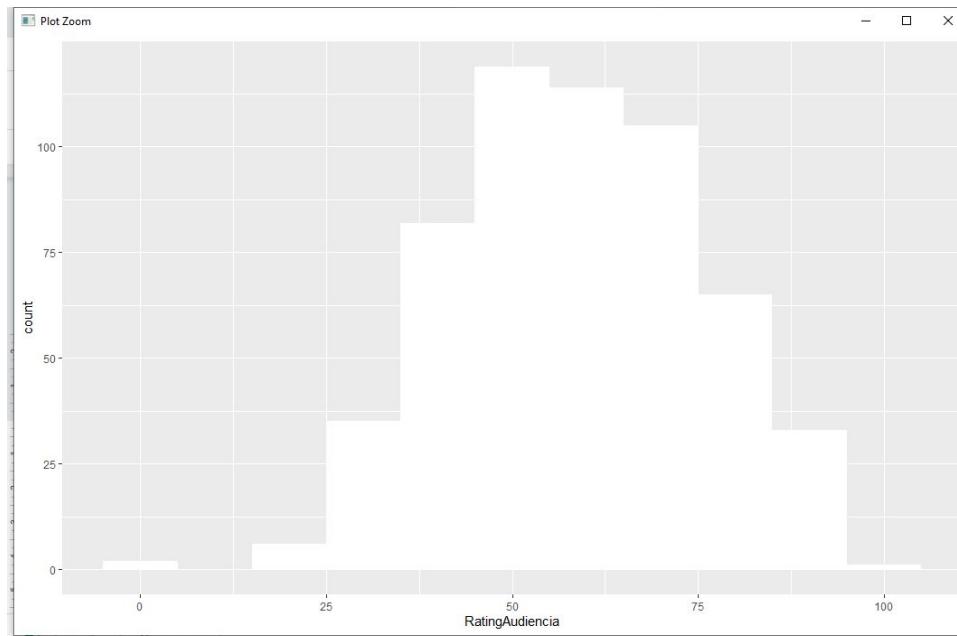


Tips para capa inicial

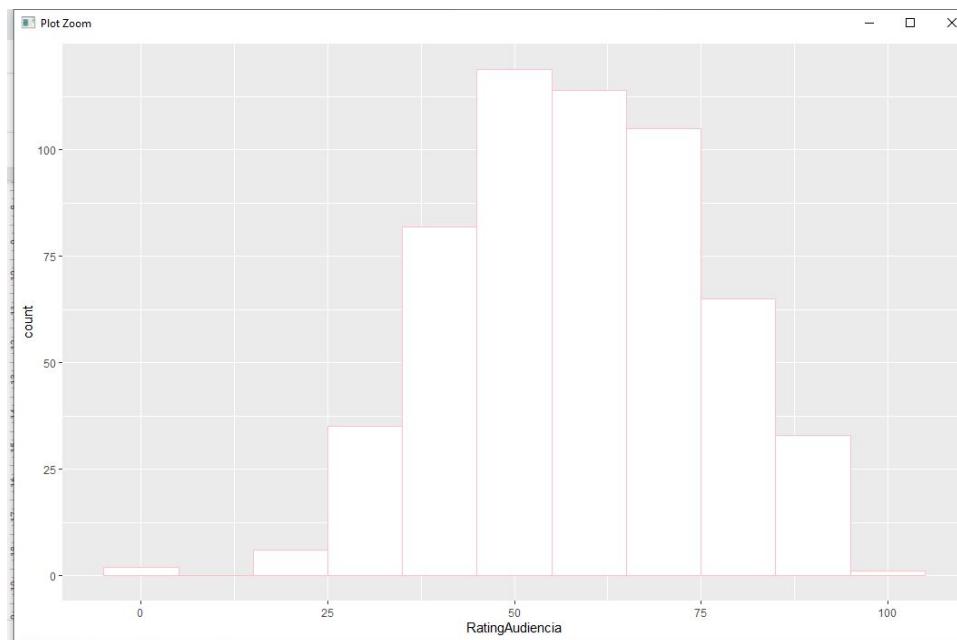
#Tips de capa inicial

```
E<- ggplot(data=datos.peliculas, aes(x=RatingAudiencia))
```

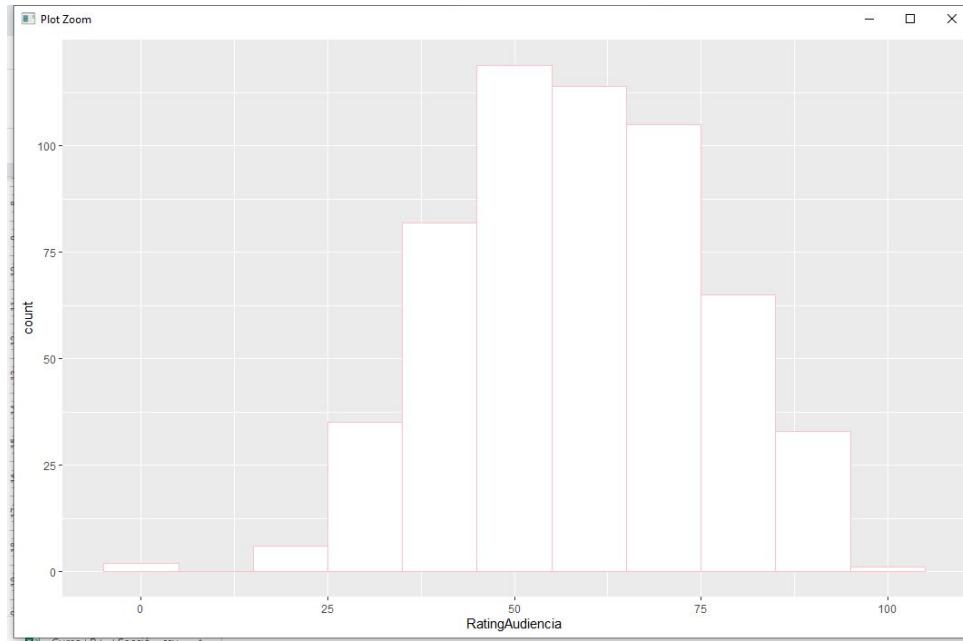
```
E + geom_histogram(binwidth = 10, fill="White")
```



```
E + geom_histogram(binwidth = 10, fill="White", color="pink")
```

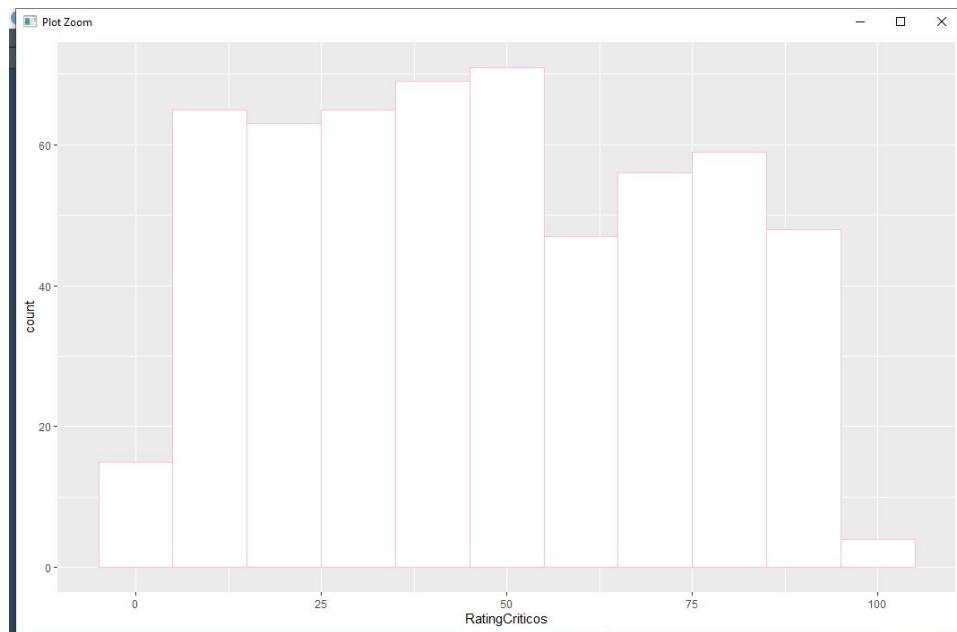


```
#Otra forma de hacer lo de arriba  
E <- ggplot(data=datos.peliculas)  
E + geom_histogram(aes(x=RatingAudencia),  
  binwidth = 10, fill="White", color="pink")
```



```
#Este último es el que tiene mayor flexibilidad, es adecuado cuando vamos a necesitar hacer  
muchas y varias cosas diferentes con el marco de datos, o cuando no estamos muy seguros  
qué hacer con él.
```

```
#Lo mismo pero ahora con los críticos  
E + geom_histogram(aes(x=RatingCriticos),  
  binwidth = 10, fill="White", color="pink")
```



#Una última opción es dejar el esqueleto de ggplot sin nada

```
E <- ggplot()
```

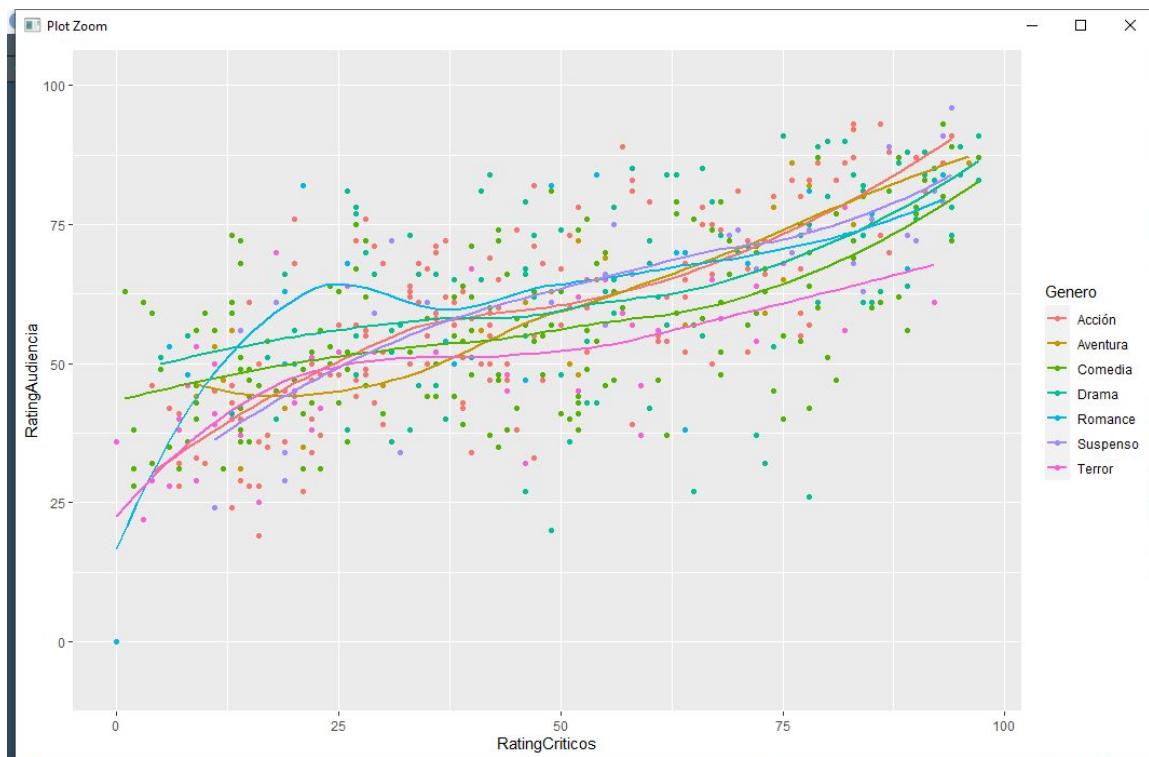
Transformaciones Estadísticas

#Transformaciones estadísticas

#Geom_smooth()

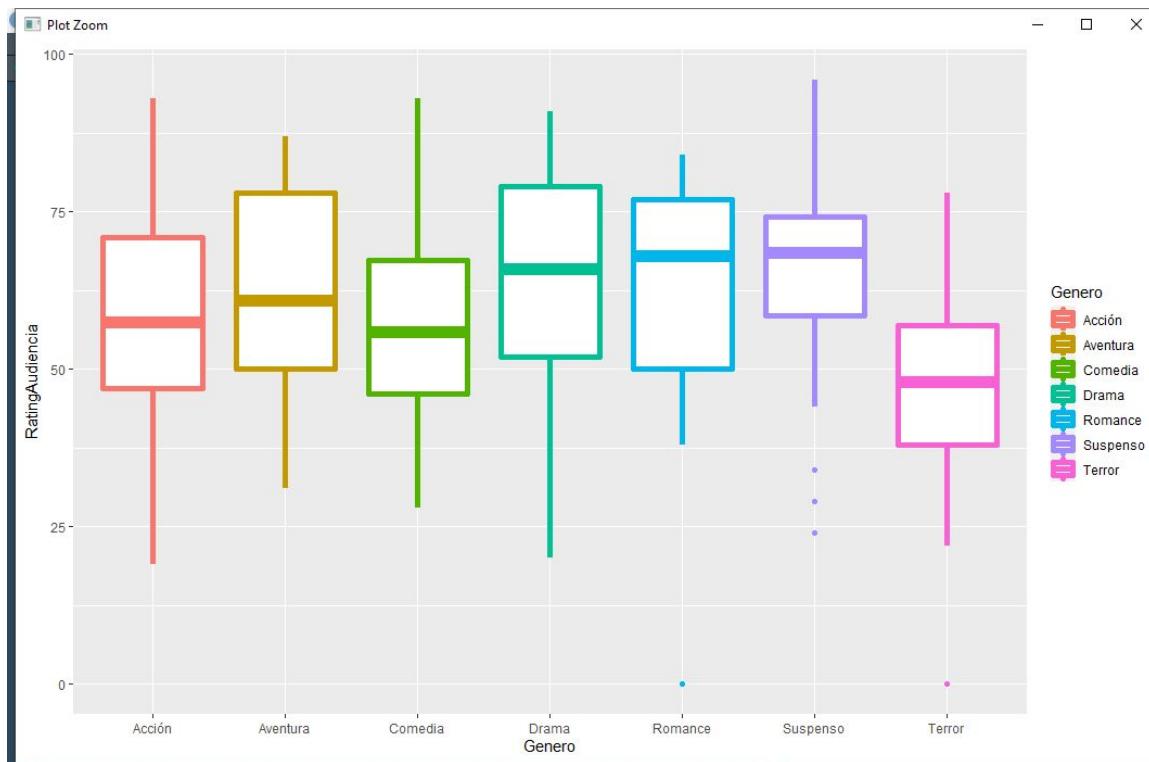
#Mediante estas líneas geom_smooth permite observar patrones que serían complicados de ver cuando los datos están dispersos

```
G <- ggplot(data= datos.peliculas, aes(x= RatingCriticos, y= RatingAudien  
cia,  
color= Genero))  
G + geom_point() + geom_smooth(fill=NA)
```



#Geom_boxplot

```
G2 <- ggplot(data=datos.peliculas, aes(x= Genero, y= RatingCriticos,  
color= Genero))
```



#Otras opciones de geom_boxpot

G2 + geom_boxplot(size= 2) + geom_point()

G2 + geom_boxplot(size=2) + geom_jitter()



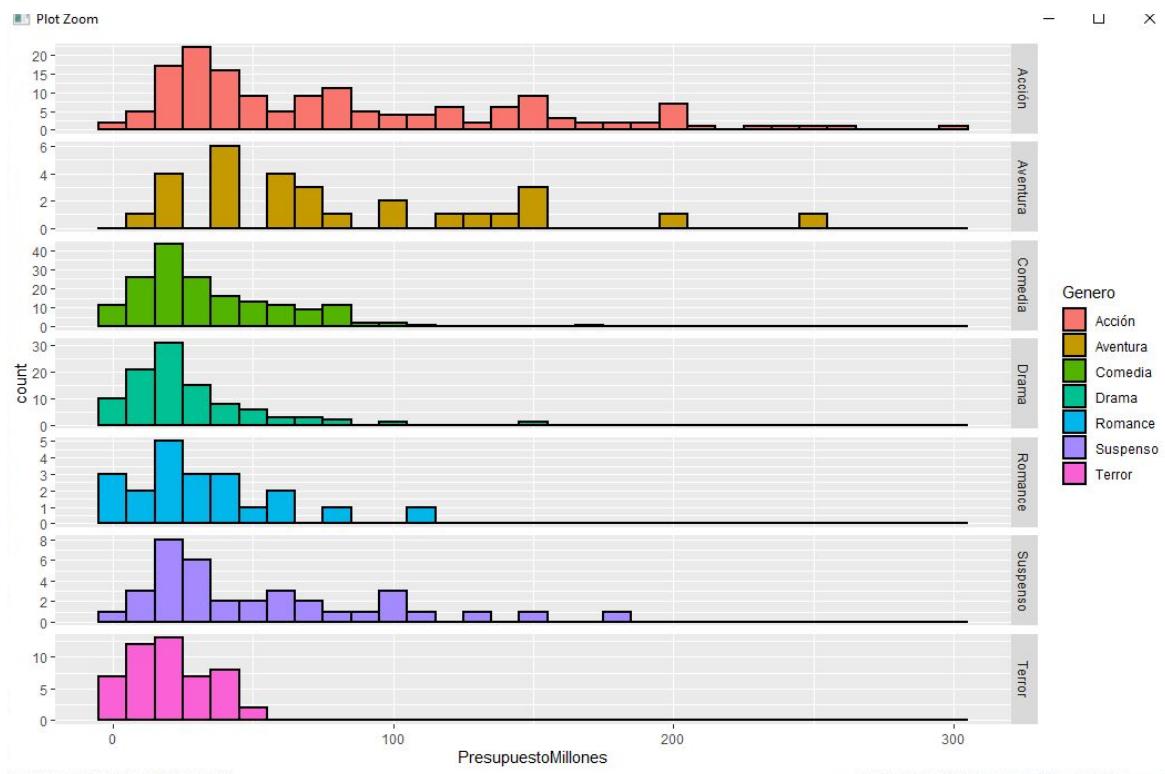
Uso de Facetas

#Usando Facetas

```
H <- ggplot(data=datos.peliculas, aes(x= PresupuestoMillones))  
H + geom_histogram(binwidth =10, aes(fill=Genero), color= "Black", size= 1)
```

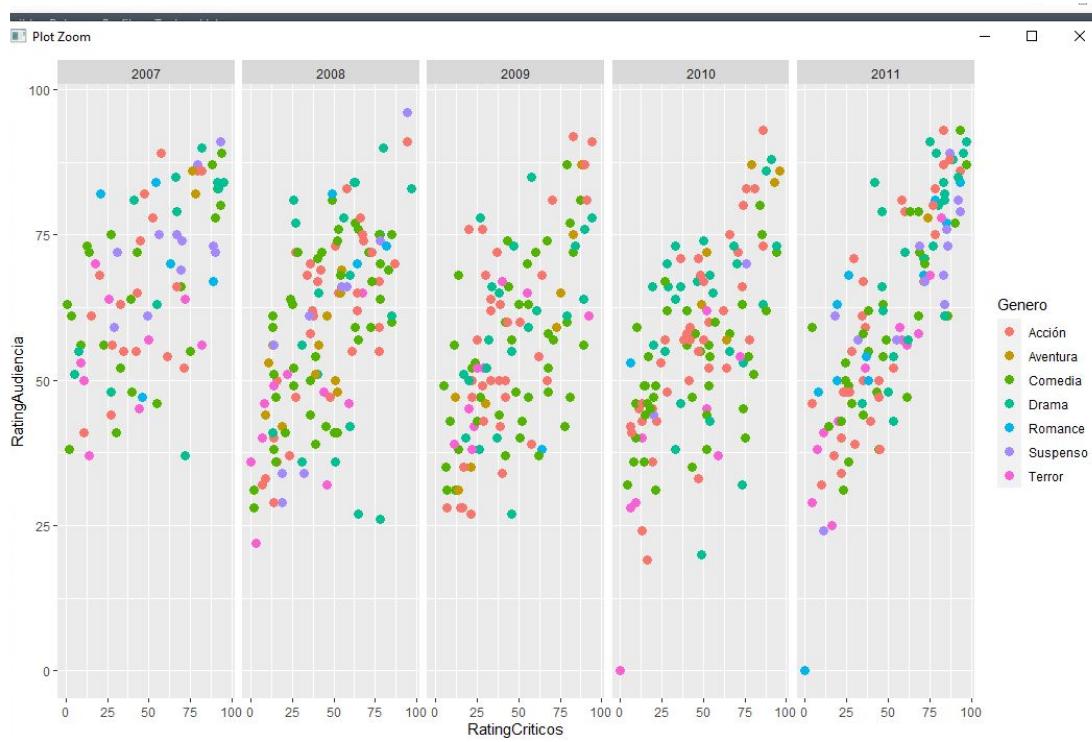
#Facetas, por medio de éstas podemos tener un histograma, o cualquier geometría, para cada uno de los factores de un conjunto de datos. En nuestro caso tenemos uno para cada uno de los géneros

```
H + geom_histogram(binwidth =10, aes(fill=Genero), color= "Black", size= 1) +  
facet_grid(Genero~.,scales = "free")
```



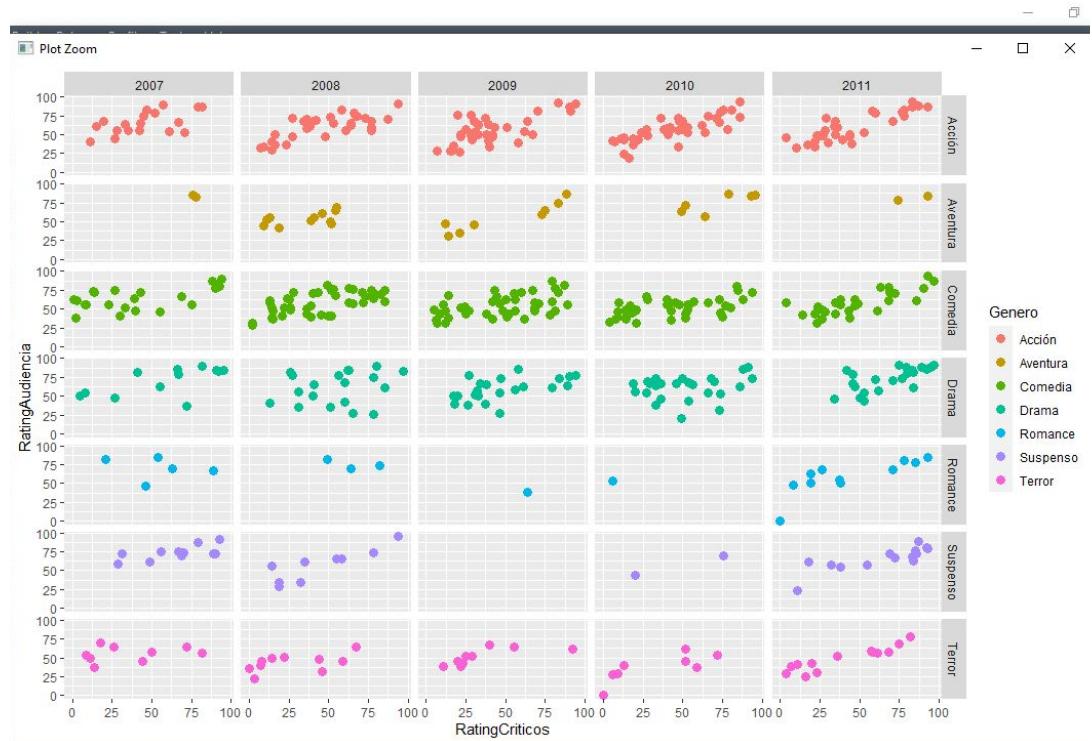
#Facetas con gráficos de dispersión

```
I <- ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudienca,  
color=Genero))  
I + geom_point(size= 3) + facet_grid(.~Año) #Columnas
```

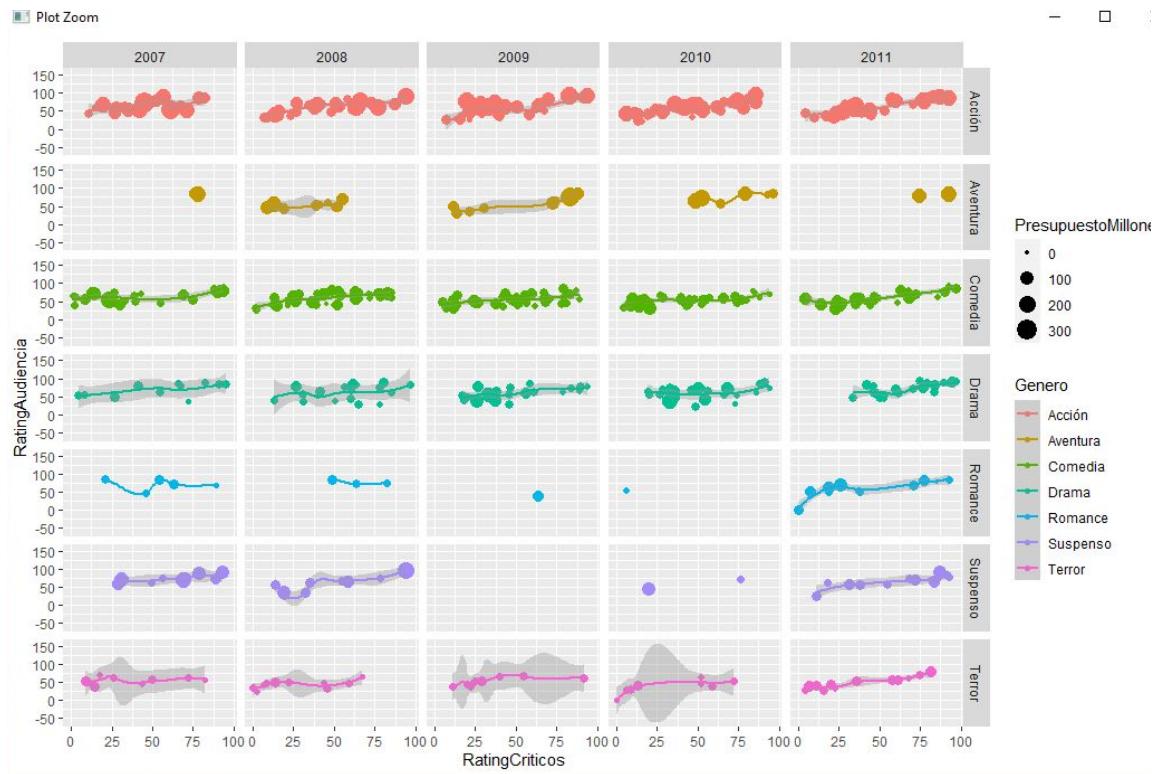


I + geom_point(size= 3) + facet_grid(Año~.) #Filas

I + geom_point(size= 3) + facet_grid(Genro~Año) #Genro en filas y columnas de Año



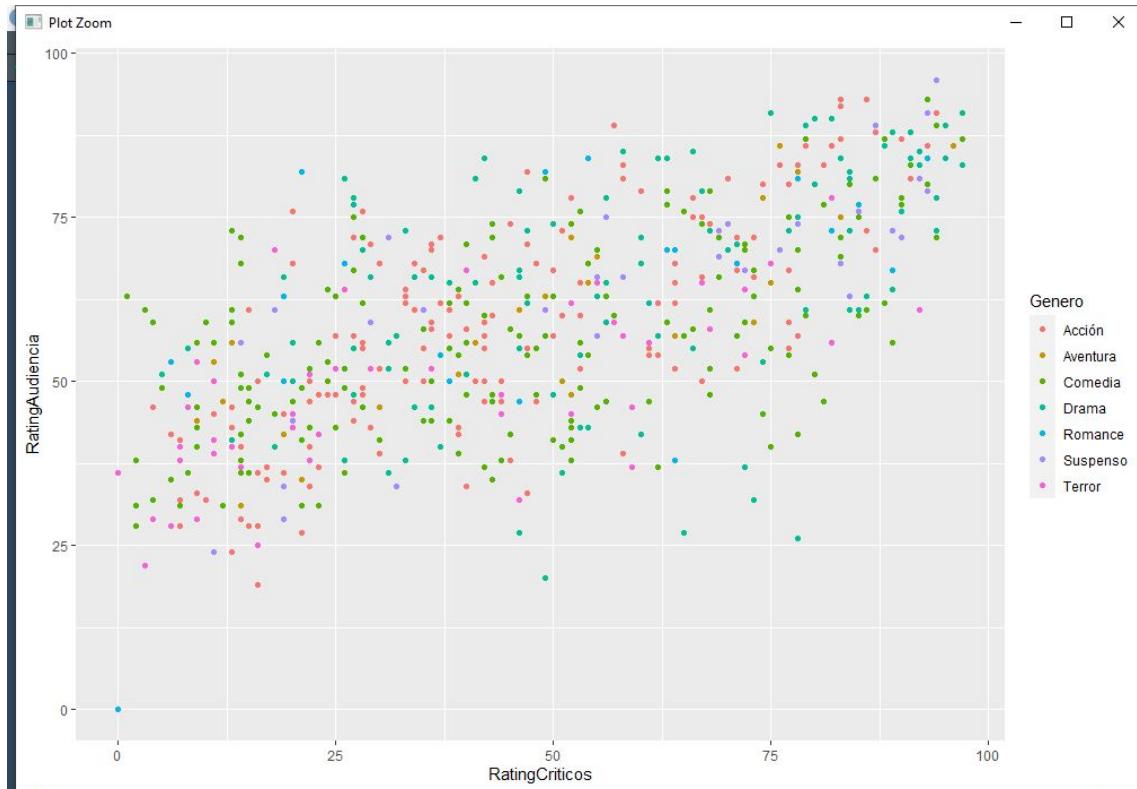
```
I + geom_point(aes(size= PresupuestoMillones)) + facet_grid(Genero~Año) +  
geom_smooth()
```



Coordenadas

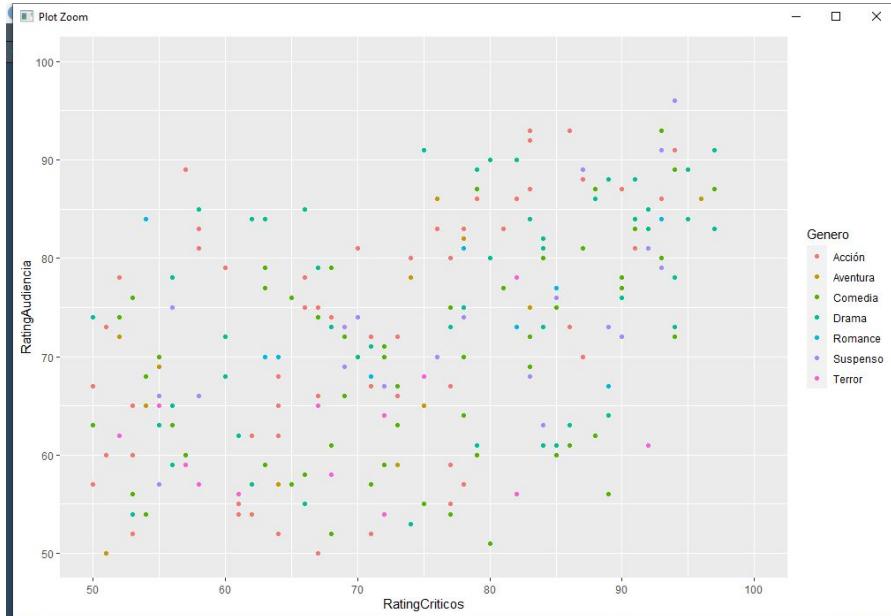
#Coordinandas

```
J <- ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudiencia,  
color=Genero))  
J + geom_point()
```



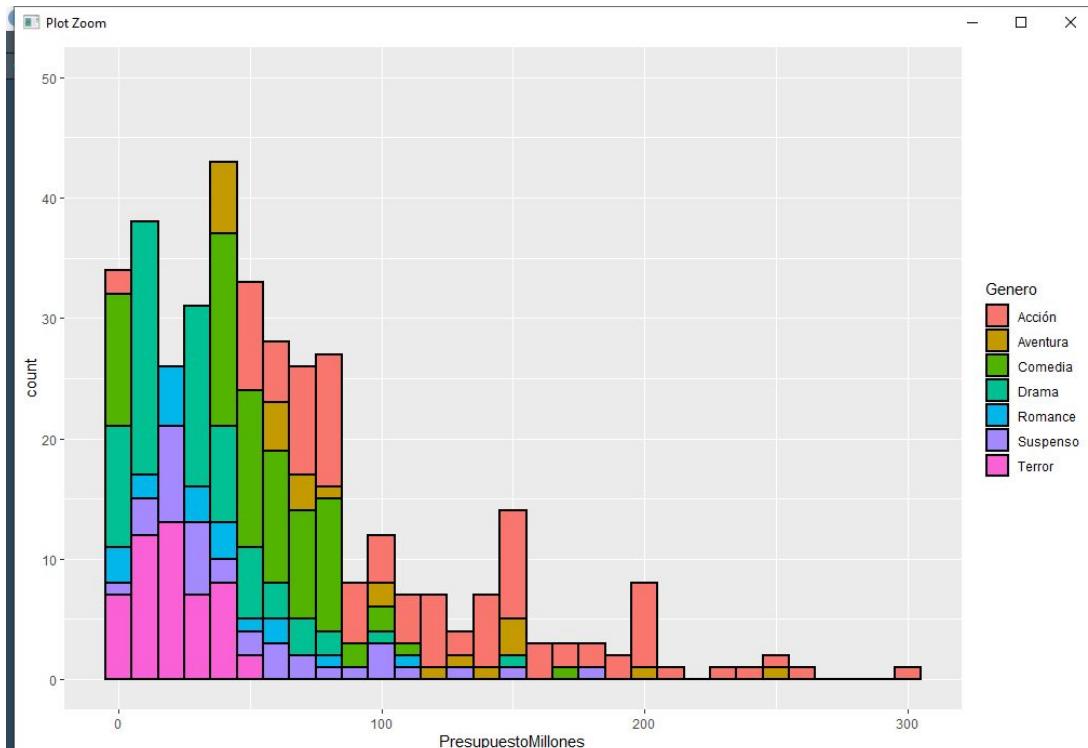
#Si + xlim() y + ylim() podemos lograr seleccionar una sección de todo el sistema de coordenadas

```
J + geom_point() + xlim(50, 100) + ylim(50, 100)
```

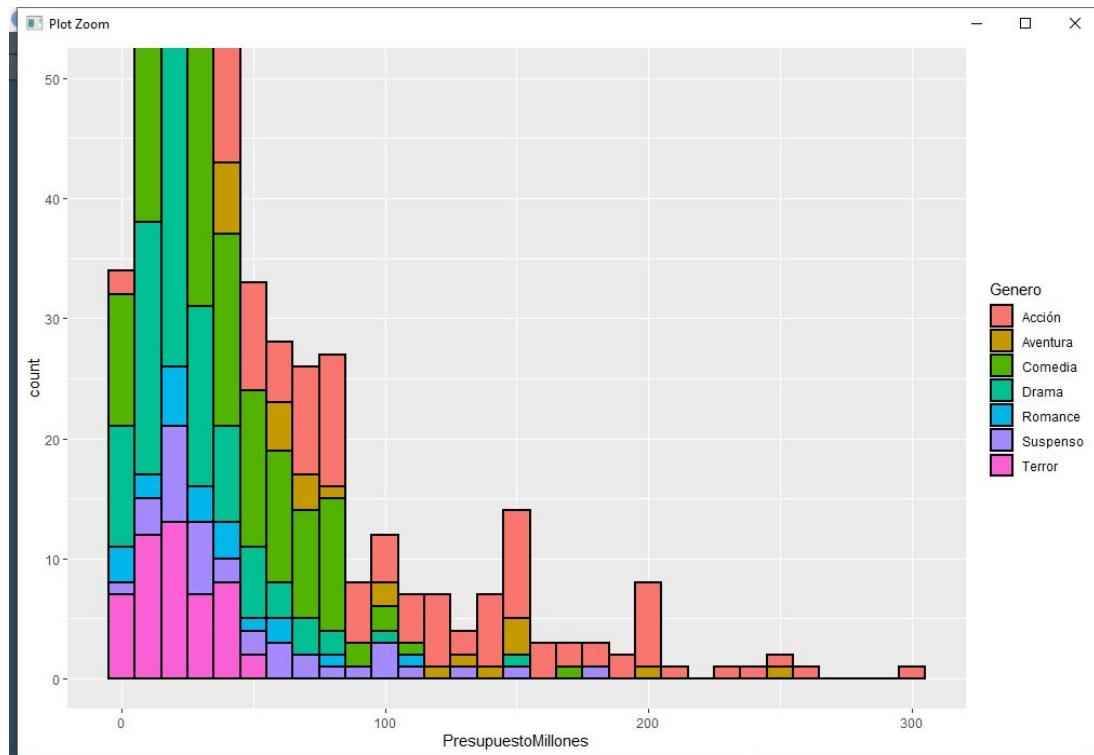


#Sin embargo no siempre es útil, como en el caso de un histograma

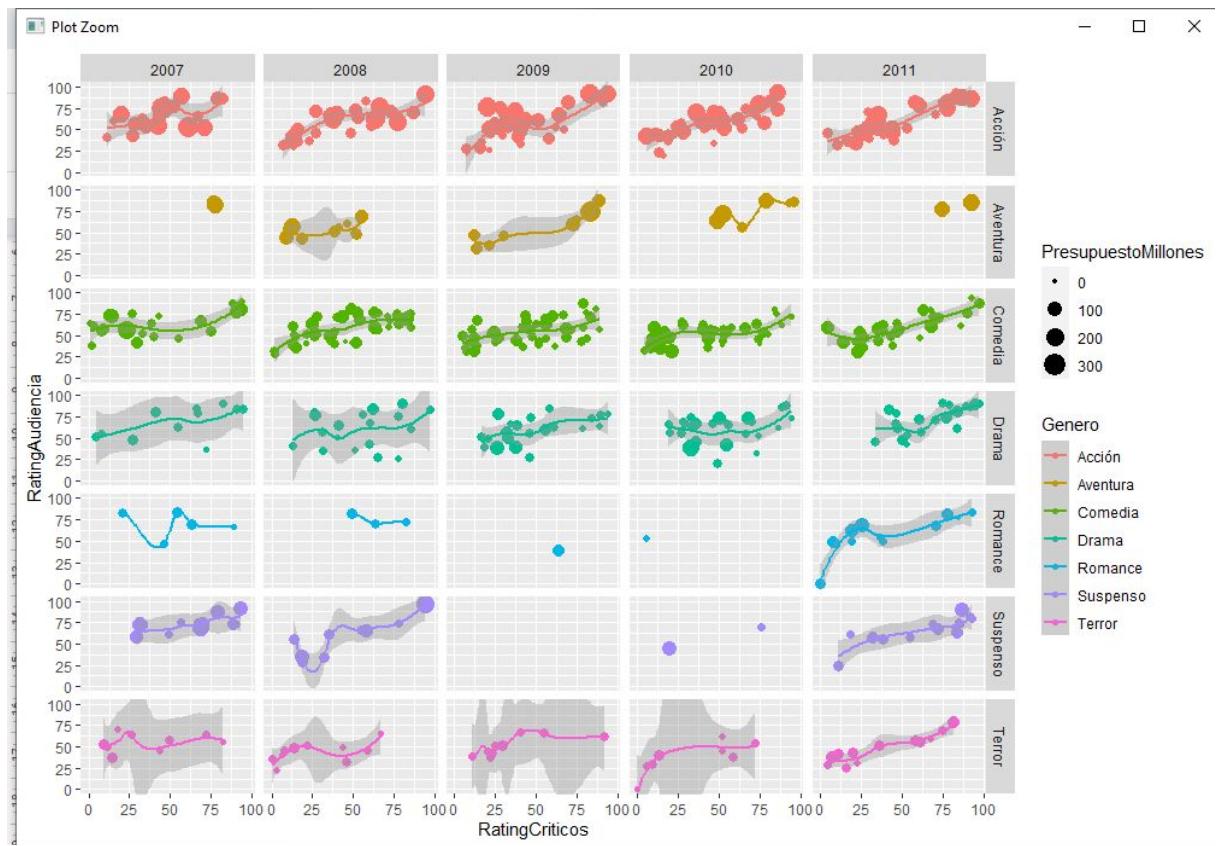
```
K <- ggplot(data=datos.peliculas, aes(x=PresupuestoMillones))
K + geom_histogram(binwidth=10, aes(fill=Genero), color="black", size= 1) +
  ylim(0,50)
```



```
#El problema con esto anterior es que se cortan los datos y se pierde mucha información
#Para resolver esto podemos usar coord_cartesian() que también hace uso de xlim() y ylim()
K + geom_histogram(binwidth=10, aes(fill=Genero), color="black", size= 1) +
coord_cartesian(ylim=c(0,50))
```

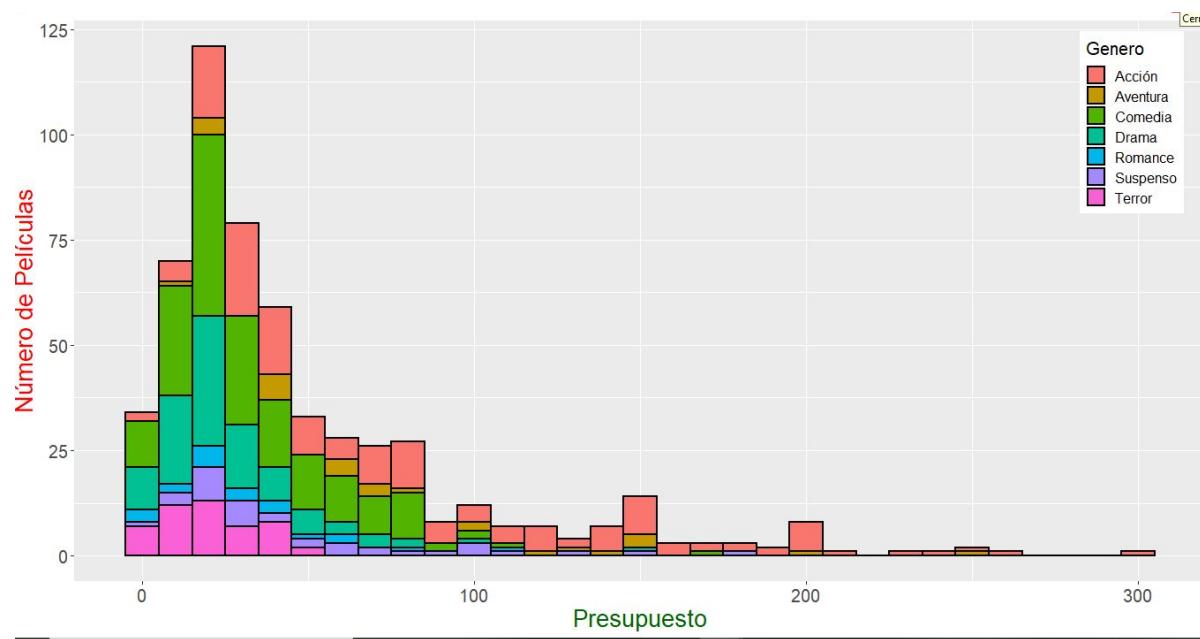


```
L <- ggplot(data=datos.peliculas, aes(x=RatingCriticos, y=RatingAudiencia, color=Genero))
L + geom_point(aes(size=PresupuestoMillones)) + geom_smooth() +
facet_grid(Genero~Año) + coord_cartesian(ylim=c(0,100))
```



Formato

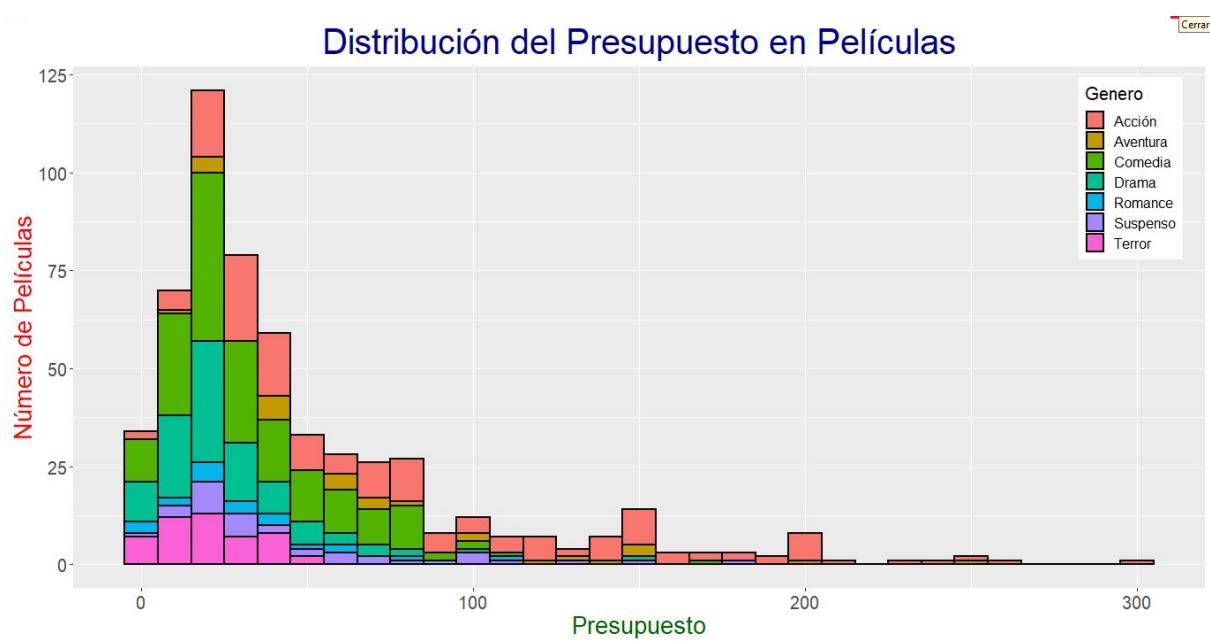
```
#Formato  
N + xlab("Presupuesto") + ylab("Número de Películas") +  
  theme(axis.title.x = element_text(color = "Darkgreen", size=20),  
        axis.title.y = element_text(color= "Red", size=20),  
        axis.text = element_text(size=15), #si no pongo .y o .x se modifican ambos  
        legend.title=element_text(size=15),  
        legend.text = element_text(size= 12),  
        legend.position = c(0.98,0.98), #Determina la posición en el gráfico  
        legend.justification = c(1,1)) #Ancla un punto de referencia para que no se salga del  
margen
```



```
#Título del diagrama
```

```
N + xlab("Presupuesto") + ylab("Número de Películas") +  
  ggtitle("Distribución del Presupuesto en Películas") +  
  theme(axis.title.x = element_text(color = "Darkgreen", size=20),  
        axis.title.y = element_text(color= "Red", size=20),  
        axis.text = element_text(size=15), #si no pongo .y o .x se modifican ambos
```

```
legend.title=element_text(size=15),  
legend.text = element_text(size= 12),  
legend.position = c(0.98,0.98), #Determina la posición en el gráfico  
legend.justification = c(1,1), #Ancla un punto de referencia para que no se salga del  
margen  
plot.title = element_text(color = "darkblue", size = 30, hjust = .5)  
)
```

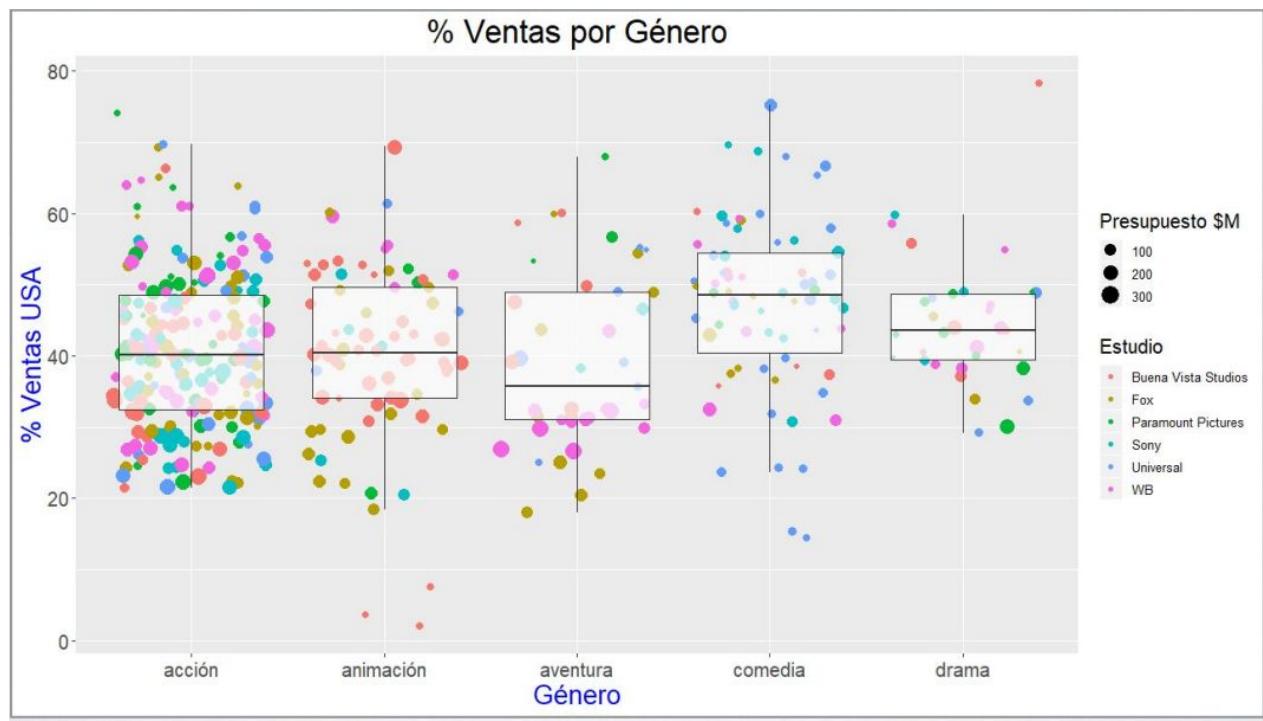


Práctica Final

La página de reseñas de películas quedó muy satisfecha con tu entregable pasado y ahora tiene un nuevo requisito para ti.

El consultor pasado creó un gráfico para ellos, el cual puedes ver en la siguiente diapositiva. Sin embargo, el código utilizado para crear el diagrama se ha perdido y no lo pueden recuperar. Tu tarea es volver a escribir el código para crear el gráfico lo más parecido que se pueda al original.

Te han dado un set de datos nuevo.



```
#Importa los datos
peliculas <- read.csv(file.choose())

#Análisis Exploratorio

head(peliculas) #filas superiores
summary(peliculas) #resumen de las columnas
str(peliculas) #estructura del set de datos

#Activar GGPlot2
#Usar install.package("ggplot2") en caso de no tener el paquete descargado
library(ggplot2)

#Fuera de alcance pero esta interesante:
ggplot(data=peliculas, aes(x=Día.de.la.Semana..lanzamiento.)) + geom_bar()
#¿Te das cuenta? No ha habido estrenos de películas en un Lunes.

#Ahora vamos a filtrar nuestro set de datos para dejar únicamente
#los Géneros y los Estudios en los que estamos interesados
#Empezaremos con el filtro de Género. Usaremos el operador lógico
#"or" para seleccionar múltiples Géneros:
filtro1 <- (peliculas$Género == "acción") | (peliculas$Género == "aventura") |
(peliculas$Género == "animación") | (peliculas$Género == "comedia") | (peliculas$Género ==
== "drama")

#Ahora hagamos lo mismo para los Estudios:
filtro2 <- (peliculas$Estudio == "Buena Vista Studios") | (peliculas$Estudio == "WB") |
(peliculas$Estudio == "Fox") | (peliculas$Estudio == "Universal") | (peliculas$Estudio ==
== "Sony") | (peliculas$Estudio == "Paramount Pictures")

#Aplica los filtros de las filas al marco de datos
peliculas2 <- peliculas[filtro1 & filtro2,]
```

```
#Prepara los datos del gráfico y las capas de estéticas
#Nota que no le cambiamos el nombre a las columnas
#Usa str() o summary() para encontrar el nombre correcto de las columnas
str(peliculas2)
p <- ggplot(data=peliculas2, aes(x=Género, y=Venta...USA))
p #No pasa nada porque se necesita una geometría

#Agrega una capa con geometría de puntos
p + geom_point()

#Puedes agregar un boxplot en lugar de los puntos
p + geom_boxplot()

#Nota que los valores atípicos son parte de la capa del boxplot
#Usaremos esa observación después (*)

#Agrega los puntos
p + geom_boxplot() + geom_point()
#No es exactamente lo que estábamos buscando

#Cambia los puntos por el jitter
p + geom_boxplot() + geom_jitter()

#Posiciona el boxplot por encima del jitter
p + geom_jitter() + geom_boxplot()

#Agrega transparencia al boxplot
p + geom_jitter() + geom_boxplot(alpha=0.7)

#Ahora puedes agregar tamaño y color a los puntos:
p + geom_jitter(aes(size=Presupuesto...mill., color=Estudio)) +
  geom_boxplot(alpha=0.7)
#¿Puedes ver los puntos negros que aún están visibles?
#¿De dónde vienen?
```

```
#Son parte del boxplot - ¿Recuerdas la observación (*) que hicimos arriba?
```

```
#Vamos a quitarlos:
```

```
p + geom_jitter(aes(size=Presupuesto...mill., color=Estudio)) +  
  geom_boxplot(alpha = 0.7, outlier.colour = NA)
```

```
#Almacenamos nuestro progreso en un nuevo objeto:
```

```
q <- p + geom_jitter(aes(size=Presupuesto...mill., color=Estudio)) +  
  geom_boxplot(alpha = 0.7, outlier.colour = NA)  
q
```

```
#Elementos que no son datos (non-data ink)
```

```
q <- q +  
  xlab("Género") #título del eje x  
  ylab("% Ventas USA") #título del eje y  
  ggtitle("Domestic Gross % by Genre", ) #título del diagrama  
q
```

```
#Para lo siguiente se requiere theme()
```

```
#Tema
```

```
q <- q +  
  theme(
```

```
#Título de los ejes:
```

```
  axis.title.x = element_text(color="Blue", size=20),  
  axis.title.y = element_text(color="Blue", size=20),
```

```
#Texto de los ejes:
```

```
  axis.text.x = element_text(size=15),  
  axis.text.y = element_text(size=15),
```

```
#Título del gráfico:
```

```
  plot.title = element_text(color="Black",  
                            size=25,
```

```
hjust = 0.5),
```

```
#Título de la Leyenda:
```

```
legend.title = element_text(size=15),
```

```
#Texto de la Leyenda
```

```
legend.text = element_text(size=10)
```

```
)
```

```
q
```

```
#Toque Final. Esto no lo habíamos visto durante el curso
```

```
#Pero de esta manera puedes cambiar individualmente el título de tu leyenda
```

```
q$labels$size = "Presupuesto $M"
```

```
q
```

Domestic Gross % by Genre

