Una guía introductoria a R

Facultad Menor del Departamento de Ciencia Política, Otoño 2019[*]

¿Qué es R?

R es un lenguaje de programación libre y colaborativo. ¿Qué significa esto? Implica que:

- Es completamente gratuito.
- Colaboradores arman librerías que facilitan el uso de R (como dplyr) o hacen que los resultados se vean mejor (como stargazer). Esto significa que si estás intentando hacer algo complicado, alguien probablemente ya hizo un "atajo" y lo ha compartido con todos.

R tiene una curva de aprendizaje un tanto complicada, pero el propósito de esta guía es hacer lo más posible por ayudarte con las cosas básicas que podrías necesitar para tus trabajos. Quizás eventualmente logres hacer todo desde R (por ejemplo, este pdf se hizo en R).

Si algo no queda en claro en esta guía, te invitamos a escribirnos un correo en fmenorcpol@gmail.com ó mandarnos un mensaje directo en **Twitter**. Por supuesto, siempre es más fácil resolver estas dudas en persona, por lo que también te invitamos a visitarnos en Facultad Menor.

De no poder resolver tu duda, la comunidad de R en línea es muy amistosa, si no encuentras a alguien que ya haya hecho tu pregunta, ten por seguro que te la contestarán con mucho gusto. (También estamos seguros de que quienes tengan esa misma duda agradecerán que hayas preguntado)

¿Cómo conseguir R?

Para esto es necesario buscar la liga adecuada para descargarlo en tu país. Para facilitar esa labor, puedes simplemente dar click **aquí** (Si no te encuentras en México, recomendamos que busques la liga adecuada al país en que te encuentras **aquí**).

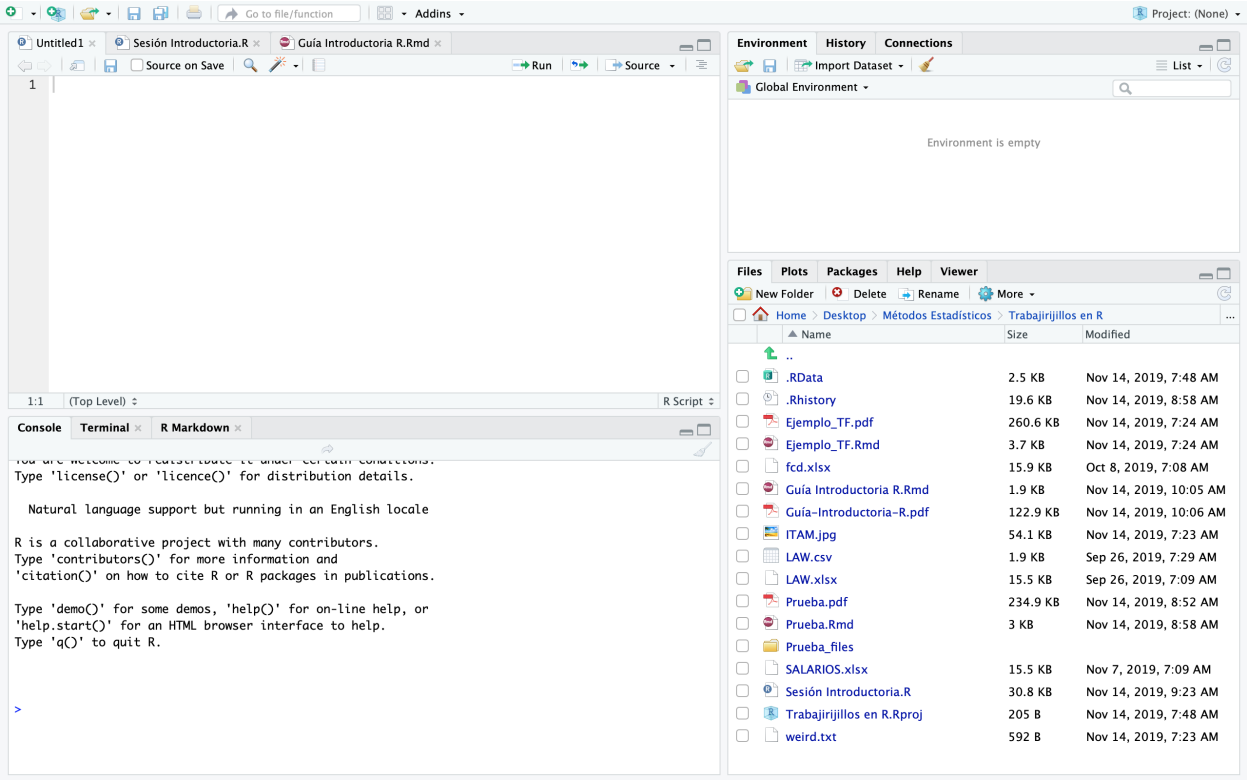
Una vez instalado, puedes utilizar R sin mayor inconveniente. Sin embargo, recomendamos el uso de un visualizador (tal como **RStudio**, que también es gratuito), ya que de lo contrario solo tendrás la consola de R para trabajar, y eso puede resultar algo confuso si apenas estás aprendiendo a usar R. Es por eso que hemos decidido utilizar RStudio a lo largo de esta guía.

¡Ojo! Los visualizadores de R no funcionarán si no haz instalado R primero.

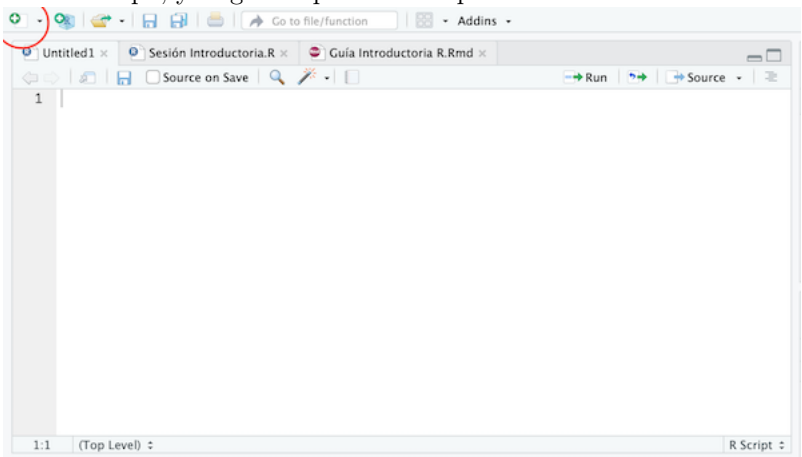
[*]Renata Millet, Edgar Monsiváis y Benjamín Contreras.

Ya tengo RStudio, ¿ahora qué?

Al abrir la aplicación, verás cuatro secciones:



Si solo logras ver tres, esto implica que no hay un Script abierto. Para abrir uno es cuestión de simplemente dar click aquí, y elegir la opción R Script:

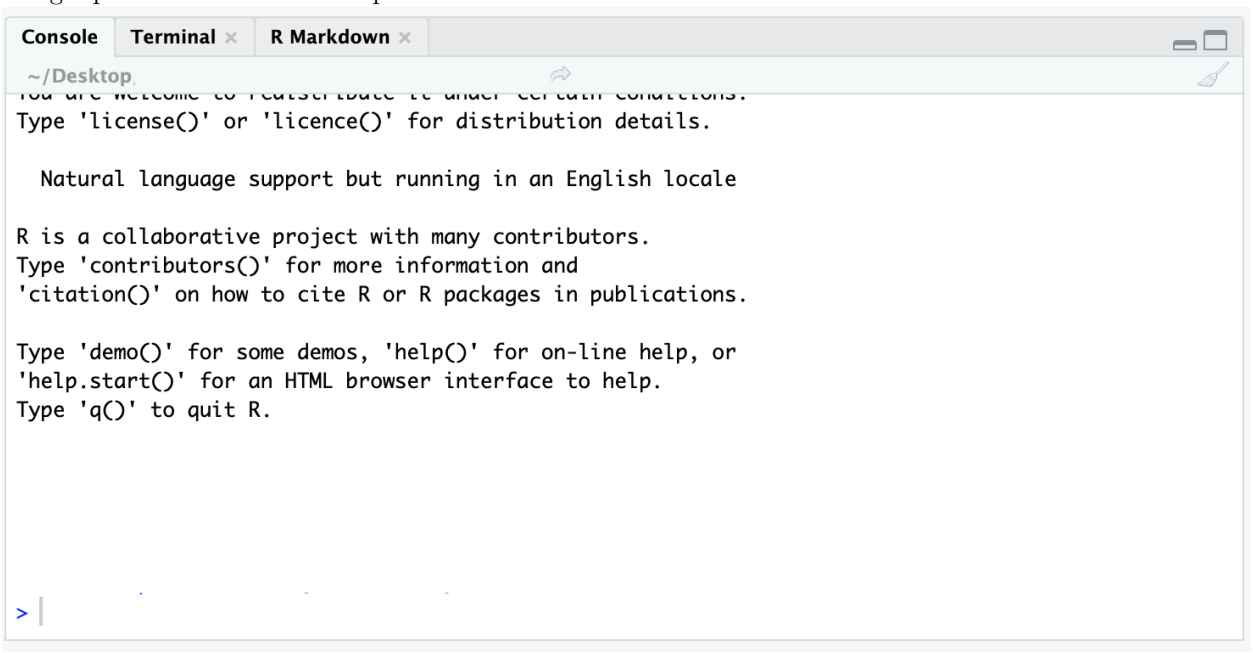


Ahora que ya tienes tus cuatro secciones, examinémoslas una por una:

- La Consola
- El Script
- El "Ambiente" de trabajo
- El Navegador

La Consola

La Consola es el área de trabajo de R, propiamente hablando. Si en algún momento abriste R como tal, en lugar de RStudio, este espacio es exactamente la misma consola. Sirve única y exclusivamente para poner el código que estés utilizando e imprimir resultados.



De momento lo más importante que se encuentra en la imágen es este símbolo >. Esto te indica que R está listo, y esperando instrucciones. Para jugar un momento, intenta escribir "1+1" (sin las comillas) en la consola y oprime enter.

Como podrás haberte dado cuenta, R utiliza símbolos matemáticos con los que ya conoces si has usado Excel antes:

- Suma: +
- Resta: -
- Multiplicación: *
- División: /
- Exponentes: ^

Adicionalmente, al igual que Excel, también utiliza paréntesis para dar prioridad a operaciones. Por ejemplo, el resultado será diferente si corres el comando "1+1/2" a si corres el comando "(1+1)/2".

Si no te aparece el cursor >, implica que R está esperando más instrucciones de algún comando que le diste antes. Si no lo hiciste intencionalmente, simplemente oprime "esc" en tu teclado, y la consola debe regresar a la normalidad.

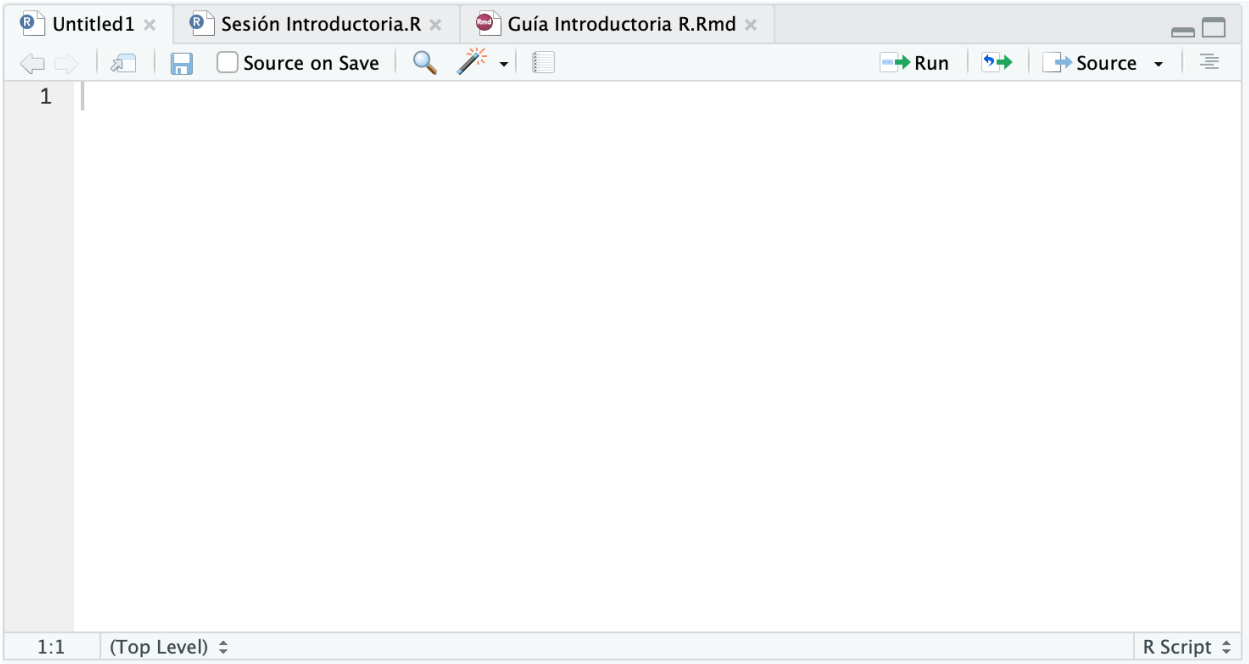
También es importante considerar que R es sensible al uso de mayúsculas y minúsculas. Por dar un ejemplo, si quieres sacar la raíz cuadrada de 4, y escribes el comando Sqrt(4), R te dirá que la función no existe (ya que solo conoce sqrt()).

> Nota: si te aparece un símbolo rojo en la esquina superior derecha de la consola, eso indíca que R aún está procesando el código que le diste. Si es algo particularmente grande, esto puede tardar unos minutos. Espera a que desaparezca antes de intentar darle más instrucciones.

¡Ojo! Todo lo que anotes en la Consola se guardará únicamente para la sesión de R en curso. Esto quiere decir que si cierras R, y lo vuelves a abrir, todo tu trabajo estará perdido. Es por esto que recomendamos anotar tódo tu código en el Script, ya que este se guarda como un archivo en tu computadora.

El Script

Esta pantalla es tu código. Puedes pensarlo como una especie de "borrador" de lo que pondrás en la consola. Como mencionabamos antes, lo que hagas en tu Script puede ser guardado como un archivo .R, lo cual tiene la enorme ventaja de poder trabajar en el mismo proyecto a lo largo de varios días.



Otra gran ventaja de trabajar en el Script es que si tu código llegara a fallar, puedes revisar tu código línea por línea de forma ordenada, lo cual facilita inmensamente econtrar la falla.

Adicionalmente, es posible escribir notas en el Script. Esto se hace mediante el símbolo de #. Todo lo que escribas posterior a este símbolo (en una misma línea de código) aparecerá en verde. Es simplemente un indicador de que R ignorará esa línea al correr el código.

> **Dato curioso: utilizar el símbolo de gato cinco veces seguidas en una misma línea (#####) encierra el código que escribas debajo en una "pestaña". Lo cual te puede ayudar a dividir tu código en secciones.**

Una función del Script (en RStudio) es que puedes correr tu código sin necesidad de pasar a la consola. Esto se hace dando click en la línea que quieres correr y haciendo una de dos cosas:

- Oprimiendo el botón "Run" en la parte superior del Script
- Oprimiendo Ctrl+Enter en Windows ó Cmd+Enter en Mac.

¡Haz la prueba! Escribe "1+1" (sin las comillas) en tu Script, y corre la línea.

Alternativamente, puedes seleccionar varias líneas de código y correrlas con los mismos métodos.

¡Ojo! Es muy recomendable que guardes los cambios que realices en tu Script de forma obsesiva. Pocas cosas tan frustrantes como perder tu código porque se te acabe la batería, o porque RStudio se cierre

El "Ambiente" de Trabajo

Esta sección de RStudio contiene todos los objetos que generes: variables, vectores, tablas, matrices, regresiones, etc.

Algunos objetos son fáciles de crear, por ejemplo, si quisieras guardar un número:

```
x <- 30
```

O si quisieras guardar el resultado de una operación:

```
x <- (15*2)/(50+10)
```

Como podrás notar, no hemos usado el signo de igual. No está mal usarlo, de hecho R te lo acepta, pero existe la posibilidad de que puedas confundirte con otros usos que toma el símbolo, por lo que es preferible usar "<-" para generar variables, vectores, tablas, etc.

Si colocas la creación de un objeto entre paréntesis, R imprimirá el resultado en la consola, en lugar de solo guardarlo:

```
(x <- (15*2)/(50+10))
```

```
## [1] 0.5
```

Otros objetos son generados de esta manera:

(1) Vectores: para hacer un vector hay dos métodos. Creación de una serie, y el comando `c()`

```
y <- 1:10
# Esto genera un vector seriado de 1 a 10
# alternativamente, si quieres poner valores especificos
y <- c(1,5,2,6,3,7,4,8,5,9)
```

Un vector no necesariamente tiene que ser numérico:

```r
# Es muy importante que estén entrecomillados, para que R
# entienda que le estas dando texto y no un comando.
(edos <- c("Puebla", "Coahuila", "Aguascalientes"))
```

```
## [1] "Puebla"         "Coahuila"       "Aguascalientes"
```

(2) Matrices: creadas con el comando `matrix(data,nrow/ncol)`. Es importante recordar que debes elegir entre usar `nrow` o `ncol`, no puedes usar ambos en un mismo comando.

```r
# Usando la y del ejemplo anterior, generaremos una matriz de 5 filas y 2 columnas
(M <- matrix(y, nrow = 5))
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    5    4
## [3,]    2    8
## [4,]    6    5
## [5,]    3    9
```

También es posible hacer una multiplicación de matrices con R, por medio del comando `%*%`

```r
w <- matrix(1:10, nrow=2)
(v <- w%*%M) # una matriz de 2*5 por una de 5*2 que resulta en una de 2*2
```

```
##      [,1] [,2]
## [1,]   95  175
## [2,]  112  208
```

```r
(l <- M%*%w) # matriz de 5*2 por una de 2*5 que resulta en una de 5*5
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   15   31   47   63   79
## [2,]   13   31   49   67   85
## [3,]   18   38   58   78   98
## [4,]   16   38   60   82  104
## [5,]   21   45   69   93  117
```

Es posible convertir una matriz en base de datos:

```r
M <- as.data.frame(M)
print(M)
```

```
##   V1 V2
## 1  1  7
## 2  5  4
## 3  2  8
## 4  6  5
## 5  3  9
```

Algunos objetos (como tablas o matrices) pueden ser visualizados en su totalidad con dos métodos:

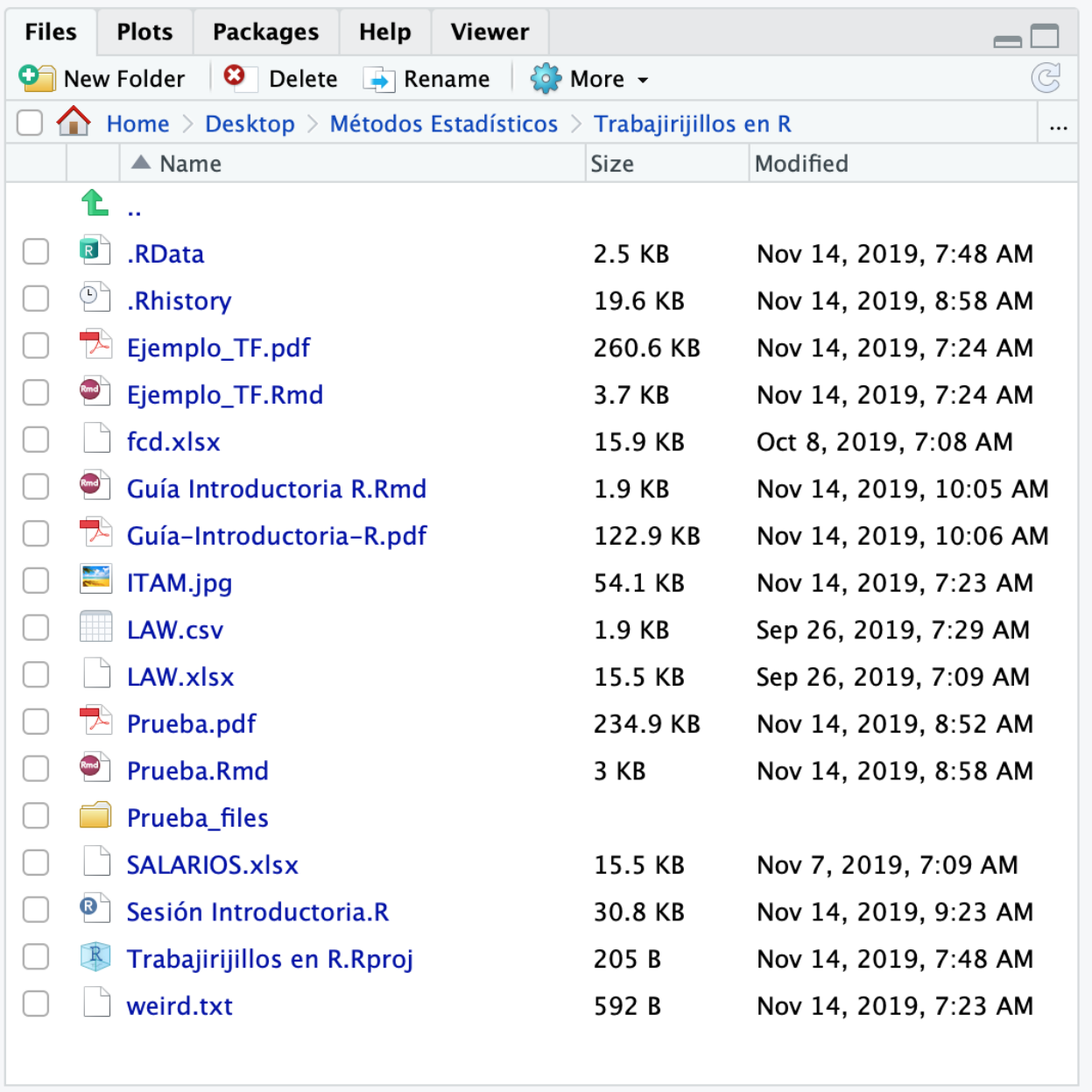
- Con el comando View(objeto), en donde "objeto" es el nombre que le has asignado.
- Dando doble click al objeto en el Ambiente de Trabajo

Dato curioso: puedes darle prácticamente cualquier nombre a un objeto. Las limitantes son que no puede haber acentos ni espacios. Aunque puede usarse un "." para reemplazar espacios, recomendamos usar guión bajo para esos casos. Ej: `x_2 <- 30`

¿Te equivocaste en el nombre de uno de tus objetos y ahora te estorba ese objeto extra? No te preocupes, puedes borrarlo con el comando `rm(objeto)`, donde objeto es el nombre que le asignaste. Si quieres borrar todos tus objetos, utiliza el comando `rm(list = ls())`, pero asegurate de en verdad querer borrar todo.

El Navegador

En esta sección de RStudio se encuentran los archivos de tu directorio de trabajo, los gráficos que generes, los paquetes que tienes instalados, la sección de ayuda de RStudio, y un visualizador de objetos interactivos (cosa que no cubriremos en esta guía).



El directorio de trabajo

El directorio de trabajo es la ubicación de tus archivos en la que R guarda lo que le pidas guardar (por ejemplo, si exportaras una tabla, aparecería en el directorio de trabajo). La forma de saber cuál es el directorio de trabajo que estás utilizando es con el comando:

```
# getwd()
## Al replicarlo en tu Script o Consola, no pongas el símbolo de gato
```

¿Por qué es importante conocer el directorio de trabajo? Además de que esto hace posible encontrar los archivos que generes, es mucho más fácil importar una tabla si la tienes guardada en tu directorio de trabajo.

Si deseas cambiar tu directorio de trabajo, es simplemente una cuestión de hacer lo siguiente:

```
wd <- "~/Desktop/Trabajos R"
# setwd(wd)
## Al replicarlo en tu codigo, debes poner setwd() sin el símbolo de gato.
```

Paso por paso, lo que esto ha hecho es guardar la ruta de la carpeta que quieres usar como tu directorio de trabajo (wd), y luego establecerla como el directorio de trabajo (setwd(wd)). No es necesario hacer esto en dos pasos, puedes poner la ruta directamente en setwd(), en lugar de guardarla en un objeto. Sin embargo, guardarla en un objeto tiene sus usos.

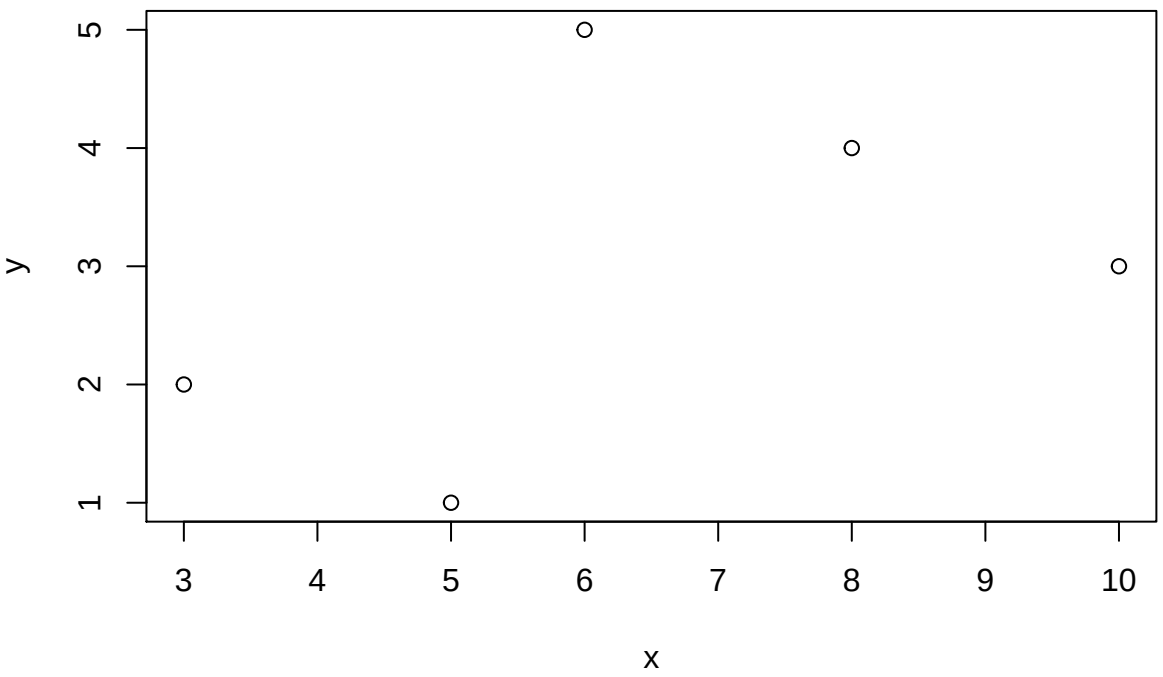
¡Ojo! Es muy importante que la ruta del directorio esté entrecomillada, independientemente de si la estás guardando en un objeto, o colocándola directamente en set(wd)

Gráficos

En cuestión de visualizaciones, R tiene un ámplio margen de posibilidades: puntos, lineas, histogramas, cajas, etc. Hay algunos paquetes que hacen estos gráficos más placenteros a la vista (entre ellos, ggplot2).

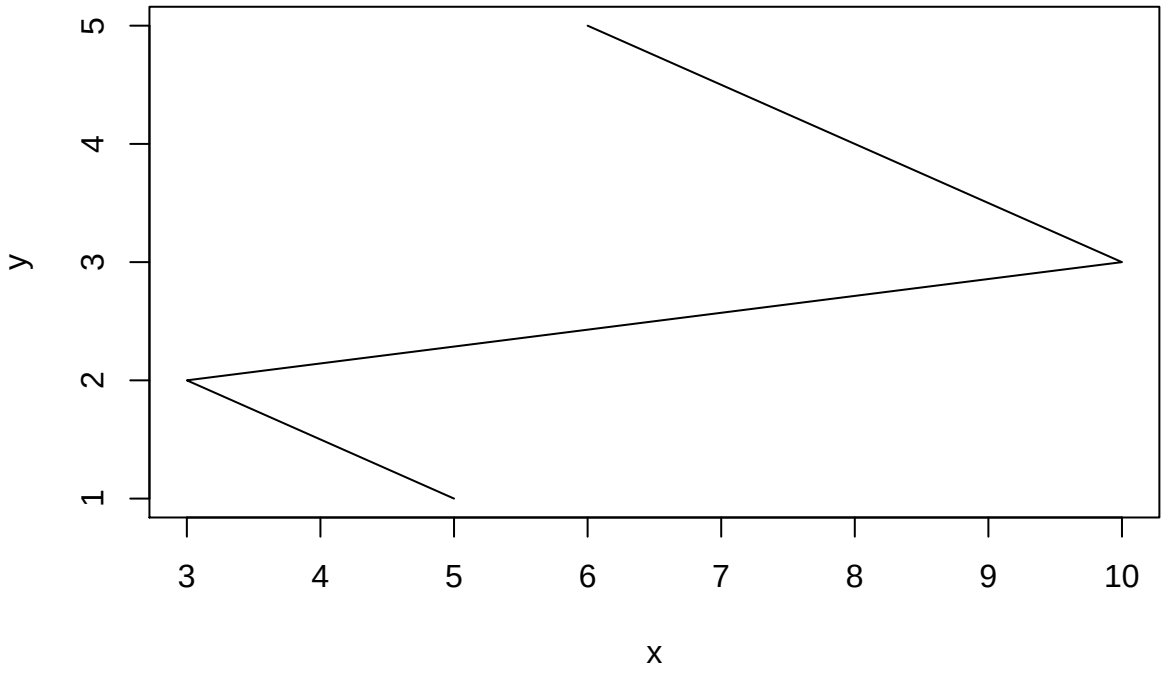
Un ejemplo rápido de gráfico es el siguiente:

```
x <- c(5,3,10,8,6)
y <- c(1,2,3,4,5)
plot(x,y)
```

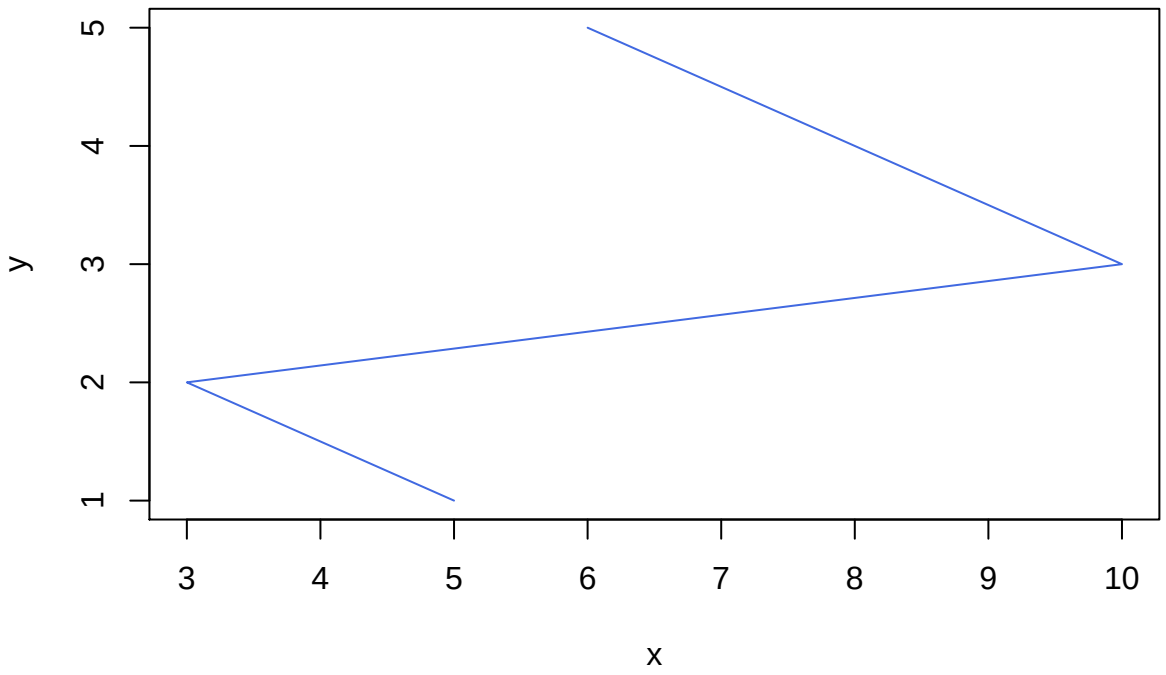


Otros gráficos que puedes hacer con los mismos datos:

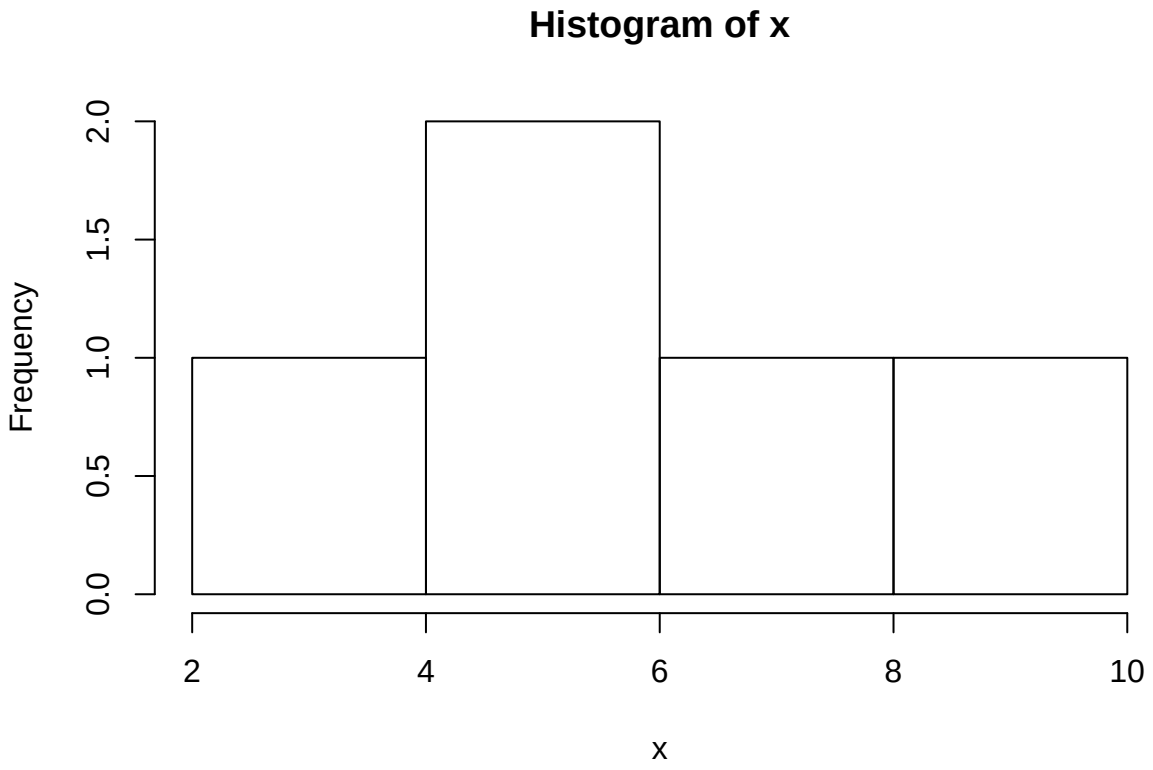
```r
# Lo siguiente es utilizando la x e y del ejemplo anterior
# Un gráfico de línea
plot(x,y, type = "l")
```



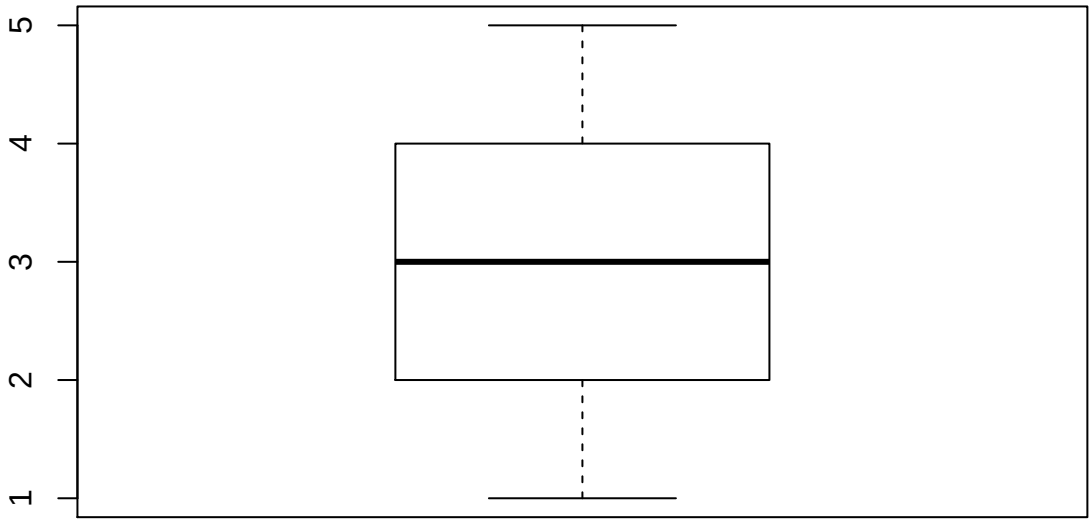

```r
## Si quisieras cambiar el color de la línea
plot(x,y, type = "l", col = "royalblue")
```



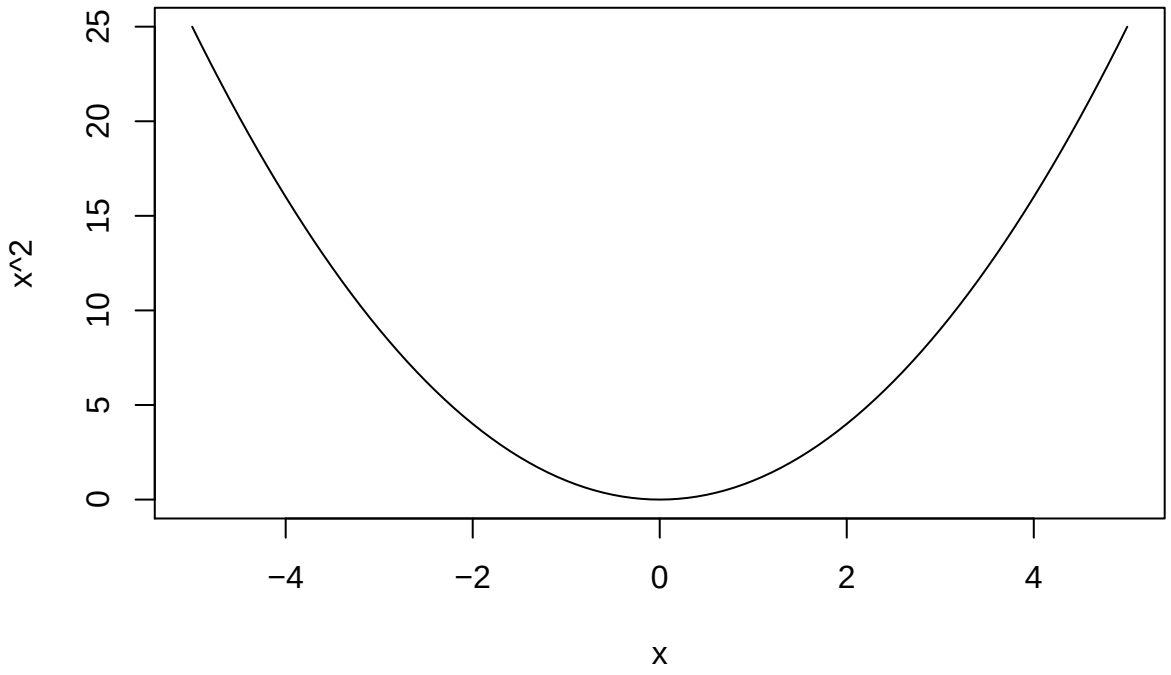

```r
# Un histograma
hist(x)
```




```r
# Un gráfico de caja y brazo
boxplot(y)
```




```r
# Un gráfico a partir de una función
# Nota: este ejemplo no utiliza la misma x de antes
curve(x^2,-5,5) # donde -5 es el límite inferior de x, y 5 el límite superior
```




```
# Es decir: curve(funcion(x), límite inferior de x, límite superior de x)
```

Esto, por supuesto, no es una lista exhaustiva de todo lo que se puede hacer con los gráficos en R (menos si nos metemos a ggplot2). Pero a modo introductorio, debe ser suficiente.

Paquetes

Los paquetes de R son herramientas sumamente poderosas, desde ayudarte a hacer mejores gráficas, hasta permitirte importar archivos de Excel a R, las posibilidades de lo que puedas encontrar son amplias. El único límite es la imaginación de la comunidad que colabora a mejorar R en línea.

Hay dos posibilidades para **instalar un paquete**:

- En la pestaña "Packages" en la sección del Navegador, oprimir el botón "Install" y escribir el nombre de los paquetes que deseas instalar.
  - Los nombres deben estar separados por una coma o un espacio.
- Utilizando el comando install.packages("nombre del paquete"). **Las comillas son de suma importancia para el comando**.
  - Si deseas instalar más de un paquete, debe hacerse a modo de vector: install.packages(c("paquete1","paquete2"))

Solo necesitarás instalar los paquetes una vez. Lo que si debes hacer cada vez que abras R es **activar los paquetes que deseas usar** mediante el comando library(paquete) o seleccionando la cajita correspondiente al paquete que deseas en la pestaña de "Packages" del Navegador.

```
library(tidyverse)
# Tidyverse es un "paquete de paquetes", incluye ggplot2 y dyplr, entre otros.
# Dependiendo del tamaño de un paquete, puede tomar un momento activarlo o instalarlo
# Espera a que concluya antes de darle más instrucciones a R
```

Nota: al usar el comando library(), no se deben usar comillas para el nombre del paquete

RStudio, y sus usuarios, han elaborado una serie de cheatsheets sobre los paquetes más utilizados. Puedes encontrar todos **aquí.**

Sección de Ayuda

Esta hecha para solicitar mayor información sobre los comandos de R, o de los paquetes que has activado. Hay dos modos de conseguir esta información:

- Escribiendo el comando en la sección de búsqueda dentro de la pestaña "Help" del Navegador.
- Usando el comando "?", lo cual te llevará automáticamente a la página de ayuda correspondiente. Por ejemplo:

```
?cor
```

El ejemplo anterior abriría la página de ayuda para la función de correlación en el Navegador.

Ejemplos de uso de R

Ya que hemos cubierto lo que es R, y lo que muestra cada pantalla de RStudio, proseguiremos a mostrar algunos ejemplos de lo que se puede hacer.

Muestreo Aleatorio Simple

Asumamos que tenemos una población cuya distribución es Normal:

```
N <- rnorm(50, mean = 0, sd = 1) # Esto genera una población de 50
# con distribución normal, media = 0 y desviación estándar = 1
# es decir, una normal estándar.

# para poder ver exactamente qué números fueron generados
print(N)
```

```
##  [1] -1.58584706  0.87793154 -1.15647903  0.84379070  0.27571547 -0.77210327
##  [7]  1.53746755  0.12588766  0.02082491  0.09420590 -1.16496495 -0.98170888
## [13] -0.70957286 -0.01812469 -1.18474833 -0.76795854 -0.34139161  1.18076939
## [19]  1.71155999  0.41021203  1.16912008  0.65497411  0.20833900 -0.02331820
## [25]  0.08149271  0.98478304 -0.47227384  1.18436271 -0.58237468 -0.88199199
## [31]  1.52803985  0.56859139 -1.28457419 -0.01461017 -0.87706897 -1.49267362
## [37] -1.82023597  0.35028499 -1.09268910  0.91515152  0.27878584  0.02014299
## [43] -0.34806497  0.59222600  0.53829607 -0.62177445  0.57219608 -0.75369623
## [49] -0.25910314  0.48234224
```

```
# Recuerda que si corres el código vas a tener números diferentes.
# Eso está bien, de lo contrario rnorm no estaría haciendo su trabajo adecuadamente.
```

De esta población, contamos con una muestra aleatoria de tamaño 30:

```
n <- sample(N, size = 30, replace = F)
# Este código extrae una muestra aleatoria, sin reemplazo, de 30 observaciones

print(n)
```

```
##  [1] -0.62177445 -0.76795854  0.56859139 -1.16496495 -1.18474833  0.27571547
##  [7]  0.02014299  0.87793154 -1.58584706 -0.01812469 -1.09268910 -0.70957286
## [13]  0.12588766 -0.25910314  0.48234224  0.57219608 -1.15647903  0.53829607
## [19]  1.18436271  1.53746755 -0.77210327  0.02082491 -0.87706897  0.20833900
## [25] -1.49267362  0.84379070 -0.01461017 -0.75369623  1.18076939  0.27878584
```

¿Cuál es la media de tu muestra? ¿Cuál es la desviación estándar? R puede calcular esto por tí:

```
# Para calcular la media y la desviación estándar hay un par de métodos.
# El primero sería ir paso por paso con las fórmulas que usamos para hacerlo a mano.
# El segundo sería usar los comandos pertinentes para obtener media y sd.
# El tercero es pedir a R un resumen, que incluye prácticamente todo.

# Método "paso a paso"
## Media
(sumatoria <- sum(n))
```

```
## [1] -3.755971
```

```
(x_barra <- sumatoria/30)
```

```
## [1] -0.125199
```

```
## Desviación Estándar
(sumatoria_2 <- sum((n-x_barra)^2))
```

```
## [1] 21.07635
```

```
(varianza <- sumatoria_2/(30-1))
```

```
## [1] 0.7267707
```

```
(sd <- sqrt(varianza))
```

```
## [1] 0.8525085
```

```
# Comandos de Media y Desviación Estandar
## Media
(x_barra <- mean(n))
```

```
## [1] -0.125199
```

```
## Desviación Estándar
(varianza <- var(n))
```

```
## [1] 0.7267707
```

```
(sd <- sqrt(varianza))
```

```
## [1] 0.8525085
```

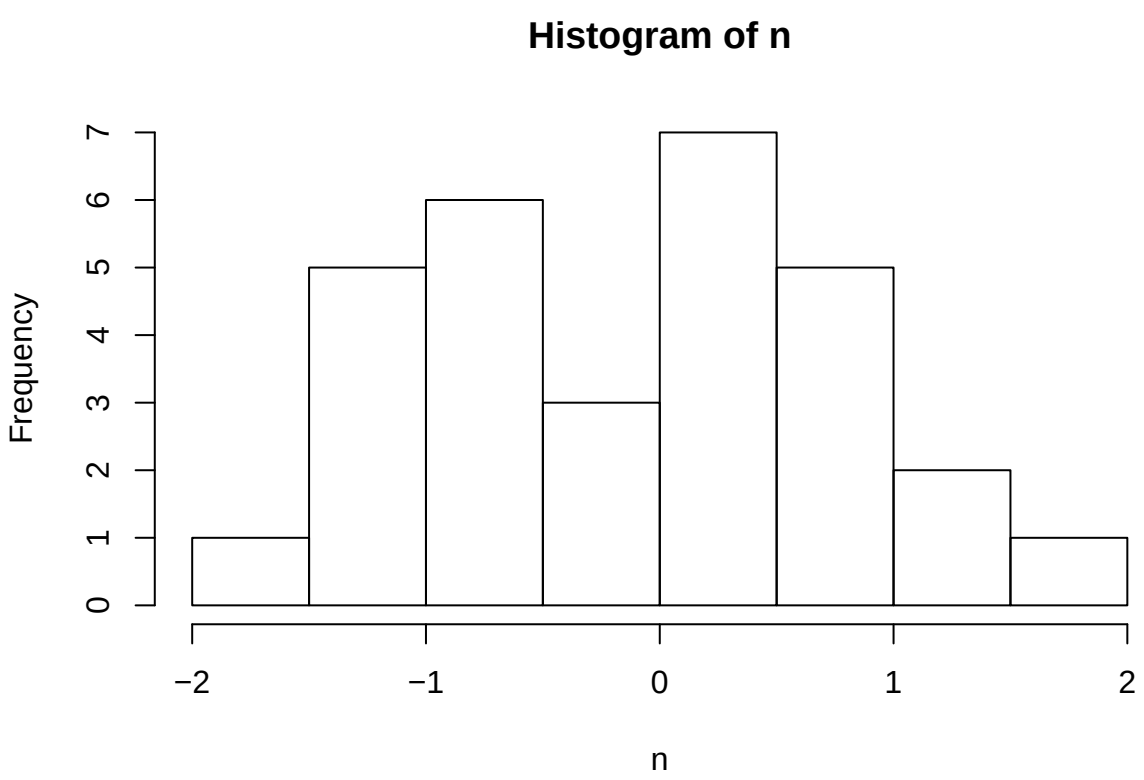
```
# Resumen
summary(n)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.585847 -0.771067  0.002766 -0.125199  0.524308  1.537467
```

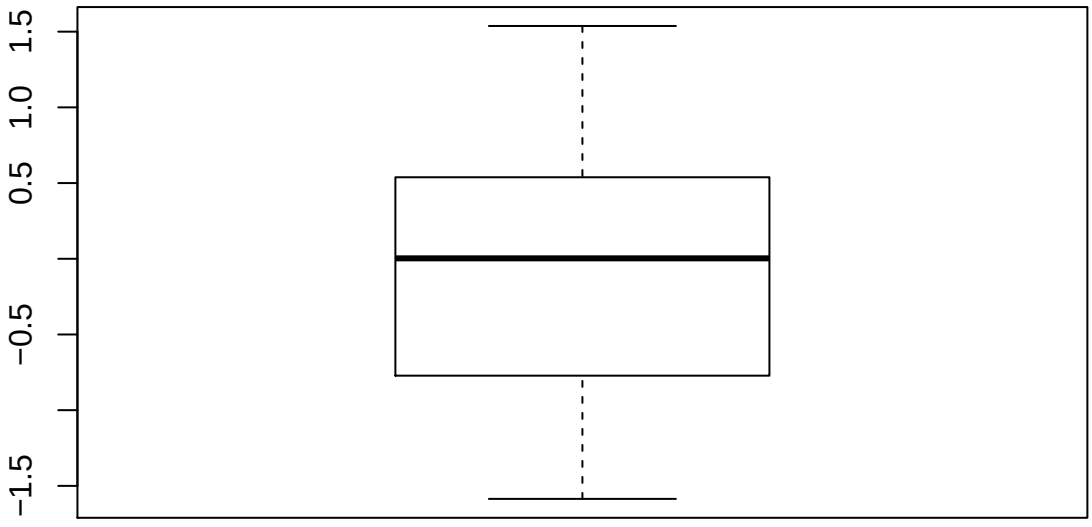
Como puedes ver, la función summary() es sumamente útil, ya que te da: valor mínimo, valor máximo, 1er cuartil, mediana, tercer cuartil y la media. En este caso no imprimió la desviación estándar, pero no es tanto inconveniente al tener los otros métodos.

También podemos hacer diversas gráficas a partir de esta muestra:

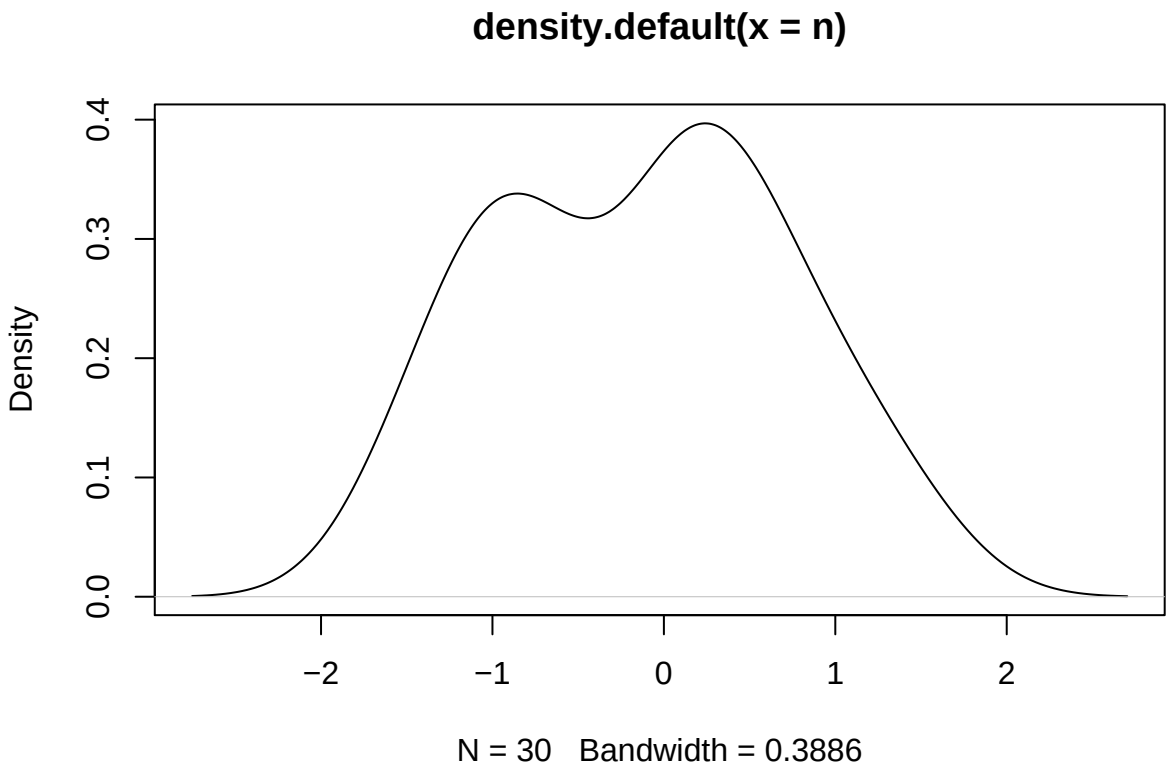
```r
hist(n)
```




```r
boxplot(n)
```




```r
plot(density(n)) # Debe ser idéntico al histograma, pero es más amigable a los ojos
```



Dado que en este caso conocemos la población y sabemos que se distribuye normal, sería redundante hacer intérvalos de confianza o pruebas de hipótesis para estimar el parámetro poblacional. Sabemos, de hecho, que la media poblacional es cero y que la varianza poblacional es uno.

Sin embargo, también sería interesante saber si nuestra muestra se encuentra entre los casos que estiman esto adecuadamente.

```
# recomendamos instalar el paquete "BSDA", que contiene varias
# pruebas de hipótesis
# para usar BSDA, es necesario instalar también "lattice"
library(BSDA)
z.test(n, alternative = "two.sided", mu = 0, sigma.x = 1, conf.level = .95)
```

```
##
##  One-sample z-Test
##
## data:  n
## z = -0.68574, p-value = 0.4929
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.4830379  0.2326398
## sample estimates:
## mean of x
## -0.125199
```

```
# si el p-value es mayor a .05, no podemos rechazar que la media poblacional
# sea cero
# Alternativamente, si 0 se encuentra en el intérvalo de confianza
# no podemos rechazar la hipótesis nula
```

```r
## otra forma de hacer la prueba (asumiendo que no conocieramos la varianza
## poblacional)

t.test(n, alternative = "two.sided", mu = 0, conf.level = .95)
```

```
##
##  One Sample t-test
##
## data:  n
## t = -0.80438, df = 29, p-value = 0.4277
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.4435309  0.1931329
## sample estimates:
## mean of x
## -0.125199
```

```r
# Los p-values son diferentes, y tiene mucho sentido, por la naturaleza
# de las pruebas (una usa parámetro y otra un estimador muestral)
# Por esto mismo, el intérvalo de confianza de la prueba t
# debe ser más grande
```

Regresiones

Agradecemos al profesor José Manuel Lecuanda Ontiveros por proveer los ejercicios y las **bases de datos** que utilizaremos en esta sección.

Para trabajar regresiones, recomendamos instalar los siguientes paquetes:

- stargazer: permite la creación de tablas de regresión
- corrplot: permite mejorar la visualización de matrices de correlación

Recuerda: si no tienes los paquetes instalados, puedes obtenerlos mediante el comando `install.packages(c("stargazer","corrplot"))`. Si ya los tienes instalados, no es necesario hacerlo otra vez.

Una vez que actives los comandos `library(stargazer)` y `library(corrplot)`. Nota: el comando library solo puede activar un paquete a la vez.

Ejemplo 1

Utilizando el archivo en esta **liga**, explica el valor agregado (VALUEADD) en trabajo (LABOR) y capital (CAPITAL).

Un método que podrías utilizar es escribir cada uno de los vectores, y unirlos mediante el comando `merge(x,y)`. Sin embargo, este es un método tardado, y que puede resultar sumamente problemático con bases de datos grandes. ¿Acaso hay una forma más sencilla de leer una base de datos?

R contiene el comando `read.csv()`, que permite la lectura de archivos tipo "comma-separated values", que pueden ser guardados como objetos. Todo lo que tienes que hacer es colocar la dirección del archivo:

```r
archivo <- "http://people.stern.nyu.edu/wgreene/Text/Edition7/TableF5-3.csv"
datos <- read.csv(archivo)
# alternativamente, puedes poner la dirección directamente en el comando
# no es necesario crear un objeto con la dirección.
print(datos)
```

```
##    OBS VALUEADD   LABOR  CAPITAL
## 1    1   657.29  162.31   279.99
## 2    2   935.93  214.43   542.50
## 3    3  1110.65  186.44   721.51
## 4    4  1200.89  245.83  1167.68
## 5    5  1052.68  211.40   811.77
## 6    6  3406.02  690.61  4558.02
## 7    7  2427.89  452.79  3069.91
## 8    8  4257.46  714.20  5585.01
## 9    9  1625.19  320.54  1618.75
## 10  10  1272.05  253.17  1562.08
## 11  11  1004.45  236.44   662.04
## 12  12   598.87  140.73   875.37
## 13  13   853.10  145.04  1696.98
## 14  14  1165.63  240.27  1078.79
## 15  15  1917.55  536.73  2109.34
## 16  16  9849.17 1564.83 13989.55
## 17  17  1088.27  214.62   884.24
## 18  18  8095.63 1083.10  9119.70
## 19  19  3175.39  521.74  5686.99
## 20  20  1653.38  304.85  1701.06
## 21  21  5159.31  835.69  5206.36
## 22  22  3378.40  284.00  3288.72
## 23  23   592.85  150.77   357.32
## 24  24  1601.98  259.91  2031.93
## 25  25  2065.85  497.60  2492.98
## 26  26  2293.87  275.20  1711.74
## 27  27   745.67  137.00   768.59
```

Ahora que ya tenemos la base de datos guardada como un objeto, podemos trabajar con cada uno de los datos. Para esto hay dos opciones:

- Utilizar `datos$NombreDeColumna` para decirle a R que estarás trabajando con una columna en específico del objeto "datos". Por ejemplo: `datos$CAPITAL`.
- Utilizar el comando `attach(NombreDeTabla)` para que R reconozca cada una de las columnas como un objeto en sí. Por ejemplo: `attach(datos)`

Es recomendable utilizar la segunda opción, ya que es menos laboriosa. Sin embargo, si por algún motivo tuvieras un objeto con el mismo nombre que una de las columnas (cosa que no recomendamos hacer), si tendrás que distinguirlos utilizando la primera opción.

```r
attach(datos)
# Ahora podemos jugar con cada uno de los datos. Por ejemplo:
mean(CAPITAL)
```

```
## [1] 2725.145
```

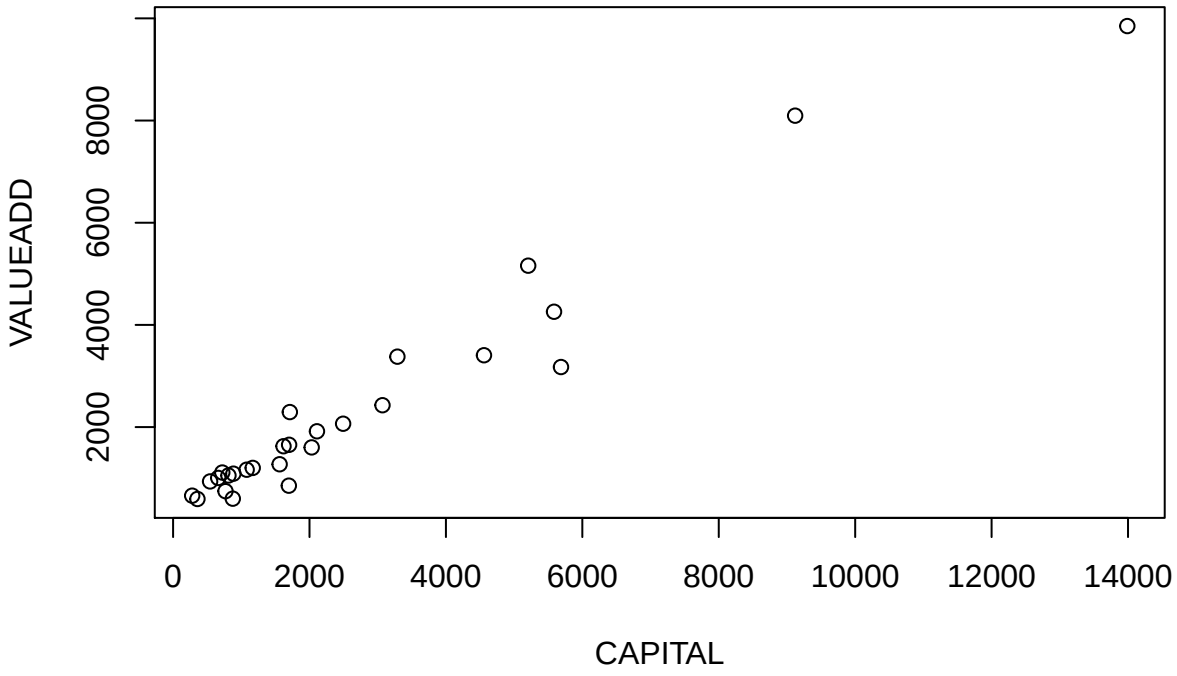
```r
# Que debe ser lo mismo que escribir:
mean(datos$CAPITAL)
```

```
## [1] 2725.145
```

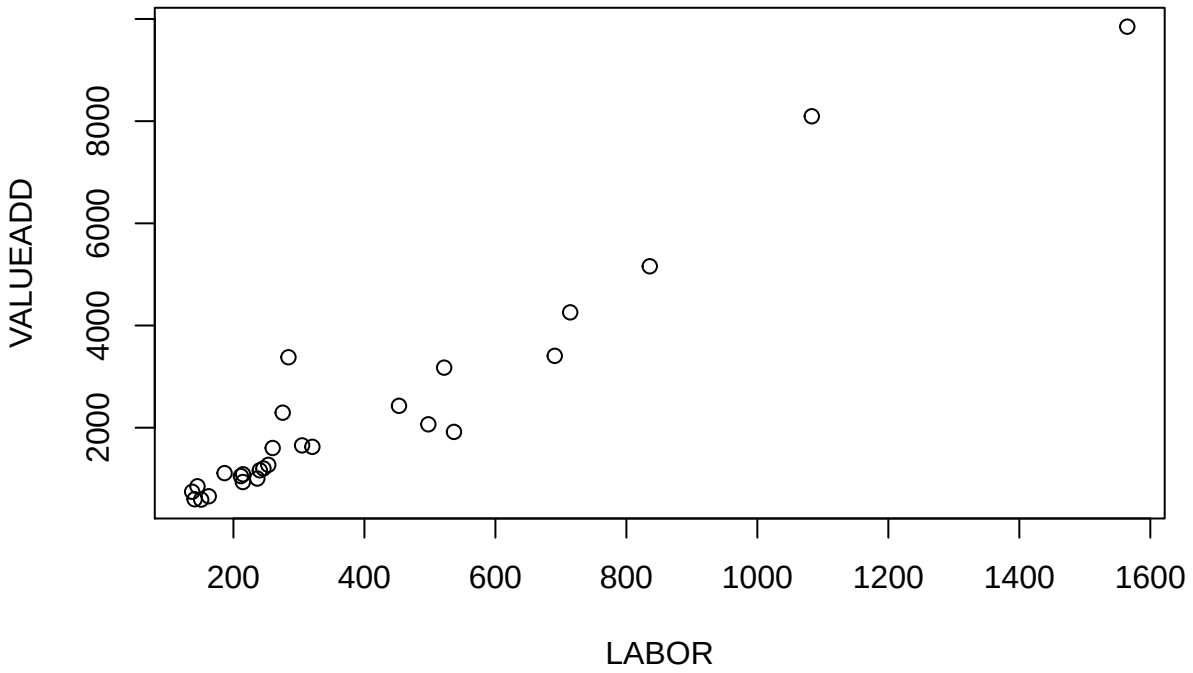
Para propósitos de este ejemplo, haremos dos regresiones: una bivariada, explicando VALUEADD solo con CAPITAL, y la otra multivariada, con CAPITAL y LABOR.

El primer paso es graficar los datos para ver con qué tipo de relaciones podríamos estar lidiando:

```r
plot(CAPITAL, VALUEADD, type = "p")
```




```r
plot(LABOR, VALUEADD, type = "p")
```



Viendo esto, ahora tenemos dos métodos de estimar una regresión bivariada: paso a paso y por medio de un comando.

Primero queremos estimar la regresión:

$VALUEADD_i = \hat{\alpha} + \hat{\beta} * CAPITAL_i + \hat{u}_i$

Paso a paso:

```
(betagorro <- cov(CAPITAL,VALUEADD)/var(CAPITAL)) # para estimar el coeficiente beta
```

```
## [1] 0.7149093
```

```
(alphagorro <- mean(VALUEADD) - betagorro*mean(CAPITAL)) # para estimar el intercepto alpha
```

```
## [1] 391.9691
```

```
ygorro <- alphagorro + betagorro*CAPITAL

datos$ygorro <- ygorro # añade una columna llamada ygorro al objeto "datos"

print(datos)
```

```
##    OBS VALUEADD   LABOR  CAPITAL     ygorro
## 1    1   657.29  162.31   279.99   592.1365
## 2    2   935.93  214.43   542.50   779.8074
## 3    3  1110.65  186.44   721.51   907.7833
## 4    4  1200.89  245.83  1167.68  1226.7544
## 5    5  1052.68  211.40   811.77   972.3110
## 6    6  3406.02  690.61  4558.02  3650.5400
## 7    7  2427.89  452.79  3069.91  2586.6763
## 8    8  4257.46  714.20  5585.01  4384.7447
```

```
## 9    9  1625.19  320.54  1618.75  1549.2285
## 10  10  1272.05  253.17  1562.08  1508.7146
## 11  11  1004.45  236.44   662.04   865.2676
## 12  12   598.87  140.73   875.37  1017.7792
## 13  13   853.10  145.04  1696.98  1605.1559
## 14  14  1165.63  240.27  1078.79  1163.2061
## 15  15  1917.55  536.73  2109.34  1899.9559
## 16  16  9849.17 1564.83 13989.55 10393.2286
## 17  17  1088.27  214.62   884.24  1024.1205
## 18  18  8095.63 1083.10  9119.70  6911.7275
## 19  19  3175.39  521.74  5686.99  4457.6512
## 20  20  1653.38  304.85  1701.06  1608.0727
## 21  21  5159.31  835.69  5206.36  4114.0443
## 22  22  3378.40  284.00  3288.72  2743.1056
## 23  23   592.85  150.77   357.32   647.4205
## 24  24  1601.98  259.91  2031.93  1844.6148
## 25  25  2065.85  497.60  2492.98  2174.2237
## 26  26  2293.87  275.20  1711.74  1615.7079
## 27  27   745.67  137.00   768.59   941.4412
```

```
ugorro <- VALUEADD - ygorro

datos$ugorro <- ugorro

print(datos)
```

```
##    OBS VALUEADD   LABOR  CAPITAL     ygorro      ugorro
## 1    1   657.29  162.31   279.99   592.1365    65.15345
## 2    2   935.93  214.43   542.50   779.8074   156.12261
## 3    3  1110.65  186.44   721.51   907.7833   202.86670
## 4    4  1200.89  245.83  1167.68  1226.7544   -25.86439
## 5    5  1052.68  211.40   811.77   972.3110    80.36898
## 6    6  3406.02  690.61  4558.02  3650.5400  -244.52000
## 7    7  2427.89  452.79  3069.91  2586.6763  -158.78631
## 8    8  4257.46  714.20  5585.01  4384.7447  -127.28471
## 9    9  1625.19  320.54  1618.75  1549.2285    75.96147
## 10  10  1272.05  253.17  1562.08  1508.7146  -236.66462
## 11  11  1004.45  236.44   662.04   865.2676   139.18235
## 12  12   598.87  140.73   875.37  1017.7792  -418.90925
## 13  13   853.10  145.04  1696.98  1605.1559  -752.05588
## 14  14  1165.63  240.27  1078.79  1163.2061     2.42390
## 15  15  1917.55  536.73  2109.34  1899.9559    17.59412
## 16  16  9849.17 1564.83 13989.55 10393.2286  -544.05856
## 17  17  1088.27  214.62   884.24  1024.1205    64.14951
## 18  18  8095.63 1083.10  9119.70  6911.7275  1183.90252
## 19  19  3175.39  521.74  5686.99  4457.6512 -1282.26116
## 20  20  1653.38  304.85  1701.06  1608.0727    45.30729
## 21  21  5159.31  835.69  5206.36  4114.0443  1045.26570
## 22  22  3378.40  284.00  3288.72  2743.1056   635.29438
## 23  23   592.85  150.77   357.32   647.4205   -54.57048
## 24  24  1601.98  259.91  2031.93  1844.6148  -242.63475
## 25  25  2065.85  497.60  2492.98  2174.2237  -108.37369
## 26  26  2293.87  275.20  1711.74  1615.7079   678.16206
## 27  27   745.67  137.00   768.59   941.4412  -195.77123
```

```r
(R2 <- var(ygorro)/var(VALUEADD))
```

```
## [1] 0.9513224
```

Una forma menos tardada de estimar todo esto es por medio del comando `lm(y~x, data = NombreDeTabla)`, ya que hace la regresión como tal por ti:

```r
mireg1 <- lm(VALUEADD~CAPITAL, data = datos)

# De aquí podemos rescatar muchos datos, puedes ver la totalidad de ellos con
names(mireg1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```r
# De estos, los que más nos interesan son "fitted.values" y "residuals"

ygorro <- mireg1$fitted.values # extrae la estimación del modelo

ugorro <- mireg1$residuals # extrae los residuales del modelo

# Podemos observar los resultados de la regresión pidiendo un resumen

summary(mireg1)
```

```
##
## Call:
## lm(formula = VALUEADD ~ CAPITAL, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1282.26  -216.22     2.42   109.78  1183.90
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 391.96909  131.43334   2.982   0.0063 **
## CAPITAL       0.71491    0.03234  22.104   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 506.6 on 25 degrees of freedom
## Multiple R-squared:  0.9513, Adjusted R-squared:  0.9494
## F-statistic: 488.6 on 1 and 25 DF,  p-value: < 2.2e-16
```

Al hacer la regresión bivariada, preferimos utilizar el comando, ya que ahorramos mucho tiempo.

$VALUEADD_i = \hat{\beta}_0 + \hat{\beta}_1 * CAPITAL_i + \hat{\beta}_2 * LABOR + \hat{u}_i$

```r
mireg2 <- lm(VALUEADD~CAPITAL + LABOR, data = datos)

summary(mireg2)
```

```
##
## Call:
## lm(formula = VALUEADD ~ CAPITAL + LABOR, data = datos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -837.66 -224.02  -38.26   58.64 1153.09
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 114.3376   173.4314   0.659 0.516001
## CAPITAL       0.4710     0.1124   4.189 0.000326 ***
## LABOR         2.3381     1.0390   2.250 0.033856 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 469.9 on 24 degrees of freedom
## Multiple R-squared:  0.9598, Adjusted R-squared:  0.9565
## F-statistic: 286.5 on 2 and 24 DF,  p-value: < 2.2e-16
```

¿Recuerdas que te recomendamos instalar stargazer? ¡Es para que puedas hacer esto!

```
stargazer(mireg1,mireg2, type = "text", title = "Ejemplo 2")
```

```
##
## Ejemplo 2
## ===============================================================
##                                 Dependent variable:
##                     ------------------------------------------
##                                       VALUEADD
##                                (1)                    (2)
## ---------------------------------------------------------------
## CAPITAL                     0.715***               0.471***
##                              (0.032)                (0.112)
##
## LABOR                                               2.338**
##                                                     (1.039)
##
## Constant                   391.969***               114.338
##                             (131.433)              (173.431)
##
## ---------------------------------------------------------------
## Observations                  27                     27
## R2                           0.951                  0.960
## Adjusted R2                  0.949                  0.956
## Residual Std. Error    506.622 (df = 25)      469.864 (df = 24)
## F Statistic         488.584*** (df = 1; 25) 286.541*** (df = 2; 24)
## ===============================================================
## Note:                                   *p<0.1; **p<0.05; ***p<0.01
```

Al meterlo a pdf, se puede visualizar más bonito, pero hemos incluido el comando que puedes utilizar para verlo directo en R. Para esto, se utiliza `type="latex"`.

Lo que hace esto es darte el código que debes poner en RMarkdown, pero veremos más sobre eso más adelante.

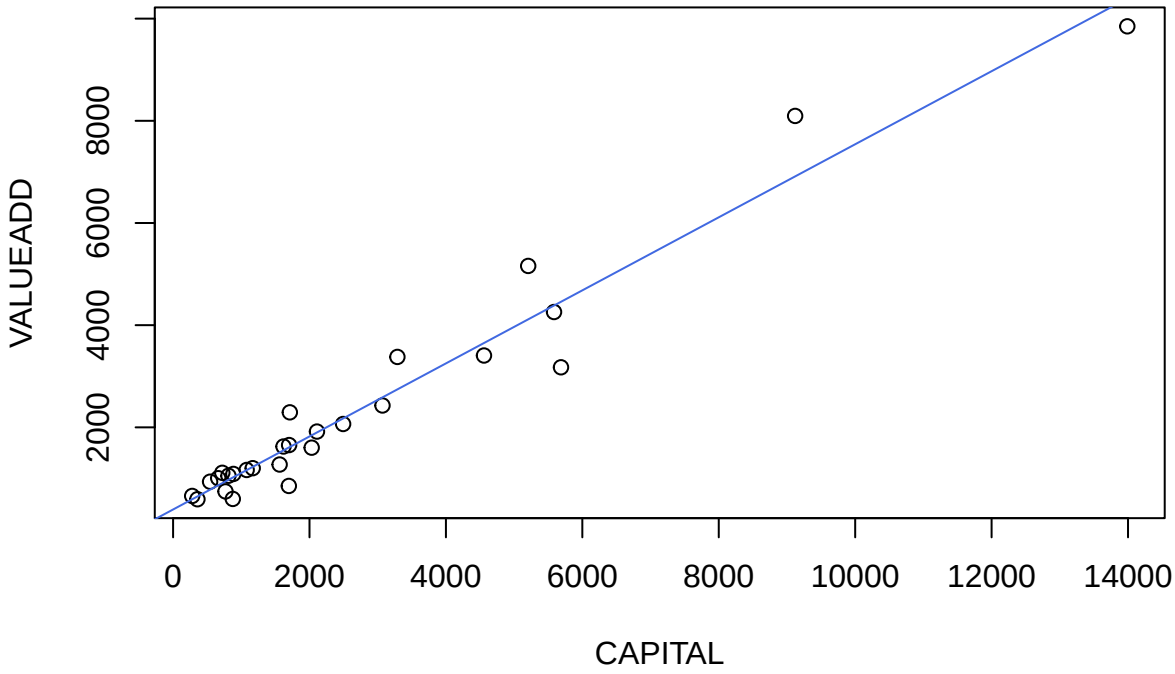
Table 1: Ejemplo 1

	Dependent variable:	
	VALUEADD	
	(1)	(2)
CAPITAL	0.715***	0.471***
	(0.032)	(0.112)
LABOR		2.338**
		(1.039)
Constant	391.969***	114.338
	(131.433)	(173.431)
Observations	27	27
R^2	0.951	0.960
Adjusted R^2	0.949	0.956
Residual Std. Error	506.622 (df = 25)	469.864 (df = 24)
F Statistic	488.584*** (df = 1; 25)	286.541*** (df = 2; 24)
Note:		*p<0.1; **p<0.05; ***p<0.01

Recuerda: si utilizas stargazer en un trabajo de investigación, debes utilizar la cita adecuada: Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables. R package version 5.2.2. https://CRAN.R-project.org/package=stargazer

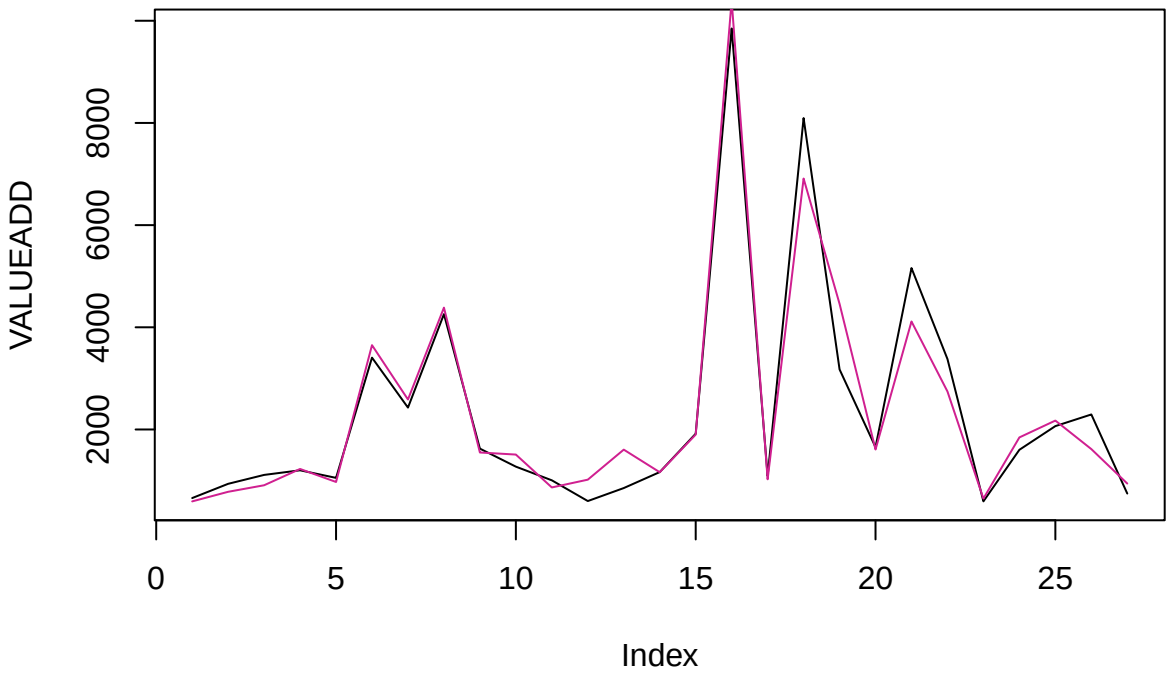
También podemos graficar nuestras regresiones:

```
plot(CAPITAL, VALUEADD, type = "p")
abline(mireg1, col = "royalblue")
```



Además podemos verificar qué tan bien se ajusta nuestro modelo a lo observado:

```
plot(VALUEADD, type = "l")
lines(ygorro, type = "l", col = "violetred") # este comando añade una línea
```




```
# a la última gráfica que hayas generado.
```

¿Hiciste cambios a la base de datos que te gustaría guardar? No es necesario que corras los comandos de R cada vez que quieras recuperar tu tabla. Con el comando `write.csv(NombreDeObjeto, "NombreDeArchivo.csv")` puedes guardar tu base de datos modificada en tu directorio de trabajo.

Ejemplo 2

Este segundo ejemplo lidia con regresiones con diferentes ajustes funcionales. Para replicarlo, es necesario descargar la base de datos LAW.xlsx del **Drive de Facultad Menor**. En este ejemplo, también importaremos una tabla directamente de Excel, sin tener que transformar el archivo xlsx a un csv, además de aprender a hacer gráficos más elaborados con ggplot2.

La base de datos de hoy está basada en el trabajo de La Porta et al (1988), en el que se intenta explicar el desarrollo económico en función del estado de derecho.

En el Drive hay dos versiones de LAW. Una csv y otra xlsx. La versión csv fue generada al guardar como csv desde Excel, para después utilizar el comando `read.csv()`.

Esta es una forma válida, pero un tanto más prolongada de lo que podría ser. ¿Qué tal si no tuvieras que abrir Excel en absoluto? El paquete "readxl" permite importar tablas hechas en Excel a R de forma directa, necesitando únicamente especificar la dirección del archivo y, en su caso, la Hoja que se quiere importar.

Recuerda instalar y activar el paquete antes de intentar usar sus comandos

Si has seguido nuestra sugerencia, guardaste la tabla LAW en tu directorio de trabajo. Esto simplifica mucho escribir la ruta al archivo.

```r
ruta <- "LAW.xlsx" # Nota, si no está en el directorio de trabajo
# debe anotarse como ~/.../Law.xlsx, donde "..." es la ruta hacia la carpeta
# en que hayas guardado el archivo. Puedes obtener la ruta dando click derecho
# sobre el archivo, y seleccionando "Obtener Información".
# Un ejemplo de una ruta: "~/Desktop/Trabajos R/LAW.xlsx"

LAW <- read_xlsx(ruta, "Hoja 1") # revisa la terminación de tu archivo, y verifica que
# si sea xlsx. Si la terminación es otra (por ejemplo, xls), la función
# también cambia (ejemplo: read_xls(ruta)), y la ruta debe ser hacia
# LAW.xls

print(LAW)
```

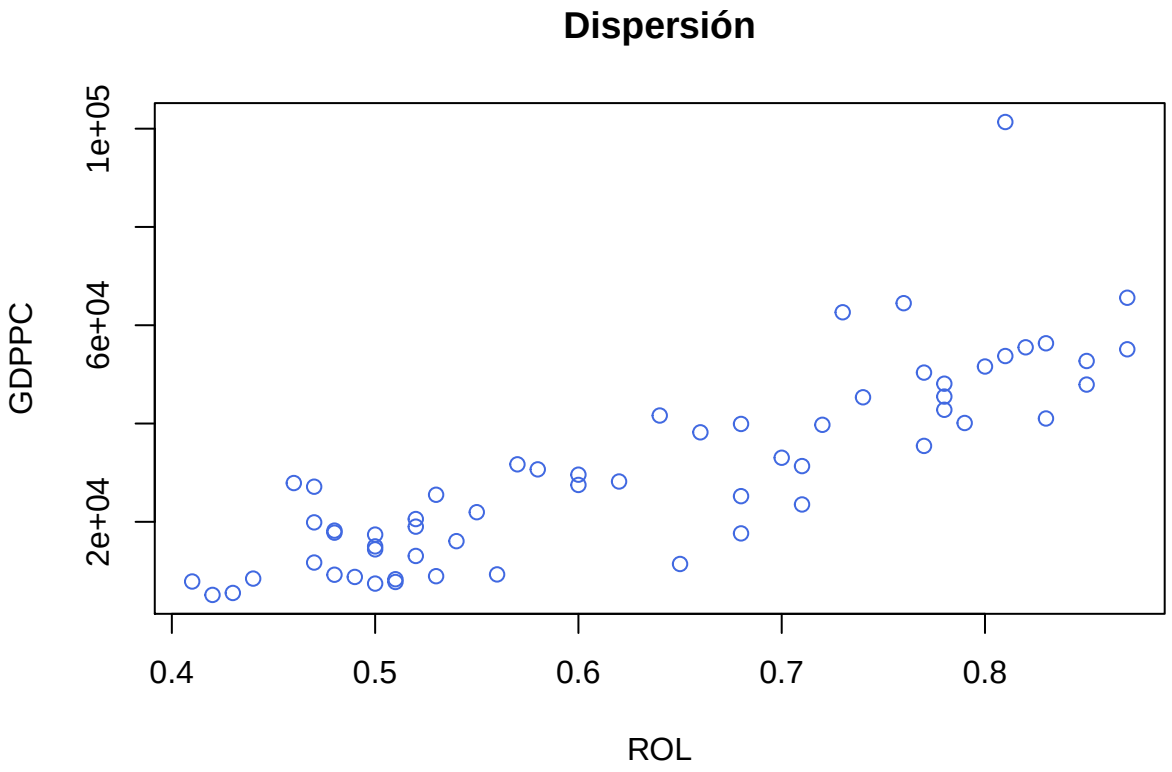
```
## # A tibble: 60 x 4
##    COD   COUNTRY              ROL  GDPPC
##    <chr> <chr>              <dbl>  <dbl>
##  1 BOL   Bolivia             0.41  7859.
##  2 HND   Honduras            0.42  5130.
##  3 NIC   Nicaragua           0.43  5524.
##  4 GTM   Guatemala           0.44  8447.
##  5 TUR   Turkey              0.46 27893.
##  6 ECU   Ecuador             0.47 11714.
##  7 MEX   Mexico              0.47 19888.
##  8 RUS   Russia              0.47 27147.
##  9 CHN   China               0.48 18210.
## 10 DOM   Dominican Republic  0.48 17799.
## # ... with 50 more rows
```

Como puedes ver, al pedirle a R que imprima la tabla, ya no aparecen todas las observaciones. Esto es normal, sobre todo con bases de datos grandes. Solo es cuestión de pedirle ver la tabla con `View(LAW)` o dando click sobre el objeto LAW en el ambiente de trabajo.

Las variables en la base de datos son: País (COUNTRY) e identificador de país (COD), el Rule of Law Index (ROL) de 2015, y su PIB per cápita, ajustado por paridad de poder adquisitivo (GDPPC). De aquí, queremos explicar GDPPC en ROL

Como siempre, verificaremos la posible relación entre nuestras variables por el método gráfico.

```r
attach(LAW)

plot(ROL, GDPPC, type = "p", main = "Dispersión", col = "royal blue")
```




```
# La función "main = título" le da título a las gráficas
```

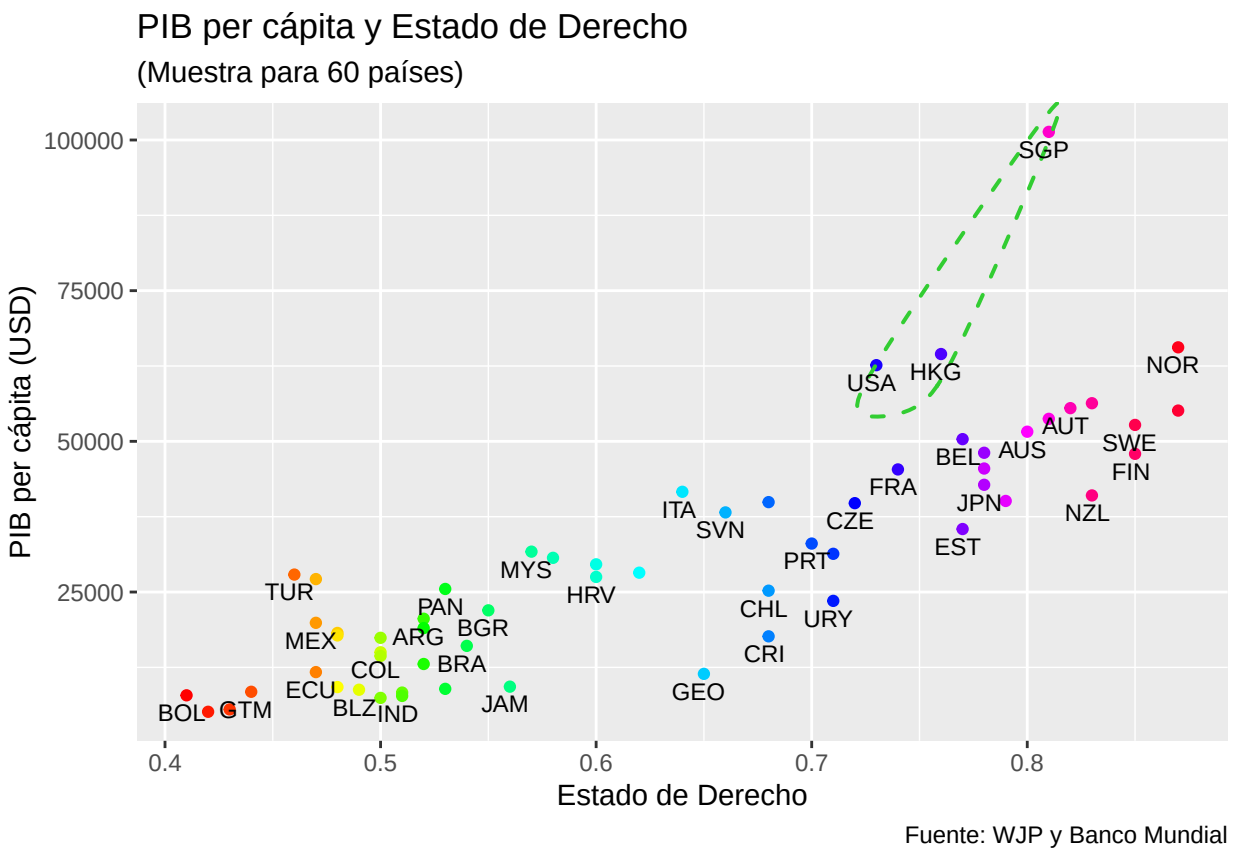
Por lo que vemos, parece que si hay una relación entre ambas variables. Pero podríamos incluso resaltar más cosas por medio de una gráfica. Esto lo lograremos con dos paquetes: "ggplot2" y "ggalt" (que es un complemento a ggplot2).

```
library(ggplot2)

library(ggalt)
```

```
## Registered S3 methods overwritten by 'ggalt':
##   method                  from
##   grid.draw.absoluteGrob  ggplot2
##   grobHeight.absoluteGrob ggplot2
##   grobWidth.absoluteGrob  ggplot2
##   grobX.absoluteGrob      ggplot2
##   grobY.absoluteGrob      ggplot2
```

```
selección <- LAW[LAW$ROL > 0.7 &
                   LAW$ROL < 0.85 &
                   LAW$GDPPC > 60000, ] # Esta es una herramienta que
# usaremos para resaltar los datos atípicos de la gráfica.

ggplot(data = LAW, aes(x = ROL, y = GDPPC)) +
  geom_point(color = rainbow(60)) +
  geom_encircle(aes(x = ROL, y = GDPPC),
                data = selección,
                color = "limegreen",
```

```
              linetype = "dashed",
              size = 2, expand = 0.05) +
geom_text(label = COD,
          size = 3,
          check_overlap = TRUE,
          vjust = 1.5,
          hjust = 0.6) +
labs(title = "PIB per cápita y Estado de Derecho" ,
     subtitle = "(Muestra para 60 países)",
     caption = "Fuente: WJP y Banco Mundial" ,
     x = "Estado de Derecho" ,
     y = "PIB per cápita (USD)" )
```



Bueno, eso fue mucho. Disequemos paso por paso qué es lo que hizo nuestro código.

ggplot define la base de datos, y las variables con las que estaríamos trabajando. Al añadir un signo + después del comando, le decimos a R que aún hay más instrucciones para esa gráfica. Si lo corrieramos así tal cual, R nos daría una gráfica en blanco.

ggpoint le dice a R que queremos hacer una gráfica de dispersión. Definimos dentro de esta función cualquier cosa en particular que queramos sobre los puntos. El color, en este caso, lo definimos con una escala de color.

geom_encircle le dice a R que queremos seleccionar algunos de los datos. Para esto debemos definir el aes (nuestra x e y) otra vez, además de definir qué datos queremos seleccionar. (Aquí es donde entra en juego el objeto **selección** que habíamos definido antes de la gráfica). Lo demás define particularidades estéticas de la línea del círculo.

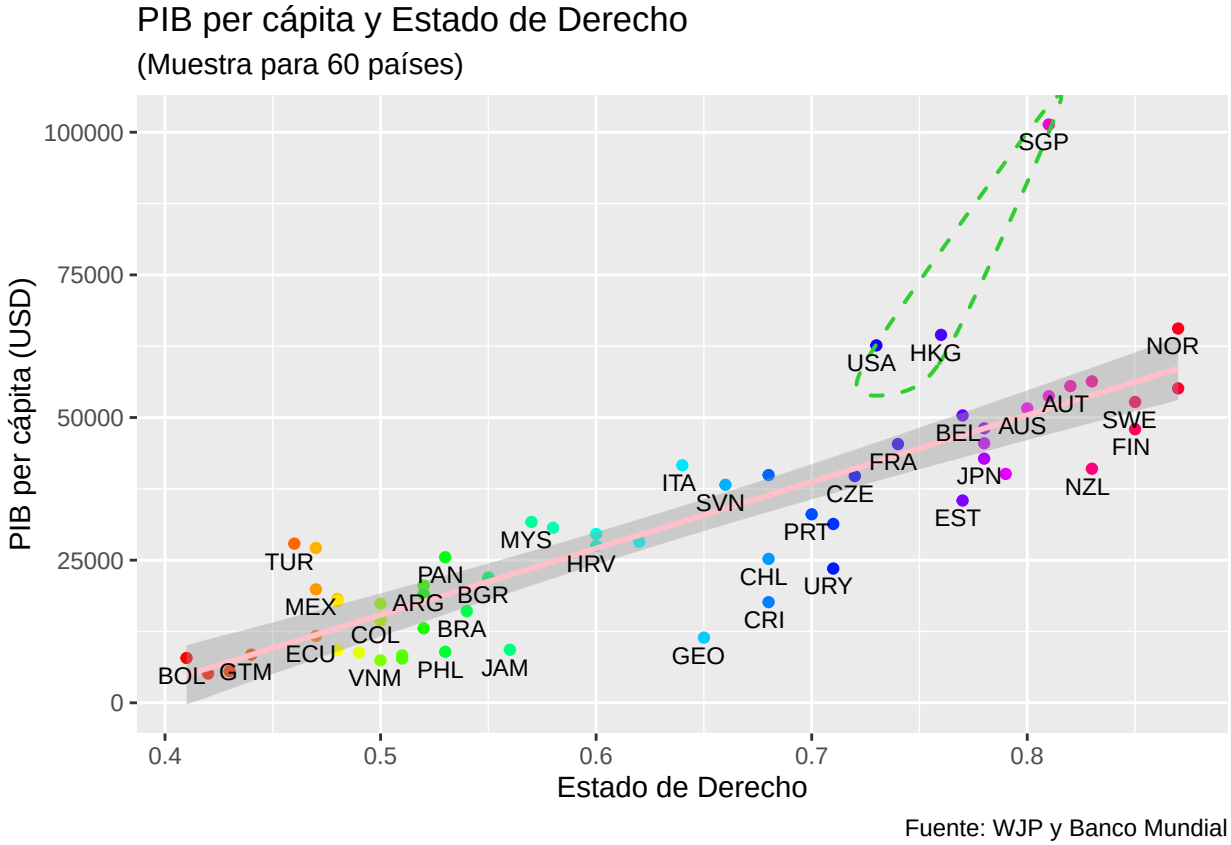
geom_text nos permite pedirle a R que le de nombre a cada punto. En este caso, le decimos que lo tome de la variable COD (el id de cada país) de la tabla. Además, ajustamos el texto para evitar que se encime

mucho.

labs se utiliza para definir título, subtítulo, nota, y el nombre de los ejes.

Ademas de esto, también podemos pedirle a ggplot que añada una recta de mejor ajuste con el comando **geom_smooth**.

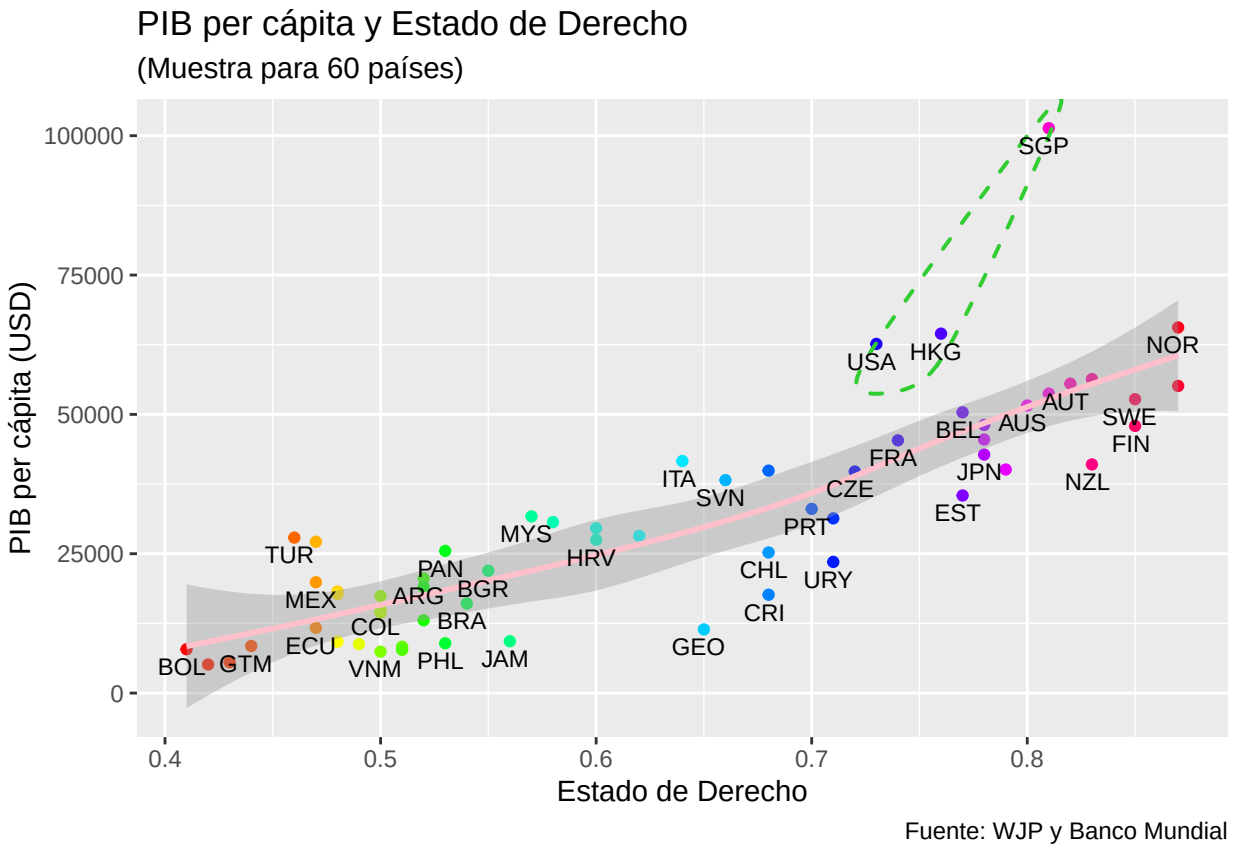
```
# Con ajuste lineal
ggplot(data = LAW, aes(x = ROL, y = GDPPC)) +
  geom_point(color = rainbow(60)) +
  geom_encircle(aes(x = ROL, y = GDPPC),
                data = selección,
                color = "limegreen",
                linetype = "dashed",
                size = 2,
                expand = 0.05) +
  geom_smooth(method = lm,
              color = "pink") +
  geom_text(label = COD,
            size = 3,
            check_overlap = TRUE,
            vjust = 1.5,
            hjust = 0.6) +
  labs(title = "PIB per cápita y Estado de Derecho" ,
       subtitle = "(Muestra para 60 países)",
       caption = "Fuente: WJP y Banco Mundial" ,
       x = "Estado de Derecho" ,
       y = "PIB per cápita (USD)" )
```



Fuente: WJP y Banco Mundial

```r
# Sin ajuste lineal
ggplot(data = LAW, aes(x = ROL, y = GDPPC)) +
  geom_point(color = rainbow(60)) +
  geom_encircle(aes(x = ROL, y = GDPPC),
                data = selección,
                color = "limegreen",
                linetype = "dashed",
                size = 2,
                expand = 0.05) +
  geom_smooth(color = "pink") +
  geom_text(label = COD,
            size = 3,
            check_overlap = TRUE,
            vjust = 1.5,
            hjust = 0.6) +
  labs(title = "PIB per cápita y Estado de Derecho" ,
       subtitle = "(Muestra para 60 países)",
       caption = "Fuente: WJP y Banco Mundial" ,
       x = "Estado de Derecho" ,
       y = "PIB per cápita (USD)" )
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Nota: no es necesario dar enter después de cada coma, lo hemos hecho solo para que no salga el código recortado en el pdf. Lo que si es importante es poner el signo + después de cerrar cada comando, si es que piensas poner un comando adicional.

Para propósitos de este ejemplo, eligiremos entre cuatro modelos:

- Modelo lineal
- Modelo log-lineal
- Modelo lineal-log
- Modelo doble log/Modelo log-log

Utilizaremos como criterio de selección el grado de correlación de las variables, aunque por lo general el ajuste logarítmico es preferible cuando hay mucha variabilidad en los datos.

```
##lineal
cor(ROL, GDPPC)
```

```
## [1] 0.839659
```

```
## log-lineal
cor(ROL, log(GDPPC))
```

```
## [1] 0.8417332
```

```
## lineal-log
cor(log(ROL), GDPPC)
```

```
## [1] 0.8292199
```

```
## log-log
cor(log(ROL), log(GDPPC))
```

```
## [1] 0.8467111
```

Dado el criterio que definimos, optaremos por un modelo log-log para la regresión, pero definiremos todos los modelos para propósitos de comparación en tabla.

```
m1 <- lm(GDPPC~ROL, data=LAW)
m2 <- lm(GDPPC~log(ROL), data=LAW)
m3 <- lm(log(GDPPC)~ROL, data=LAW)
m4 <- lm(log(GDPPC)~log(ROL), data=LAW)

# recuerda activar stargazer, si aún no lo has hecho.

# stargazer(m1,m2,m3,m4, type = "text", title = "Ejemplo 2")
## Al replicar esto, escribir el comando sin el símbolo de gato
```

Lo importante a recordar de este ejemplo es que, cuando se corren regresiones con ajuste lineal, se puede simple y sencillamente poner el ajuste dentro del comando `lm()`.

Table 2: Ejemplo 2

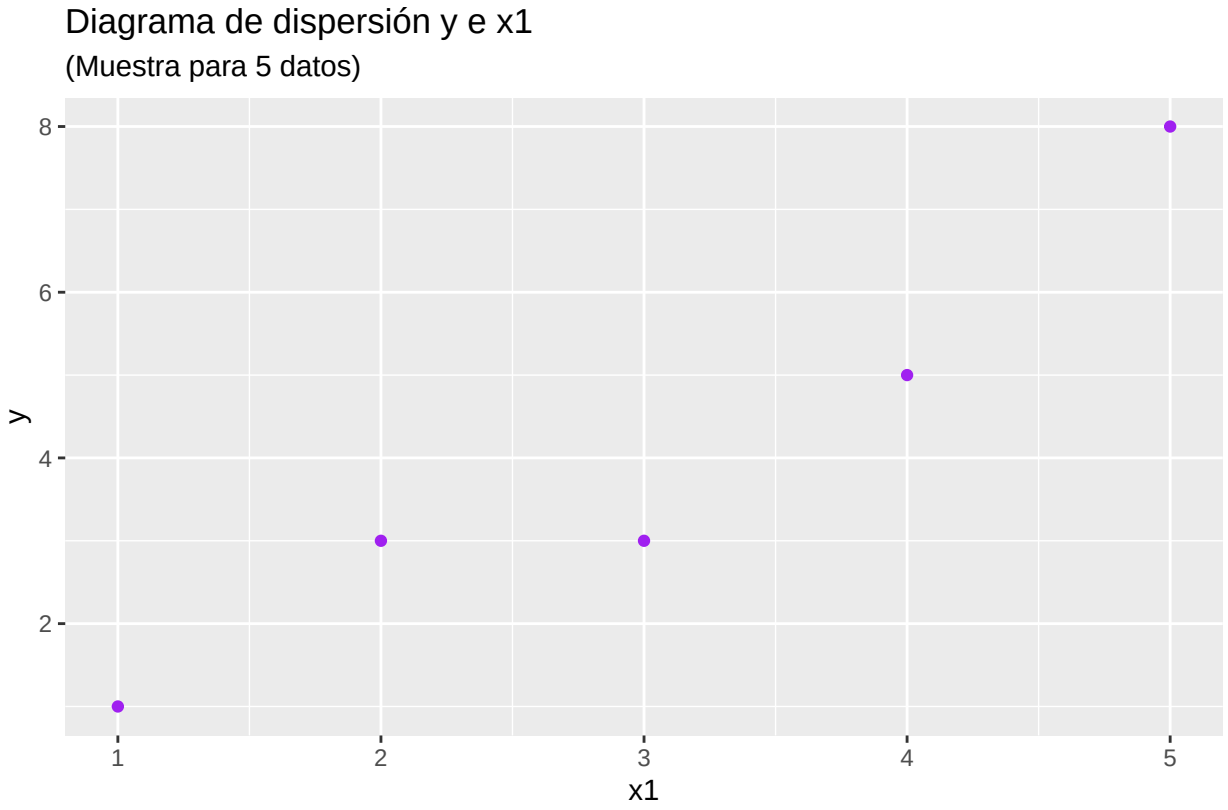
	Dependent variable:			
	GDPPC		log(GDPPC)	
	(1)	(2)	(3)	(4)
ROL	116,706.400***		4.320***	
	(9,912.168)		(0.364)	
log(ROL)		71,907.570***		2.711***
		(6,364.182)		(0.224)
Constant	−42,959.500***	65,577.150***	7.378***	11.420***
	(6,400.976)	(3,405.555)	(0.235)	(0.120)
Observations	60	60	60	60
R^2	0.705	0.688	0.709	0.717
Adjusted R^2	0.700	0.682	0.703	0.712
Residual Std. Error (df = 58)	10,718.370	11,030.350	0.393	0.388
F Statistic (df = 1; 58)	138.628***	127.663***	140.981***	146.889***
Note:				*p<0.1; **p<0.05; ***p<0.01

Ejemplo 3

En este ejemplo cubriremos la creación de matrices de correlación utilizando el paquete **corrplot**. Corrplot es un paquete que facilita la visualización de estas matrices, al añadir una escala de color, haciendo las correlaciones más fuertes oscuras, y las más débiles claras.

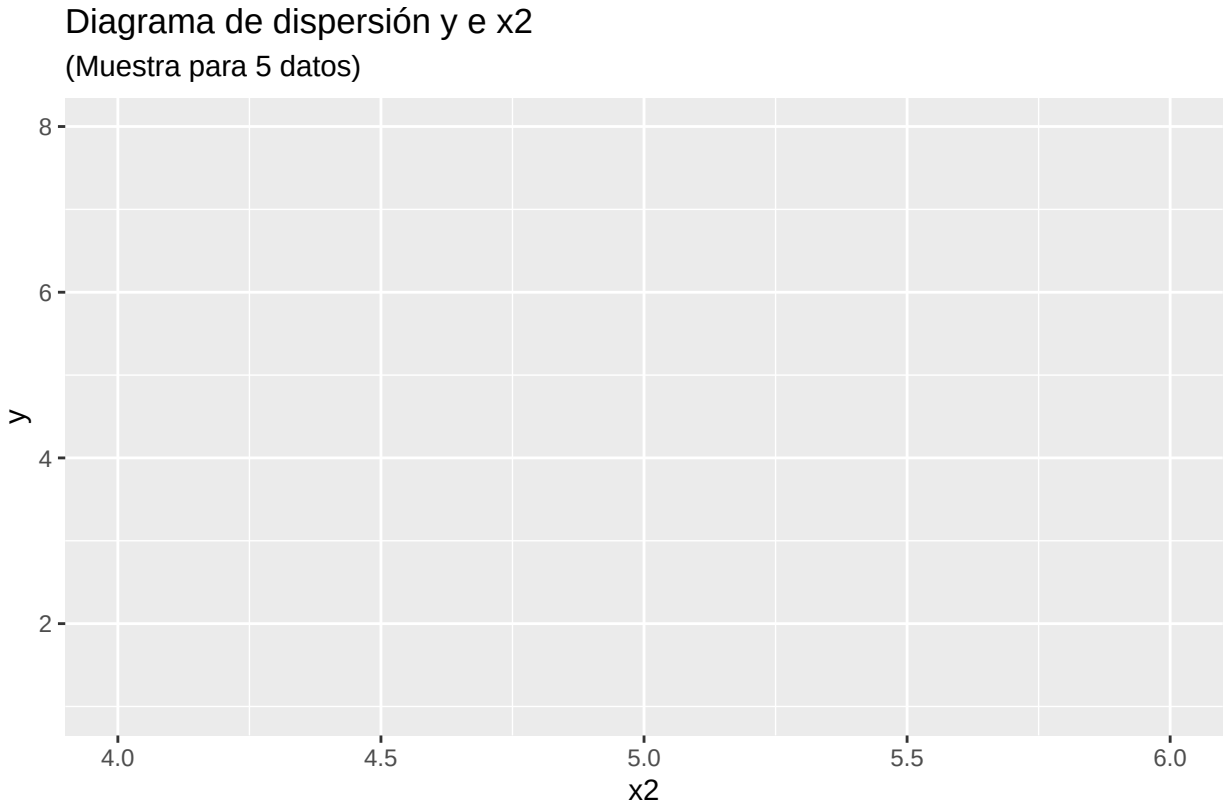
Recuerda instalar y activar el paquete corrplot para poder replicar este ejemplo.

```
library(corrplot)

y <- c(3,1,8,3,5)
x1 <- c(3,1,5,2,4)
x2 <- c(5,4,6,4,6)

datos <- data.frame(y,x1,x2) # Esto une los tres vectores para hacerlos tabla

## (y,x1)
ggplot(data = datos, aes(x = x1, y = y)) +
  geom_point(color = "purple") +
  labs(
    title = "Diagrama de dispersión y e x1",
    subtitle = "(Muestra para 5 datos)",
    caption = "Fuente: Elaboración propia",
    x = "x1",
    y = "y"
  )
```



Fuente: Elaboración propia

```
## (y,x2)
ggplot(data = datos, aes(x = x2, y = y)) +
  geom_point(color = "royal blue") +
  labs(
    title = "Diagrama de dispersión y e x2",
    subtitle = "(Muestra para 5 datos)",
    caption = "Fuente: Elaboración propia",
    x = "x2",
    y = "y"
  )
```

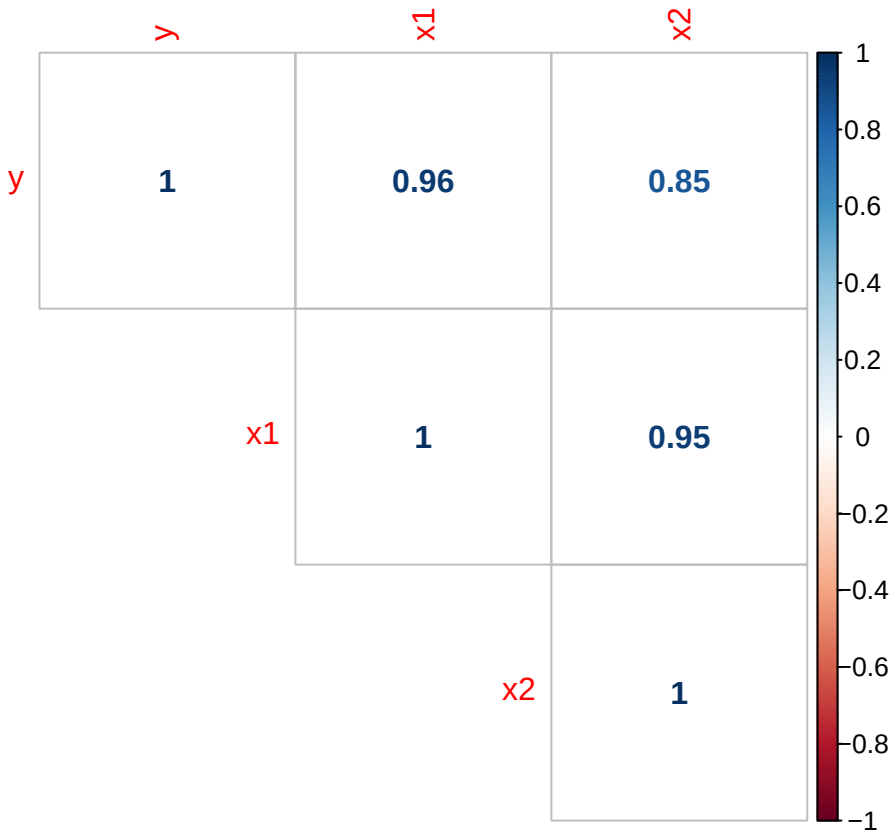


Fuente: Elaboración propia

```
M <- round(cor(datos), 4) # redondeamos a cuatro decimales las correlaciones

# Si quisieramos ver la matriz de correlación tal cual
print(M)
```

```
##         y     x1     x2
## y  1.0000 0.9562 0.8504
## x1 0.9562 1.0000 0.9487
## x2 0.8504 0.9487 1.0000
```

```
# Si queremos arreglarlo para que se vea mejor:

corrplot(M, method="number", type="upper")
```




```
regmul1 <- lm(y~x1+x2, data = datos)
summary(regmul1)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = datos)
##
## Residuals:
##          1          2          3          4          5
## -1.000e+00  5.000e-01  5.000e-01  1.110e-16  3.331e-16
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.000      4.475   0.894    0.466
## x1             2.500      0.866   2.887    0.102
## x2            -1.500      1.369  -1.095    0.388
##
## Residual standard error: 0.866 on 2 degrees of freedom
## Multiple R-squared:  0.9464, Adjusted R-squared:  0.8929
## F-statistic: 17.67 on 2 and 2 DF,  p-value: 0.05357
```

```
# stargazer(regmul1, type ="text", title = "Ejemplo 3")
```

Hay muchas opciones para manejar corrplot, puedes encontrarlas **aquí**.

Table 3: Ejemplo 3

	Dependent variable:
	y
x1	2.500
	(0.866)
x2	−1.500
	(1.369)
Constant	4.000
	(4.475)
Observations	5
R^2	0.946
Adjusted R^2	0.893
Residual Std. Error	0.866 (df = 2)
F Statistic	17.667* (df = 2; 2)
Note:	*p<0.1; **p<0.05; ***p<0.01

```
# ¡Ojo! Aquí hay un problema de multicolinealidad que se puede solucionar con
# componentes principales, por medio de la fórmula prcomp()

pca <- prcomp(datos, scale = TRUE)
# scale lo centra alrededor de 0
names(pca) # para ver qué datos se pueden rescatar
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

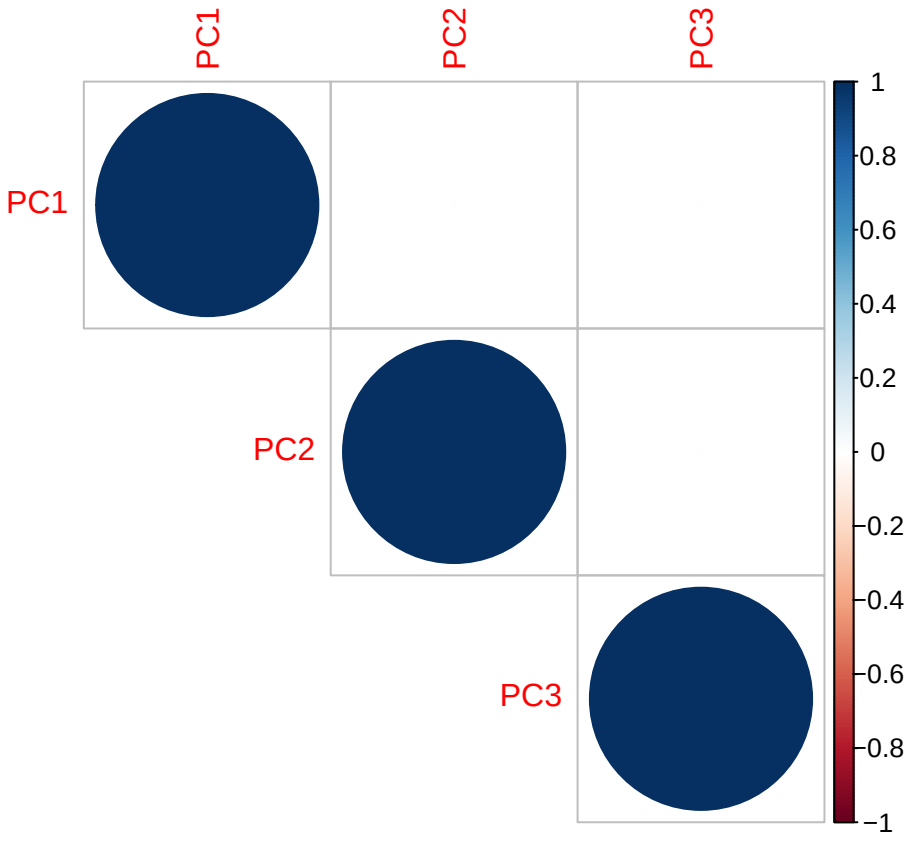
```
# Disponemos de desviación estandar (sdev), la media (center), x, y rotación

# La que más nos interesan son x y rotación
# Que son los valores ponderados y la matriz de componentes principales

pca$rotation
```

```
##          PC1         PC2       PC3
## y  0.5711394  0.69304955  0.439866
## x1 0.5911690  0.02448521 -0.806176
## x2 0.5694901 -0.72047400  0.395725
```

```
pca$x
```

```
##             PC1         PC2         PC3
## [1,] -0.2158704 -0.26194811 -0.16625371
## [2,] -1.9648775 -0.09634193  0.12525480
## [3,]  2.1807479  0.35829004  0.04099892
## [4,] -1.1592486  0.44304009 -0.05210822
## [5,]  1.1592486 -0.44304009  0.05210822
```

```r
# Para mostrar que soluciona el problema de multicolinealidad

M <- round(cor(pca$x),4)

corrplot(M, type = "upper") # La nueva matriz de correlación debe mostrar ceros.
```




```r
summary(pca) # nos da desviación estándar, proporción de varianza que explican y la proporción acumulad
```

```
## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     1.6845 0.3869 0.1123
## Proportion of Variance 0.9459 0.0499 0.0042
## Cumulative Proportion  0.9459 0.9958 1.0000
```

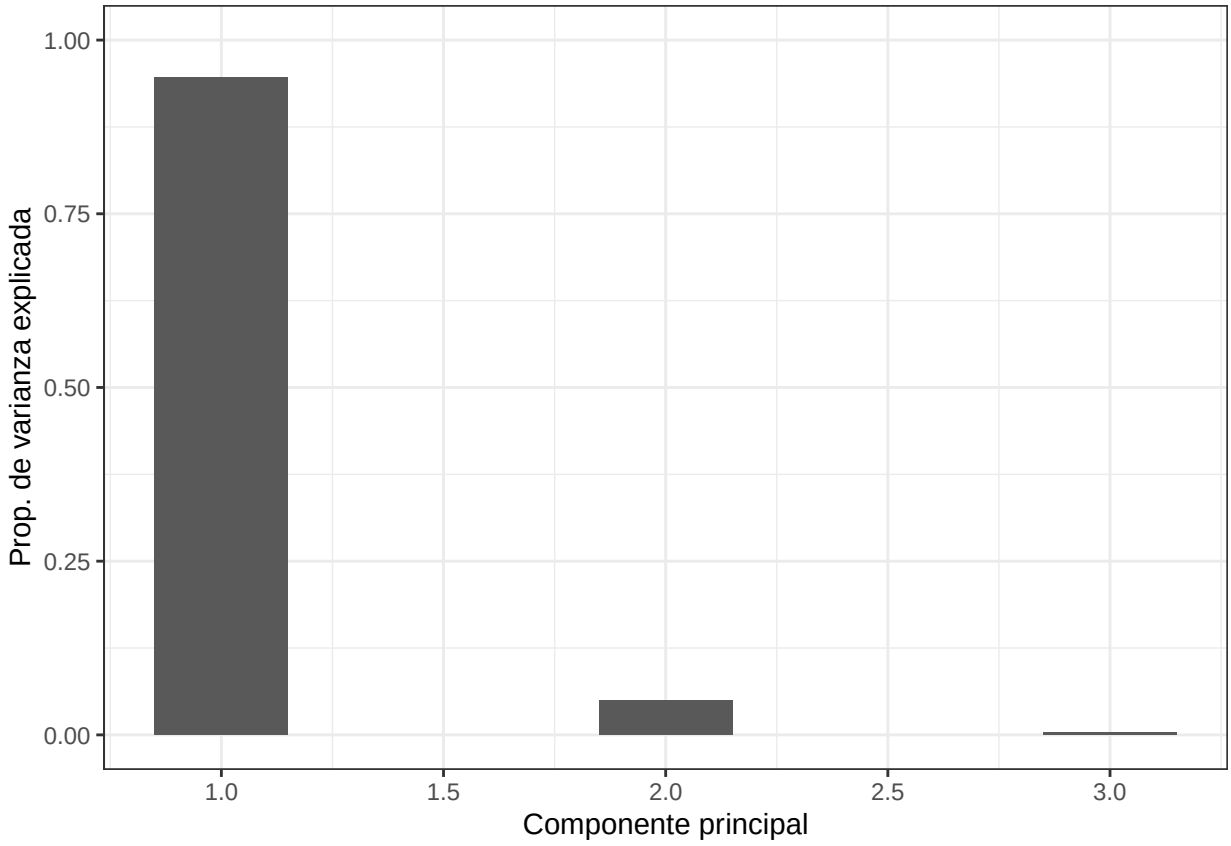
```r
# Para graficar el porcentaje de varianza explicada por componente principal
pca$sdev^2 # nos da la varianza
```

```
## [1] 2.83768003 0.14970984 0.01261013
```

```r
prop_varianza <- pca$sdev^2 / sum(pca$sdev^2)
prop_varianza
```

```
## [1] 0.945893343 0.049903281 0.004203376
```

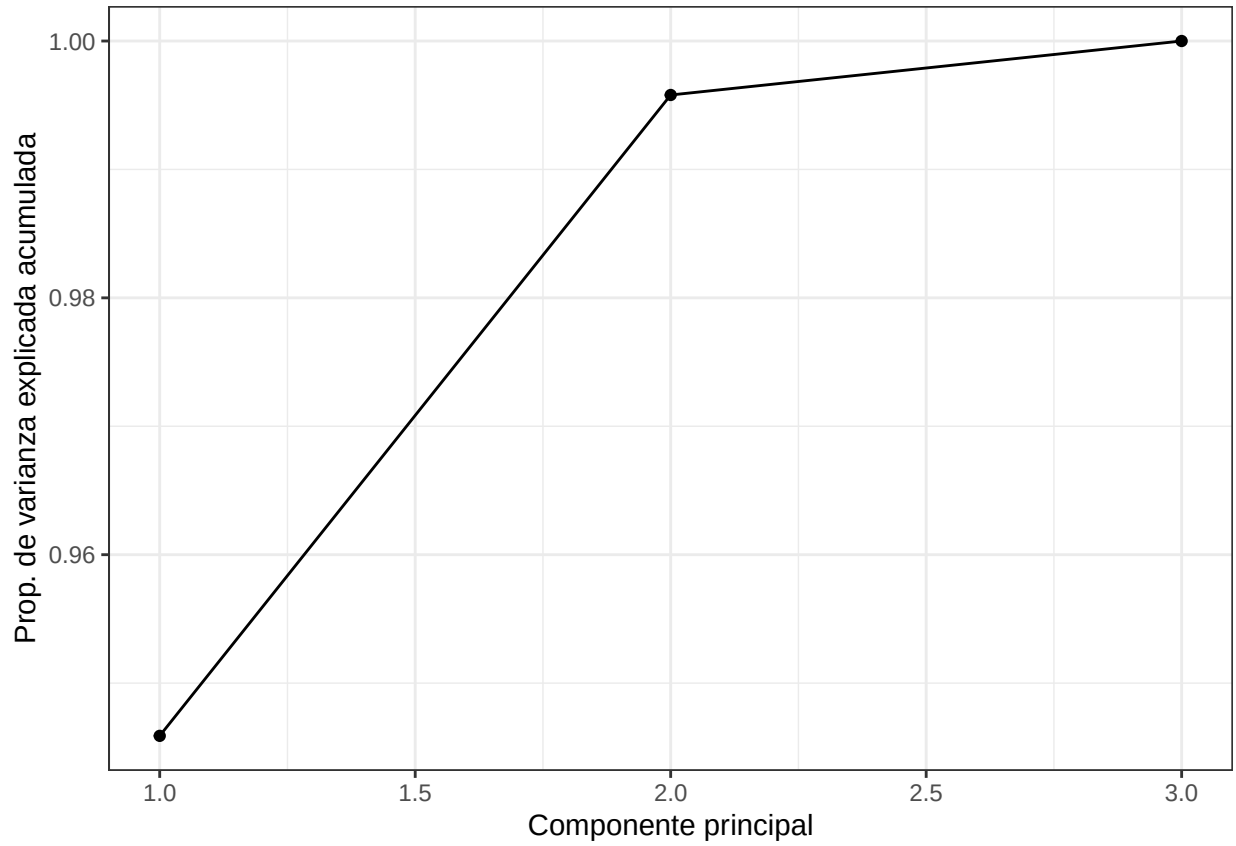
```r
ggplot(data = data.frame(prop_varianza, pc = 1:3),
       aes(x = pc, y = prop_varianza)) +
  geom_col(width = .3) +
  scale_y_continuous(limits = c(0,1)) +
  theme_bw() +
  labs(x = "Componente principal",
       y = "Prop. de varianza explicada")
```



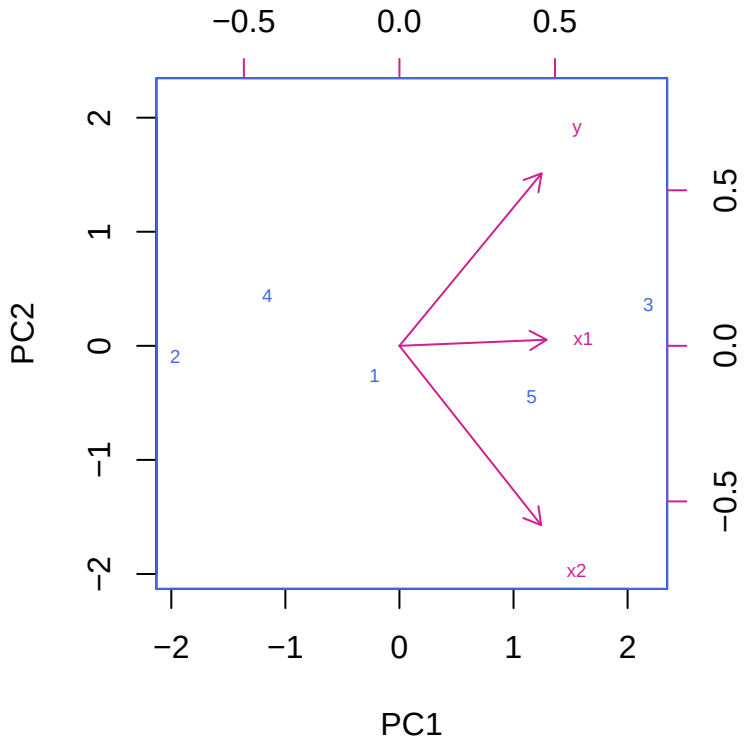

```r
# Para proporción de varianza acumulada

prop_varianza_acum <- cumsum(prop_varianza) # suma acumulada
prop_varianza_acum
```

```
## [1] 0.9458933 0.9957966 1.0000000
```

```r
ggplot(data = data.frame(prop_varianza_acum, pc = 1:3),
       aes(x = pc, y = prop_varianza_acum, group = 1)) +
  geom_point() +
  geom_line() +
  theme_bw() +
  labs(x = "Componente principal",
       y = "Prop. de varianza explicada acumulada")
```




```r
# Para ver el gráfico bidimensional entre PC1 y PC2
biplot(x = pca, scale = 0, cex = 0.6, col = c("royalblue", "violetred"))
```



```r
# Nos indica los nuevos valores centrados en 0 de todas las variables
# Las flechas nos indican si las variables se mueven en el mismo sentido, permite visualizar
# componentes principales

PC <- pca$x
PC <- data.frame(PC)

datos$PC1 <- PC$PC1

mireg2 <- lm(y~PC1, datos)
stargazer(mireg2, type = "text", title = "Ejemplo 3.1")
```

```
##
## Ejemplo 3.1
## ================================================
##                          Dependent variable:
##                      ---------------------------
##                                   y
## ------------------------------------------------
## PC1                            1.511***
##                                 (0.247)
##
## Constant                       4.000***
##                                 (0.373)
##
## ------------------------------------------------
## Observations                      5
## R2                               0.926
## Adjusted R2                      0.901
## Residual Std. Error         0.833 (df = 3)
## F Statistic              37.351*** (df = 1; 3)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

Ejemplo 4

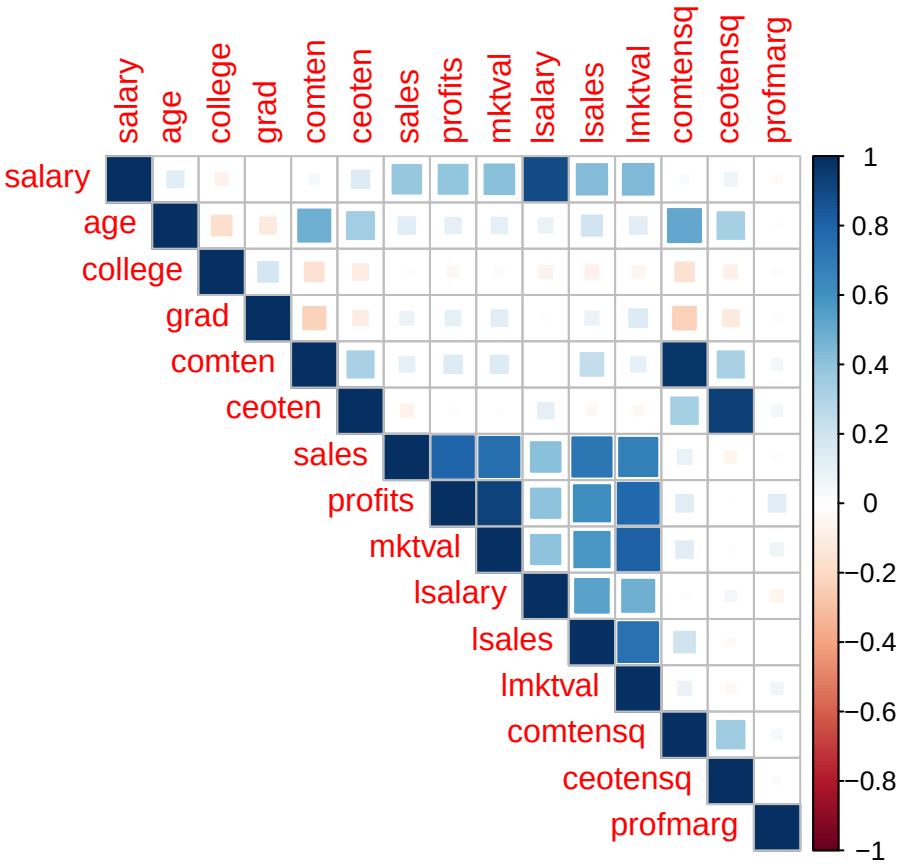
Para este ejemplo, utilizaremos otra base de datos en línea. Los detalles de cada variable pueden ser consultados **aquí**. El objetivo de este ejemplo es mostrar cómo hacer pruebas de significancia conjunta en R.

Dado que los datos están en formato de stata, necesitamos el paquete "foreign", para poder trabajar con ellos en R. De no tenerlo instalado, recuerda hacerlo antes de intentar activarlo.

```r
library(foreign)

ruta <- "http://fmwww.bc.edu/ec-p/data/wooldridge/ceosal2.dta"
base <- read.dta(ruta)

attach(base)
```

Ahora que tenemos nuestra base de datos a la mano, repetiremos los pasos de ejemplos anteriores con una diferencia: no le pediremos a stargazer que nos de la significancia conjunta.

```r
library(corrplot)

M <- round(cor(base), 2)

corrplot(M, method = "square", type = "upper")
```




```r
# Por lo que vemos, preferimos explicar el logaritmo de salario
# a partir de logartimo de valor de mercado y logaritmo de ventas

m1 <- lm(lsalary~lsales, data = base)
m2 <- lm(lsalary~lsales+lmktval, data = base)
m3 <- lm(lsalary~lsales+lmktval+profits, data = base)
m4 <- lm(lsalary~lsales+lmktval+profits+ceoten, data = base)
m5 <- lm(lsalary~lsales+lmktval+profits+ceoten+age, data = base)

library(stargazer)

stargazer(list(m1,m2,m3,m4,m5), type = "text", keep.stat = c("rsq", "adj.rsq"), title = "Ejemplo 4")
```

```
##
## Ejemplo 4
## ================================================
##                         Dependent variable:
##             -----------------------------------
##                               lsalary
```

```
##                (1)       (2)       (3)       (4)       (5)
## ------------------------------------------------------------
## lsales      0.224*** 0.162*** 0.161*** 0.162*** 0.169***
##             (0.027)  (0.040)  (0.040)  (0.039)  (0.040)
##
## lmktval              0.107**   0.098    0.102    0.101
##                      (0.050)  (0.064)  (0.063)  (0.063)
##
## profits                       0.00004  0.00003  0.00003
##                               (0.0002) (0.0002) (0.0002)
##
## ceoten                                 0.012**  0.014**
##                                        (0.005)  (0.006)
##
## age                                               -0.005
##                                                   (0.005)
##
## Constant    4.961*** 4.621*** 4.687*** 4.558*** 4.778***
##             (0.200)  (0.254)  (0.380)  (0.380)  (0.440)
##
## ------------------------------------------------------------
## R2           0.281    0.299    0.299    0.318    0.322
## Adjusted R2  0.277    0.291    0.287    0.302    0.302
## ============================================================
## Note:                          *p<0.1; **p<0.05; ***p<0.01
```

Por criterio de parsimonia, preferimos, en este caso, el modelo 4 (R cuadrada ajustada igual, o muy similar, a la del modelo 5, pero es un modelo más simple). De aquí, en particular, nos interesa hacer la prueba para ver si profits y age pueden ser omitidos del modelo o no. Para esto tendríamos que comparar al modelo 5 con un modelo que restringiera esas dos variables.

Siempre es posible construir el modelo nosotros mismos, pero R es capaz de crear ese modelo implícitamente. Para esto necesitamos el paquete "car".

```
library(car)

miH0 <- c("age=0","profits=0") # miH0 es un nombre arbitrario
# Esta función le define a R cuales son las variables que queremos
# poner a prueba

linearHypothesis(m5, miH0)
```

```
## Linear hypothesis test
##
## Hypothesis:
## age = 0
## profits = 0
##
## Model 1: restricted model
## Model 2: lsalary ~ lsales + lmktval + profits + ceoten + age
##
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    173 44.079
## 2    171 43.814  2   0.26451 0.5162 0.5977
```

Table 4: Ejemplo 4

	Dependent variable:				
	lsalary				
	(1)	(2)	(3)	(4)	(5)
lsales	0.224***	0.162***	0.161***	0.162***	0.169***
	(0.027)	(0.040)	(0.040)	(0.039)	(0.040)
lmktval		0.107**	0.098	0.102	0.101
		(0.050)	(0.064)	(0.063)	(0.063)
profits			0.00004	0.00003	0.00003
			(0.0002)	(0.0002)	(0.0002)
ceoten				0.012**	0.014**
				(0.005)	(0.006)
age					−0.005
					(0.005)
Constant	4.961***	4.621***	4.687***	4.558***	4.778***
	(0.200)	(0.254)	(0.380)	(0.380)	(0.440)
R^2	0.281	0.299	0.299	0.318	0.322
Adjusted R^2	0.277	0.291	0.287	0.302	0.302
Note:					*p<0.1; **p<0.05; ***p<0.01

En este caso, no podemos rechazar que age y profits no sean significativamente diferentes de 0. Por lo que optamos por el modelo restringido (que sería un modelo 6).

```
m6 <- lm(lsalary~lsales+lmktval+ceoten, data = base)
stargazer(list(m1,m2,m3,m4,m5, m6), type = "text", keep.stat = c("rsq", "adj.rsq"))
```

```
##
## ===============================================================
##                               Dependent variable:
##              ---------------------------------------------------
##                                    lsalary
##                (1)      (2)      (3)      (4)      (5)      (6)
## ---------------------------------------------------------------
## lsales      0.224*** 0.162*** 0.161*** 0.162*** 0.169*** 0.163***
##             (0.027)  (0.040)  (0.040)  (0.039)  (0.040)  (0.039)
##
## lmktval              0.107**   0.098    0.102    0.101   0.109**
##                      (0.050)  (0.064)  (0.063)  (0.063)  (0.050)
##
## profits                       0.00004  0.00003  0.00003
##                               (0.0002) (0.0002) (0.0002)
##
## ceoten                                 0.012**  0.014**  0.012**
##                                        (0.005)  (0.006)  (0.005)
##
## age                                               -0.005
##                                                  (0.005)
##
## Constant    4.961*** 4.621*** 4.687*** 4.558*** 4.778*** 4.504***
##             (0.200)  (0.254)  (0.380)  (0.380)  (0.440)  (0.257)
##
## ---------------------------------------------------------------
## R2           0.281    0.299    0.299    0.318    0.322    0.318
## Adjusted R2  0.277    0.291    0.287    0.302    0.302    0.306
## ===============================================================
## Note:                                *p<0.1; **p<0.05; ***p<0.01
```

También es posible pedir a R los intervalos de confianza por su cuenta mediante el comando `confint(modelo, level = NivelDeConfianza)`. Por default, si omites "level", R lo calcula al 95%.

Table 5: Ejemplo 4.2

	Dependent variable:					
	lsalary					
	(1)	(2)	(3)	(4)	(5)	(6)
lsales	0.224***	0.162***	0.161***	0.162***	0.169***	0.163***
	(0.027)	(0.040)	(0.040)	(0.039)	(0.040)	(0.039)
lmktval		0.107**	0.098	0.102	0.101	0.109**
		(0.050)	(0.064)	(0.063)	(0.063)	(0.050)
profits			0.00004	0.00003	0.00003	
			(0.0002)	(0.0002)	(0.0002)	
ceoten				0.012**	0.014**	0.012**
				(0.005)	(0.006)	(0.005)
age					−0.005	
					(0.005)	
Constant	4.961***	4.621***	4.687***	4.558***	4.778***	4.504***
	(0.200)	(0.254)	(0.380)	(0.380)	(0.440)	(0.257)
R^2	0.281	0.299	0.299	0.318	0.322	0.318
Adjusted R^2	0.277	0.291	0.287	0.302	0.302	0.306
Note:						*p<0.1; **p<0.05; ***p<0.01

Ejemplo 5

Para este ejemplo, es necesario descargar la librería "AER", que contiene bases de datos. En particular, utilizaremos la base llamada "GrowthSW", que contiene los siguientes datos:

- growth: crecimiento anual promedio del PIB real de 1960 a 1995.
- rgdp60: Valor del PIB per cápita en 1960, convertido a dólares de 1960.
- tradeshare: porcentaje promedio de comercio de 1960 a 1995, medido por la suma de exportaciones (X) más importaciones (M), dividido entre el PIB; es decir, el valor promedio de (X + M)/GDP de 1960 a 1995.
- education: años de educación promedio de la población adulta de cada país en 1960.
- revolutions: número de revoluciones anuales, insurrecciones (hayan tenido éxito o no) y golpes de Estado en cada país de 1960 a 1995.
- assassinations: número promedio de asesinatos políticos anuales de 1960 a 1995 (por millones de habitantes).

```
library(AER)
```

```
## Loading required package: lmtest
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Loading required package: survival
```

```
data("GrowthSW") # No es necesario guardarla en un objeto

attach(GrowthSW)
```

Lo que queremos hacer con este ejemplo es mostrar un pequeño truco para generar una regresión con todas las interacciones posibles entre las variables disponibles (cosa que puede resultar preeliminarmente útil para seleccionar interacciones, aunque no debe ser el único criterio).

```
m1 <- lm(growth~.^2, data = GrowthSW)

stargazer(m1, type = "text", title = "Ejemplo 5")
```

```
##
## Ejemplo 5
## ================================================
##                              Dependent variable:
##                          ----------------------
```

```
##                                      growth
## ---------------------------------------------------
## rgdp60                               0.0003
##                                     (0.0004)
##
## tradeshare                           4.368
##                                     (2.606)
##
## education                           1.275***
##                                     (0.443)
##
## revolutions                          -0.955
##                                     (7.896)
##
## assassinations                       4.879*
##                                     (2.728)
##
## rgdp60:tradeshare                    -0.0004
##                                     (0.001)
##
## rgdp60:education                   -0.0001***
##                                    (0.00004)
##
## rgdp60:revolutions                   -0.001
##                                     (0.001)
##
## rgdp60:assassinations                0.001
##                                     (0.001)
##
## tradeshare:education                 -0.431
##                                     (0.518)
##
## tradeshare:revolutions               1.190
##                                    (10.800)
##
## tradeshare:assassinations            -7.397*
##                                     (4.349)
##
## education:revolutions                0.240
##                                     (0.896)
##
## education:assassinations             -1.040
##                                     (0.633)
##
## revolutions:assassinations           -0.913
##                                     (2.332)
##
## Constant                             -2.707
##                                     (2.021)
##
## ---------------------------------------------------
## Observations                          65
## R2                                   0.499
## Adjusted R2                          0.346
```

```
## Residual Std. Error              1.535 (df = 49)
## F Statistic                  3.253*** (df = 15; 49)
## ==============================================================
## Note:                        *p<0.1; **p<0.05; ***p<0.01
```

Además, también es posible crear transformaciones de forma sencilla:

```
m2 <- lm(growth~assassinations+I(assassinations^2), data = GrowthSW)
# Sin I(), R no separará el coeficiente de assassinations^2

m3 <- lm(growth~log(rgdp60), data = GrowthSW)

stargazer(list(m2,m3), type = "text", title = "Ejemplo 5.1")
```

```
##
## Ejemplo 5.1
## ==============================================================
##                               Dependent variable:
##                     ---------------------------------------
##                                      growth
##                             (1)                (2)
## --------------------------------------------------------------
## assassinations             -0.595
##                           (1.459)
##
## I(assassinations2)         0.007
##                           (0.728)
##
## log(rgdp60)                                    0.387
##                                               (0.278)
##
## Constant                  2.106***             -1.042
##                           (0.312)             (2.151)
##
## --------------------------------------------------------------
## Observations                 65                  65
## R2                          0.023               0.030
## Adjusted R2                 -0.009              0.015
## Residual Std. Error  1.905 (df = 62)    1.883 (df = 63)
## F Statistic         0.720 (df = 2; 62) 1.948 (df = 1; 63)
## ==============================================================
## Note:                           *p<0.1; **p<0.05; ***p<0.01
```

Filtrar Tablas

En este ejemplo veremos cómo filtrar datos de una tabla hacia una tabla nueva, y el cómo contar ocurrencias únicas en la tabla. Para esto necesitaran el paquete "dyplr", que está incluido en "tidyverse". Además, deberán descargar el archivo "mmALL_020619_v15.RData"[1] del Drive de Facultad Menor para poder replicar este ejemplo.

[1]Este archivo es parte de un trabajo realizado por David Clark y Patrick Regan, en el que crearon una base de datos sobre protestas, para mayor información, favor de seguir esta liga https://doi.org/10.7910/DVN/HTTWYL

```r
library(tidyverse)

load("mmALL_020619_v15.RData") # no es necesario crear un objeto
# si no lo tienes en tu directorio de trabajo, deberás escribir la ruta completa

head(table) # es como print, pero solo muestra los primeros datos
```

```
## # A tibble: 6 x 31
##       id country ccode  year region protest protestnumber startday startmonth
##    <dbl> <chr>   <dbl> <dbl>  <dbl>   <dbl>         <dbl>    <dbl>      <dbl>
## 1 2.02e8 Canada     20  1990      3       1             1       15          1
## 2 2.02e8 Canada     20  1990      3       1             2       25          6
## 3 2.02e8 Canada     20  1990      3       1             3        1          7
## 4 2.02e8 Canada     20  1990      3       1             4       12          7
## 5 2.02e8 Canada     20  1990      3       1             5       14          8
## 6 2.02e8 Canada     20  1990      3       1             6       19          9
## # ... with 22 more variables: startyear <dbl>, endday <dbl>, endmonth <dbl>,
## #   endyear <dbl>, protesterviolence <dbl>, location <chr>,
## #   participants_category <chr>, participants <chr>, protesteridentity <chr>,
## #   protesterdemand1 <chr>, protesterdemand2 <chr>, protesterdemand3 <chr>,
## #   protesterdemand4 <chr>, stateresponse1 <chr>, stateresponse2 <chr>,
## #   stateresponse3 <chr>, stateresponse4 <chr>, stateresponse5 <chr>,
## #   stateresponse6 <chr>, stateresponse7 <chr>, sources <chr>, notes <chr>
```

```r
table1 <- table %>% # Este es el formato de dyplr para trabajar tablas
  filter(!is.na(startday)) # Para quitar todos los NA

head(table1)
```

```
## # A tibble: 6 x 31
##       id country ccode  year region protest protestnumber startday startmonth
##    <dbl> <chr>   <dbl> <dbl>  <dbl>   <dbl>         <dbl>    <dbl>      <dbl>
## 1 2.02e8 Canada     20  1990      3       1             1       15          1
## 2 2.02e8 Canada     20  1990      3       1             2       25          6
## 3 2.02e8 Canada     20  1990      3       1             3        1          7
## 4 2.02e8 Canada     20  1990      3       1             4       12          7
## 5 2.02e8 Canada     20  1990      3       1             5       14          8
## 6 2.02e8 Canada     20  1990      3       1             6       19          9
## # ... with 22 more variables: startyear <dbl>, endday <dbl>, endmonth <dbl>,
## #   endyear <dbl>, protesterviolence <dbl>, location <chr>,
## #   participants_category <chr>, participants <chr>, protesteridentity <chr>,
## #   protesterdemand1 <chr>, protesterdemand2 <chr>, protesterdemand3 <chr>,
## #   protesterdemand4 <chr>, stateresponse1 <chr>, stateresponse2 <chr>,
## #   stateresponse3 <chr>, stateresponse4 <chr>, stateresponse5 <chr>,
## #   stateresponse6 <chr>, stateresponse7 <chr>, sources <chr>, notes <chr>
```

```r
ProtestasLatAm <- table1 %>% # Para pedir ver datos de países en específico
  filter(country == "Brazil" | # Los dos == es el operador lógico de igual a.
           country == "Mexico" | # El | es el operador lógico de "o"
           country == "Colombia" |
           country == "Argentina" |
           country == "Peru" |
```

```
           country == "Venezuela" |
           country == "Chile" |
           country == "Guatemala" |
           country == "Ecuador" |
           country == "Cuba" |
           country == "Bolivia" |
           country == "Haiti" |
           country == "Dominican Republic" |
           country == "Honduras" |
           country == "Paraguay" |
           country == "El Salvador" |
           country == "Nicaragua" |
           country == "Costa Rica" |
           country == "Panama" |
           country == "Uruguay")

head(ProtestasLatAm)
```

```
## # A tibble: 6 x 31
##       id country ccode  year region protest protestnumber startday startmonth
##    <dbl> <chr>   <dbl> <dbl>  <dbl>   <dbl>         <dbl>    <dbl>      <dbl>
## 1 4.02e8 Cuba       40  1994      3       1             1        5          8
## 2 4.02e8 Cuba       40  1999      3       1             1       15          6
## 3 4.02e8 Cuba       40  2012      3       1             1       24          7
## 4 4.02e8 Cuba       40  2014      3       1             1       21          1
## 5 4.02e8 Cuba       40  2014      3       1             2       30         12
## 6 4.02e8 Cuba       40  2015      3       1             1        4          1
## # ... with 22 more variables: startyear <dbl>, endday <dbl>, endmonth <dbl>,
## #   endyear <dbl>, protesterviolence <dbl>, location <chr>,
## #   participants_category <chr>, participants <chr>, protesteridentity <chr>,
## #   protesterdemand1 <chr>, protesterdemand2 <chr>, protesterdemand3 <chr>,
## #   protesterdemand4 <chr>, stateresponse1 <chr>, stateresponse2 <chr>,
## #   stateresponse3 <chr>, stateresponse4 <chr>, stateresponse5 <chr>,
## #   stateresponse6 <chr>, stateresponse7 <chr>, sources <chr>, notes <chr>
```

```
Represion <- ProtestasLatAm %>%
  filter(stateresponse1 == "arrests" |
           stateresponse1 == "beatings" |
           stateresponse1 == "killings" |
           stateresponse1 == "shootings"|
           stateresponse2 == "arrests" |
           stateresponse2 == "beatings" |
           stateresponse2 == "killings" |
           stateresponse2 == "shootings"|
           stateresponse3 == "arrests" |
           stateresponse3 == "beatings" |
           stateresponse3 == "killings" |
           stateresponse3 == "shootings"|
           stateresponse4 == "arrests" |
           stateresponse4 == "beatings" |
           stateresponse4 == "killings" |
           stateresponse4 == "shootings"|
           stateresponse5 == "arrests" |
```

```
           stateresponse5 == "beatings" |
           stateresponse5 == "killings" |
           stateresponse5 == "shootings"|
           stateresponse6 == "arrests" |
           stateresponse6 == "beatings" |
           stateresponse6 == "killings" |
           stateresponse6 == "shootings"|
           stateresponse7 == "arrests" |
           stateresponse7 == "beatings" |
           stateresponse7 == "killings" |
           stateresponse7 == "shootings")

head(Represion)
```

```
## # A tibble: 6 x 31
##       id country ccode  year region protest protestnumber startday startmonth
##    <dbl> <chr>   <dbl> <dbl>  <dbl>   <dbl>         <dbl>    <dbl>      <dbl>
## 1 4.02e8 Cuba       40  1994      3       1             1        5          8
## 2 4.02e8 Cuba       40  2012      3       1             1       24          7
## 3 4.02e8 Cuba       40  2014      3       1             1       21          1
## 4 4.02e8 Cuba       40  2014      3       1             2       30         12
## 5 4.02e8 Cuba       40  2015      3       1             1        4          1
## 6 4.02e8 Cuba       40  2015      3       1             2       11          1
## # ... with 22 more variables: startyear <dbl>, endday <dbl>, endmonth <dbl>,
## #   endyear <dbl>, protesterviolence <dbl>, location <chr>,
## #   participants_category <chr>, participants <chr>, protesteridentity <chr>,
## #   protesterdemand1 <chr>, protesterdemand2 <chr>, protesterdemand3 <chr>,
## #   protesterdemand4 <chr>, stateresponse1 <chr>, stateresponse2 <chr>,
## #   stateresponse3 <chr>, stateresponse4 <chr>, stateresponse5 <chr>,
## #   stateresponse6 <chr>, stateresponse7 <chr>, sources <chr>, notes <chr>
```

```
country <- c("Brazil" ,
             "Mexico" ,
             "Colombia" ,
             "Argentina" ,
             "Peru" ,
             "Venezuela" ,
             "Chile" ,
             "Guatemala" ,
             "Ecuador" ,
             "Cuba" ,
             "Bolivia" ,
             "Haiti" ,
             "Dominican Republic" ,
             "Honduras" ,
             "Paraguay" ,
             "El Salvador" ,
             "Nicaragua" ,
             "Costa Rica" ,
             "Panama" ,
             "Uruguay")

datos <- data.frame(country)
```

```r
head(datos)
```

```
##     country
## 1    Brazil
## 2    Mexico
## 3  Colombia
## 4 Argentina
## 5      Peru
## 6 Venezuela
```

```r
NumProt <- ProtestasLatAm %>%# Para contar el número de protestas al año por país
  group_by(country) %>% # Le da a R un criterio de agrupamiento
  count(year) # Le dice a R qué queremos que cuente

datos <- merge(x = datos,y = NumProt, by = "country")
# Junta las tablas en una sola

datos <- rename(datos, nprot = n) # Para renombrar y no perdernos
# el formato es nombrenuevo = nombreviejo

head(datos)
```

```
##     country year nprot
## 1 Argentina 1992     2
## 2 Argentina 2008     6
## 3 Argentina 1997     5
## 4 Argentina 1994     7
## 5 Argentina 1995     3
## 6 Argentina 1996     6
```

```r
NumRep <- NumProt <- Represion %>%
  group_by(country) %>%
  count(year)

datos <- merge(x = datos, y = NumProt, by = c("country", "year"))

datos <- rename(datos, nrep = n)

head(datos)
```

```
##     country year nprot nrep
## 1 Argentina 1994     7    2
## 2 Argentina 1996     6    2
## 3 Argentina 1997     5    2
## 4 Argentina 1999     4    1
## 5 Argentina 2000     4    2
## 6 Argentina 2001    11    2
```

Creando un pdf

Bien, quizás te estés preguntando cómo hacer un pdf en R. Para esto, es necesario seguir una serie de pasos:

1. Instalar el paquete RMarkdown. (Las instrucciones para usarlo a su potencial completo están aquí).
2. Descargar e instalar el lenguaje de programación LaTex (disponible aquí).
3. Adicionalmente, si tienes una MAC o Linux, deberás instalar y activar el paquete "tinytex", siguiendo los siguientes pasos:

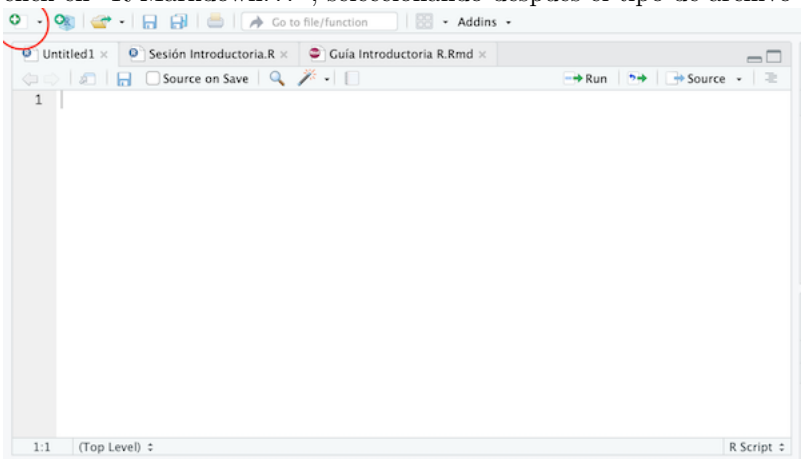
install.packages("tinytex")
library(tinytex)
tinytex::install_tinytex()

Para MAC, alternativamente, puedes seguir las instrucciones en esta **liga**. Se trata de una versión más completa de LaTex, pero no es necesaria para replicar lo que hemos enseñado en esta guía.

Para ver particularidades de formato en RMarkdown, puedes encontrar casi toda la información necesaria en estos cheatsheets:

- https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf
- https://rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
- https://github.com/rstudio/cheatsheets/raw/master/rmarkdown-2.0.pdf

Para abrir un archivo nuevo, es cuestión de seguir los mismos pasos que para abrir un nuevo script, y dar click en "R Markdown...", seleccionando después el tipo de archivo que deseas crear.



Una vez que crees un nuevo archivo de R Markdown, este incluirá una plantilla que tendrá las cosas básicas preparadas (la sección de información general del archivo) además de incluir un par de ejemplos sobre cómo funciona R Markdown.

R Markdown te permite replicar tu código dentro del pdf (tal como has visto que hemos hecho en esta guía). Esto se hace por medio de espacios llamados "chunks", abiertos con tres acentos graves (o acento invertido) y cerrados de la misma manera.

Los chunks deben contener información sobre el lenguaje de programación que usarás en ellos y un id para el chunk.

```
```{r nombredechunk}
# Tu código/sección de tu código
```
```

Si no quieres que tu código aparezca en el pdf, es cuestión de escribir al inicio de tu chunk:

```
{r nombredechunk, echo = FALSE, message = FALSE}
```

Si quieres revisar cómo se está viendo tu pdf, o si has concluido, oprime el botón "Knit". Esto generará una visualización de cómo se está viendo, además de guardar el pdf en tu directorio de trabajo.

Referencias

Clark, David; Regan, Patrick, 2016, "Mass Mobilization Protest Data", https://doi.org/10.7910/DVN/HTTWYL, Harvard Dataverse, V3, UNF:6:/IpCl1u0zpeC/hwamcT+Ng== [fileUNF]
Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables. R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
Lecuanda, José Manuel."Laboratorios de R". Métodos Estadísticos para Ciencia Política y Relaciones Internacionales. México: Semestre Otoño 2019.