

Apuntes del curso
Introducción a la estadística con R

S. Ardanza-Trevijano y A. Garcimartín

2019

Apuntes del curso Introducción a la estadística con R

Sergio Ardanza-Trevijano Moras

Angel Garcimartín Montero

Departamento de Física y Matemática Aplicada
Facultad de Ciencias
Universidad de Navarra
31080 Pamplona

Índice general

1. Preliminares: Instalación de R y RStudio	9
2. Estadística descriptiva y procedimientos básicos	11
Aritmética y Entorno	11
Vectores y matrices	12
Crear un vector: concatenar, secuencia, repetición	12
Máximo, mínimo, rango, longitud, operaciones lógicas, ordenación.	12
Matrices	14
Tipos de datos	14
Funciones matemáticas elementales	15
Estadística descriptiva elemental	15
Entrada y Salida	16
Probabilidad	16
Gráficos elementales	19
Inferencia Estadística	21
Control de flujo	23
3. Lectura y escritura de datos con R	27
Funciones principales para lectura de datos en R	27
read.table()	27
write.table()	28
Ejemplo	28
Optimizar la lectura	29
Calculando la memoria necesaria	29
Escritura de archivos.	29
Leer archivos de texto con readr.	29
Leer archivos con el menú import dataset.	30
Conexiones	30
Otros tipos de datos.	31
4. Manipulación de bases de datos	33
Preliminares	33
Tipos de Datos y asignaciones (recordatorio).	33
Objetos	33
Asignaciones	34
Vectores	34
Subconjuntos de datos	37
Operadores [], [[]] y \$	37
Ejemplos de subconjuntos	37
Valores perdidos (NA)	39
Eliminar NA	40

Introducción al Tidyverse, dplyr y tidyr	40
dplyr o base R.	40
Manipulación de dataframes con dplyr	41
pipes ‘%>%’	43
Filtros con dplyr	43
Calculos en grupos con dplyr	43
split-apply-combine con dplyr	43
dplyr group_by	44
summarize con dplyr	45
mutate con dplyr	47
arrange (ordenar) con dplyr	48
Organizar datos de manera tidy.	49
Tidying Data: Formatos ancho y largo.	49
Transformando Gapminder en tidy	50
tidyr	51
gather tidyr	51
select en tidyr	52
separate en tidyr	53
spread en tidyr	54
Unión de bases de datos.	55
5. Gráficos	59
Funciones de alto nivel	59
Funciones de bajo nivel	62
Parámetros gráficos	63
Propiedades de los elementos gráficos (símbolos, líneas, texto . . .)	63
Propiedades de los ejes, márgenes, dimensiones	64
Consideraciones finales	68
6. Introducción a ggplot2	69
Preliminares	69
Partes de un gráfico en ggplot	69
Ejemplo básico	70
Paneles	78
Escalas	80
Temas	81
Otras geometrías	83
Paletas de colores. El paquete RcolorBrewer	90
Cómo guardar gráficos con ggplot2	92
Gráficos interactivos con manipulate	93
<i>html widgets</i> . Exportando gráficos en plotly	93
7. Markdown y R	95
Formateo del texto	96
Cómo insertar código en R	98
Encabezado. Tipos de documento.	99
8. Inferencia	101
Intervalos de confianza	101
Tests de hipótesis	102
Test paramétrico para una media	102
Comparación de dos medias	105

Tests no paramétricos para las medias	107
Test de proporciones	108
Tablas de contingencia	110
Test de independencia	110
Test de homogeneidad	112
9. Regresión Lineal	115
Ajuste lineal	115
Fórmulas	118
Bandas de confianza	121
Correlación	123
10. Anova	125
Anova de una vía.	125
Test Post-Hoc	127
Verificación de las condiciones	130
Alternativa no paramétrica. Test de Kruskal-Wallis.	131
Transformaciones de los datos	131
Anova de dos vías	133
Otros diseños.	135
11. Bibliografía y recursos	137

Prefacio

Sirvan estas líneas no solo de presentación sino también de guía, para aprovechar el material aquí contenido. Estos apuntes no pretenden ser sino un resumen de la teoría necesaria para aprender a usar el lenguaje R, con vistas a la estadística y al tratamiento de datos básico. Se explican los conceptos y procedimientos necesarios para implementar las tareas más habituales. Estas ideas deben ser objeto de estudio personal *antes* de asistir a las sesiones del curso.

No hemos pretendido escribir unos apuntes de estadística, ni de R. Hay magníficos libros, algunos de los cuales se citan en la bibliografía. No se proporcionan demostraciones ni se hace especial énfasis en el rigor matemático. El objetivo es que quienes usen la estadística como una herramienta de trabajo logren un conocimiento operativo de la manera más rápida y eficiente. No es una obra sistemática, se repiten los conceptos en distintos lugares bajo distintos aspectos, y en varios casos se sacrifica el rigor en aras de la sencillez. El formato de los capítulos es muy variado: en algunos casos está redactado como un manual, en otros se incluyen diapositivas, hay intercalados algunos documentos que hemos juzgado interesantes, etc.

Un método que suele dar buen resultado es proponerse como objeto de estudio un problema real que uno tenga entre manos, y tratar de resolverlo al hilo de este curso. Si se carece de tal cosa, se puede pedir un proyecto –unos datos, o un caso– para ir aplicando los conceptos.

Créditos referentes al origen de los datos

En este texto hemos usado datos públicos obtenidos de diversas fuentes. Parte de ellos se hallan en la misma distribución de R y RStudio y desde el programa mismo se puede acceder a la información sobre su origen. Para más información, visitar las páginas Web del proyecto R <https://cran.r-project.org/> y RStudio <https://rstudio.com/>. Además, algunas *cheatsheets* reproducidas aquí se ofrecen libremente en la Web de RStudio: <https://rstudio.com/resources/cheatsheets/>

Se han empleado también datos puestos a disposición del público por el Cornell Lab of Ornithology <https://ebird.org/home>, y por el Statistics Online Computational Resource de la University of California at Los Angeles <http://www.socr.ucla.edu/> y http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data. También se ha hecho uso de los datos públicos de Gapminder: <https://www.gapminder.org/>, distribuidos desde esa página o por otros cauces.

Otros datos son propios de los autores. Algunos (como se puede deducir por el contexto o si se indica expresamente) son inventados.

Algunos ejemplos están inspirados en el *SCF/D-Lab R bootcamp at UC Berkeley, August 2017*, disponible en GitHub: <https://github.com/berkeley-scf/r-bootcamp-2017>, y en el repositorio *R Programming for Data Science*, creado por Roger D. Peng, también disponible libremente en GitHub: <https://github.com/rdpeng/rprogdatascience>.

En cualquier caso, hay que tener en cuenta que la finalidad que se persigue es puramente didáctica. No se deben emplear los datos y los análisis como referencia de autoridad; no respondemos tampoco de la exactitud de los mismos.

Si usted advierte que por descuido hemos empleado algún otro material sin atribuirlo correctamente, por favor póngase en contacto con los autores para que corriamos el error.

Todas las URL citadas fueron consultadas en noviembre de 2019.

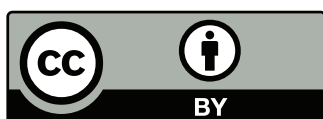
Los autores agradeceremos cualquier sugerencia o crítica que se nos haga.

Distribución

Esta obra se puede distribuir libremente, utilizar extractos, adaptarla, o incluirla en otras obras, incluso con fines comerciales, siempre que se cite el origen y se atribuya a los autores. Si se realizan cambios, así debe indicarse.

Esta obra está licenciada bajo la Licencia Creative Commons Atribución 4.0 Internacional. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by/4.0/>.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.



Capítulo 1

Preliminares: Instalación de R y RStudio

El lenguaje R y el entorno RStudio resultan una gran ayuda para los cálculos estadísticos. Aprenderlos exige un cierto esfuerzo, pero las ventajas que se obtienen son muy grandes. Es una de las razones de su popularidad.

Un lenguaje de programación consiste en una serie de comandos para realizar tareas específicas. Estos comandos se pueden ejecutar secuencialmente, dando lugar a programas. Las reglas gramaticales se denominan *sintaxis*. Al igual que hay muchos modelos de vehículos adaptados a cometidos muy diversos, hay también muchos lenguajes de programación, cuyas características los hacen más aptos para algunas tareas. **R** es un lenguaje de programación con un particular énfasis en la estadística. Ha sido desarrollado benévolamente por una serie de colaboradores que lo ofrecen completamente gratis (es *software libre*). Además, como el código fuente es abierto, se pueden lograr muy fácilmente versiones para casi todos los sistemas operativos. Se puede descargar de una red de repositorios de Internet para cualquier plataforma (Linux, Windows, Mac OS): para instalarlo se necesita simplemente un ordenador (*cualquier* ordenador, de hecho) y una conexión a Internet.

El lenguaje de programación R y el entorno RStudio.

Trabajar con R a secas es perfectamente factible, pero una interfaz de usuario (más apropiadamente, un IDE: *integrated development environment*) llamada RStudio puede servir de gran ayuda. RStudio llama a R y aporta algunas características adicionales (un editor para escribir programas, gráficos y ayuda integrados, por poner algunos casos). RStudio ha sido desarrollado por un equipo que ofrece un producto comercial a empresas; pero una versión del programa, llamada RStudio Desktop, es gratuita y de código abierto.

Un detalle antes de la instalación. A RStudio y a R no le gustan los nombres con acentos o con espacios en blanco ni las ñ. Algunos sistemas operativos los manejan sin problemas; pero es mejor evitarlos. Por ejemplo, es buena idea instalar estos programas en una carpeta con un nombre sencillo y sin esas florituras.

En primer lugar hay que descargar R e instalarlo. Después, tras comprobar que R funciona correctamente, se descarga e instala RStudio. **R** se puede encontrar en el CRAN, la *comprehensive R archive network*. CRAN es un conjunto de servidores distribuidos por todo el mundo en el que se almacenan los componentes de R. Se puede ir directamente a <https://cran.r-project.org/> y escoger el servidor más cercano en el menú, o bien usar un servidor en la nube: <https://cloud.r-project.org/>, que automáticamente decidirá cuál es el mejor servidor para uno. En ambos casos, se redigirá a una página de descargas, donde debe escogerse el sistema operativo específico. Hay que prestar

atención a esto: los archivos son diferentes dependiendo del sistema. Esto es especialmente importante para el MacOS. Descarga R en tu ordenador e instálalo (es muy sencillo; si alguna pregunta no la comprendes, deja la marcada la opción que viene por defecto).

Comprueba que has instalado correctamente **R**: ejecútalo y escribe en el *prompt* (la línea que comienza con un signo `>`) una operación sencilla:

```
1+1
```

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

Si funciona, sal de R escribiendo `quit()`.

Ahora es cuando se puede instalar RStudio Desktop. Ir a <https://www.rstudio.com/products/rstudio/#Desktop> y asegúrate de escoger RStudio Desktop Open Source y no otra versión. Descárgala e instálala. (De nuevo: si no entiendes alguna pregunta, deja la opción por defecto.)

Al ejecutar RStudio, encontrarás un entorno con ventanas cuya apariencia depende del sistema operativo. Una de ellas se llama *Console*. Si se escribe allí alguna operación matemática sencilla, como antes, y se obtiene la respuesta correcta, RStudio está listo para usarse.

Capítulo 2

Estadística descriptiva y procedimientos básicos

En este capítulo se incluyen, con un formato característico, comandos en R y la salida a que dan lugar (los ejemplos son a veces más rápidos que una explicación). Así, la instrucción

```
2+1
```

```
## [1] 3
```

calcula la suma indicada y muestra la salida tras el símbolo `##`.

Este es un resumen donde solo se incluye lo elemental. En general, los comandos admiten muchas opciones y variantes; para más información se puede consultar la bibliografía y la ayuda de R Studio.

Aritmética y Entorno

Para establecer la carpeta de trabajo y averiguar cuál es, se emplean `setwd()` y `getwd()` respectivamente:

```
getwd()
```

```
## [1] "D:/docencia"
```

```
setwd('D:/docencia/master/cursoR/material')
```

```
getwd()
```

```
## [1] "D:/docencia/master/cursoR/material"
```

Para salir de R (y Rstudio), `quit()`

El operador de asignación `<-` (también sirven `=` y `->`) asigna un valor a una variable. La suma, resta, multiplicación, división y potenciación son `+` `-` `*` `/` `^`. Para mostrar el valor de una variable, se escribe su nombre o se emplea `print()`.

```
altura <- 5
```

```
2 -> radio
```

```
volumen = pi * (radio^2) * altura
```

```
print(volumen)
```

```
## [1] 62.83185
```

Lista de variables en el entorno y borrar una variable:

```
ls()
```

```
## [1] "altura" "radio" "volumen"
```

```
rm(volumen)
```

```
ls()
```

```
## [1] "altura" "radio"
```

Existen muchos *paquetes* (conjuntos de funciones) de R para tareas específicas. Por ejemplo: *lubridate* es un paquete para facilitar el formateo de fechas. Si se quiere utilizar un paquete, primero se debe descargar e instalar (una sola vez), y después cargar (cada vez que se usen). Instala tú mismo el paquete **readxl**, que contiene funciones para leer hojas de cálculo de Excel. Desde la pestaña **Packages** de RStudio se descarga e instala. Cuando se quiera usar la función `read_excel`, contenida en el paquete, hay que cargarlo así, con el comando `library`:

```
library(readxl)
```

Vectores y matrices

Crear un vector: concatenar, secuencia, repetición

```
v1 = c(3, -1, 6.44) # c viene de concatenar
v1
```

```
## [1] 3.00 -1.00 6.44
```

```
v2 = seq(from=1,to=3,by=0.2) # secencia equiespaciada
v2
```

```
## [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
```

```
v3=seq(0,7,pi) # se puede omitir from= to= by=
v3
```

```
## [1] 0.000000 3.141593 6.283185
```

```
v4 = c(rep('A',5),rep("B",3)) # las comillas simples y dobles son equivalentes
v4
```

```
## [1] "A" "A" "A" "A" "A" "B" "B" "B"
```

```
v5=rep(v4,2)
v5
```

```
## [1] "A" "A" "A" "A" "A" "B" "B" "B" "A" "A" "A" "A" "A" "B" "B" "B"
```

Máximo, mínimo, rango, longitud, operaciones lógicas, ordenación.

```
max(v1) ; min(v1) # se pueden escribir dos comandos seguidos separados por ;
```

```
## [1] 6.44
```

```
## [1] -1
range(v2)

## [1] 1 3
length(v2)

## [1] 11
v2[0] # el primer índice empieza en 1, el 0 no existe: dará un elemento vacío

## numeric(0)
v2[1] # el primer componente es el 1

## [1] 1
v2[5] # el quinto

## [1] 1.8
```

Las condiciones son operaciones relacionales que resultan en VERDADERO o FALSO Para ello se emplean los símbolos <, <=, >, >=, ==, !=. Además, el *and* y *or* lógicos se indican con & y |.

```
w3 <- (v3>pi) # la condición se evalúa para cada elemento del vector
print(w3)

## [1] FALSE FALSE TRUE

w4 = (v2==2)
print(w4)

## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
w5 = v2[v2>2] # los elementos del vector v2 para los que la condición se cumple
print(w5)

## [1] 2.2 2.4 2.6 2.8 3.0
v2[w4] # w4 es cierto en los elementos tales que (v2==2)

## [1] 2
w1 = sort(v1)
print(w1)

## [1] -1.00 3.00 6.44
order(v1) # con estos índices se podría ordenar otro vector

## [1] 2 1 3
# por ejemplo: si v1 fueran las abscisas y v3 las ordenadas de unos puntos,
# al ordenar v1 se haría necesario reordenar v3 con el mismo orden
v3[order(v1)] # elementos segundo, primero, tercero de v3

## [1] 3.141593 0.000000 6.283185
w2=sort(v1,decreasing=TRUE) # por defecto, decreasing=FALSE
print(w2)
```

```
## [1] 6.44 3.00 -1.00
```

Matrices

```
M = matrix(seq(1,11,2),nrow=2,ncol=3)
```

```
M
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    5    9
```

```
## [2,]    3    7   11
```

```
M[,2] # segunda columna
```

```
## [1] 5 7
```

```
M[1,] # primera fila
```

```
## [1] 1 5 9
```

```
M[2,3] # elemento de la fila 2, columna 3
```

```
## [1] 11
```

Tipos de datos

Por vía de ejemplo:

```
nombres= c('Eminem','Madonna',"U2") # vector de caracteres
```

```
PrimerDisco = c(1999, 1982, 1980) # variable numérica
```

```
print(PrimerDisco)
```

```
## [1] 1999 1982 1980
```

```
nac = c('US','US', 'Ireland') # vector de caracteres
```

```
nac
```

```
## [1] "US"      "US"      "Ireland"
```

```
NN=factor(nac) # un 'factor' es una variable cualitativa
```

```
NN
```

```
## [1] US      US      Ireland
```

```
## Levels: Ireland US
```

```
aprecio = c('poco','mucho','mucho')
```

```
aprecio
```

```
## [1] "poco"  "mucho" "mucho"
```

```
# al definir como factor se pueden indicar todos los niveles
```

```
aprecio=factor(aprecio,levels=c("mucho","regular","poco"),ordered=TRUE)
```

```
aprecio
```

```
## [1] poco  mucho mucho
```

```
## Levels: mucho < regular < poco
```



```
banda3 <- list(nombre="U2", miembros=4, ventas = '107000000') # lista
banda3
```

```
## $nombre
## [1] "U2"
##
## $miembros
## [1] 4
##
## $ventas
## [1] "107000000"
```

```
banda3$miembros # para acceder a un objeto de la lista se usa $
```

```
## [1] 4
```

Las listas pueden contener tipos de datos heterogéneos. Las variables se pueden agrupar en **data frames**, que comparten muchas propiedades de las listas.

```
mi_mus = data.frame(nombres, NN, PrimerDisco, precio)
mi_mus
```

```
##  nombres      NN PrimerDisco precio
## 1  Eminem      US      1999 poco
## 2  Madonna     US      1982 mucho
## 3      U2 Ireland     1980 mucho
```

```
mi_mus$PrimerDisco # se accede a las columnas como en las listas
```

```
## [1] 1999 1982 1980
```

Funciones matemáticas elementales

Las funciones trigonométricas son **sin**, **cos** y **tan** para seno, coseno y tangente. La exponencial es **exp**, el logaritmo neperiano **log**, y la raíz cuadrada **sqrt**. Redondear al entero más cercano es **round**, y redondear hacia arriba o hacia abajo, **ceiling** y **floor** respectivamente.

Estadística descriptiva elemental

Si se tienen unos datos cualesquiera, la media, la desviación típica, la varianza y los cuantiles se pueden obtener con **mean**, **sd**, **var** y **quantile**:

```
mean(PrimerDisco)
```

```
## [1] 1987
```

```
var(PrimerDisco)
```

```
## [1] 109
```

```
sd(PrimerDisco)
```

```
## [1] 10.44031
```

```
quantile(PrimerDisco, prob=0.5) # esto sería la mediana
```

```
## 50%
## 1982
```

Entrada y Salida

Probablemente, lo más sencillo sea utilizar el menú *Import Dataset* desde la pestaña *Environment* en RStudio. En el diálogo que aparece, se pueden escoger diversas opciones, como el punto decimal, el separador de columnas, etc. Si se quiere luego realizar la tarea programáticamente, se puede copiar y pegar la instrucción que aparece en la consola cuando el resultado sea satisfactorio. La mayoría de las veces bastan estas funciones: `read.csv`, `read.delim`, `read.table` y `read_excel`, con las opciones adecuadas.

Para escribir datos en un fichero del disco, se emplean preferentemente `write.table` y `write.csv`.

Los *scripts* son comandos de R guardados como ficheros de texto con la extensión `.R`. Si se abren en el editor de RStudio, se pueden ejecutar línea por línea poniendo el cursor encima y apretando el botón *Run*, o bien ejecutar todo el *script* apretando el botón *Source*. Se puede llamar a un *script* desde otro *script* con el comando `source(nombre del archivo.R)`.

Probabilidad

R se puede utilizar como una calculadora para obtener valores numéricos concernientes a las principales distribuciones estadísticas (como calcular el área bajo la curva, obtener una muestra aleatoria de una distribución o hallar los cuantiles). Estas tareas se llevan a cabo con funciones que comienzan por las letras **d p q r** seguidas de una abreviatura del nombre de la distribución.

- **funciones d** : proporcionan la **probabilidad** de un valor dado si se trata de una distribución discreta, o el valor de la **pdf** (*probability density function*) para distribuciones continuas. **d** viene de *density*.
- **funciones p** : calculan numéricamente la integral (o suma) de la *pdf* desde $-\infty$ hasta el valor indicado de la variable aleatoria. En otras palabras, devuelven la **Función de distribución** (*F*), en inglés **Cumulative Distribution Function** (o también, **Probability Distribution Function**) para cierto valor de la variable aleatoria. **p** es apócope de *probability*.
- **funciones q** : dan el valor de la variable aleatoria correspondiente al cuantil *p* (*p* se le debe pasar a la función). Se llaman **q** por *quantile*.
- **funciones r** : devuelven una muestra aleatoria de la distribución estadística con los parámetros especificados. **r** significa *random*.

A estas funciones se les deben pasar como argumentos los parámetros de la distribución, como la media, la varianza, o cualquier otro valor que se necesite para definirla. A continuación se ofrece un ejemplo con la distribución normal.

La abreviatura de la distribución normal (o Gaussiana) es *norm*. Por tanto, las funciones son `dnorm`, `pnorm`, `qnorm` y `rnorm`. La expresión analítica de la función de densidad es

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

donde μ es la media y σ la desviación típica. El comando para obtener la *pdf* en x , o sea, $f(x)$, es `dnorm(x,mean=a,sd=b)`. Por ejemplo: si $\mu = 0$ y $\sigma = 1$, $f(0) = \frac{1}{\sqrt{2\pi}} e^{-\frac{0}{2}} = \frac{1}{\sqrt{2\pi}} = 0,3989423$. Con `dnorm`:

```
x=0
dnorm(x,mean=0,sd=1)
```

```
## [1] 0.3989423
```

Nótese que x puede ser un vector. Si no se indica, por defecto se toma $\mu = 0$ y $\sigma = 1$

```
x=seq(-2,2,1)
x
```

```
## [1] -2 -1 0 1 2
```

```
dnorm(x)
```

```
## [1] 0.05399097 0.24197072 0.39894228 0.24197072 0.05399097
```

Si $f(x)$ es la *pdf*, entonces la función de distribución F se define como

$$F(x) = P(X \leq x) = \int_{-\infty}^x f(y) dy$$

Para la normal,

$$F(x) = \int_{-\infty}^x \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy$$

que no se puede calcular exactamente en términos de funciones elementales. La integral numérica se obtiene con `pnorm`. Se puede interpretar como el área bajo la *pdf* hasta el valor considerado.

Evidentemente, $F(\infty) = 1$, por normalización, y $F(0)$ debe valer 0.5 por simetría.

```
pnorm(1000,mean=0,sd=1) # 1000 es a todos los efectos infinito
```

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

```
pnorm(0)
```

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

```
pnorm(-1)
```

```
## [1] 0.1586553
```

(si la media es 0 y la desviación típica es 1 se pueden omitir, ya que por defecto se toman esos valores). El valor del área bajo la curva de la región complementaria a $x < -1$ sería $1 - F(x)$; se puede obtener también con la siguiente sintaxis:

```
pnorm(-1,lower.tail=FALSE)
```

```
## [1] 0.8413447
```

La función de distribución, que devuelve el área bajo la *pdf*, proporciona entonces una manera sencilla de calcular la probabilidad de que un individuo seleccionado al azar caiga en un determinado rango de valores. Recuerdese que

$$P(x_1 < X \leq x_2) = \int_{x_1}^{x_2} f(y) dy = F(x_2) - F(x_1)$$

Por ejemplo, en el caso de una distribución normal tipificada, la probabilidad de que una muestra tomada aleatoriamente valga entre -1 and 2 es

$$P(-1 < X \leq 2) = \int_{-1}^2 f(y) dy = F(2) - F(-1)$$

Otro ejemplo. Si las alturas de una población están distribuidas normalmente con una media $\mu = 178$ cm y una desviación típica de $\sigma = 9$ cm, ¿cuál es la probabilidad de que la altura de una persona elegida al azar esté entre 176 y 180 cm?

```
pnorm(180,mean=178,sd=9)-pnorm(176,mean=178,sd=9)
```

```
## [1] 0.1758591
```

Siguiendo con este ejemplo (media, 178 cm; desviación típica, 9 cm), podemos preguntarnos cuál es la altura por debajo de la cual se halla el 90% de la población (que es precisamente la definición de cuantil 0.90, o percentil 90). Se calcula con `qnorm`:

```
qnorm(0.90,mean=178,sd=9)
```

```
## [1] 189.534
```

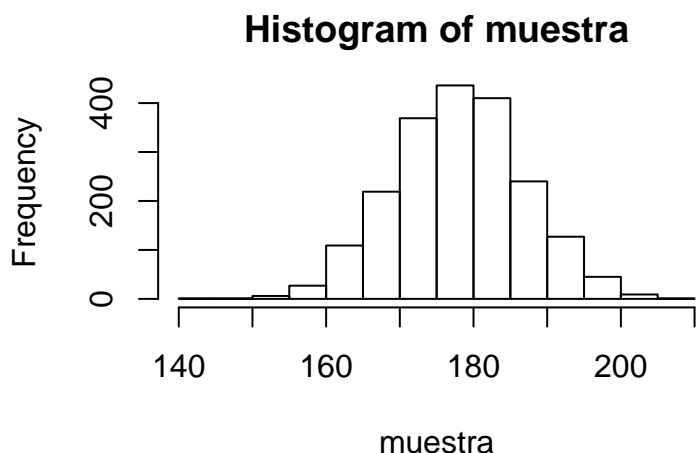
Los cuartiles se pueden calcular con una sola instrucción, puesto que la función `qnorm` admite un vector como argumento:

```
v=c(0.25,0.5,0.75)
qnorm(v,mean=178,sd=9)
```

```
## [1] 171.9296 178.0000 184.0704
```

Se obtiene una muestra aleatoria de n valores tomados de una distribución normal mediante el comando `rnorm`, que toma como argumento el tamaño de la muestra deseado. Si deseamos dos mil valores aleatorios de las alturas de la población anterior y dibujar su histograma, se hace así:

```
muestra = rnorm(2000,mean=178,sd=9)
hist(muestra)
```



Para otras distribuciones, se puede obtener de manera similar la función de densidad, la función de distribución, los cuantiles y una muestra aleatoria. Para ello, se emplean las funciones **d p q r** seguidas de la abreviatura de la distribución. Algunas son:

- Distribución binomial, abr. `binom`
- Distribución de Poisson, abr. `pois`
- Distribución F de Fisher-Snedecor, abr. `f`
- Distribución t de Student, abr. `t`
- Distribución χ^2 , abr. `chisq`

Así, la *pdf* de una distribución de Poisson se obtiene con `dpois`. En la ayuda de R se puede ver que hay que suministrar como argumento el parámetro λ , que es la media de la distribución de Poisson. Los parámetros que se deben introducir como argumento son específicos para cada distribución.

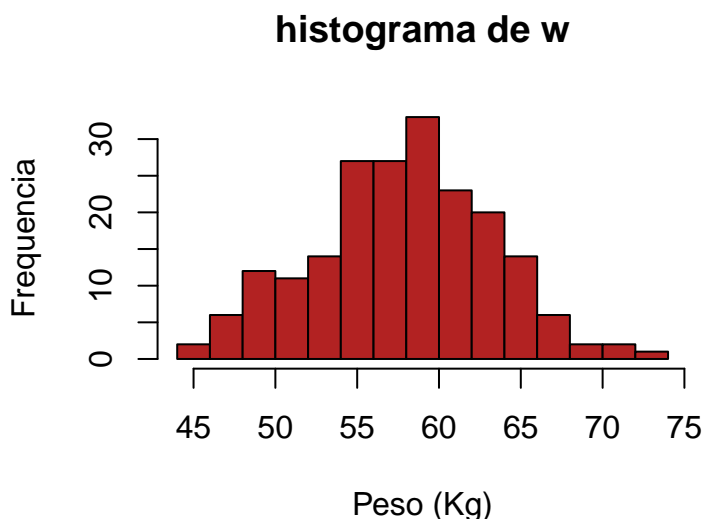
En un anexo se ofrece una explicación de las funciones `d p q r` con ejemplos gráficos.

Gráficos elementales

(En un capítulo posterior se explican con más profundidad).

Tomemos los datos del fichero `hw.txt`, que contiene la altura y peso de unas doscientas personas, y dibujemos el **histograma** del peso con 15 bins:

```
datos = read.table('hw.txt',header=FALSE)
# si las columnas no tienen encabezamiento, se nombran automáticamente V1, V2, ...
w = datos$V2
hist(w,15, main="histograma de w", xlab="Peso (Kg)", ylab="Frecuencia",
     col="firebrick")
```

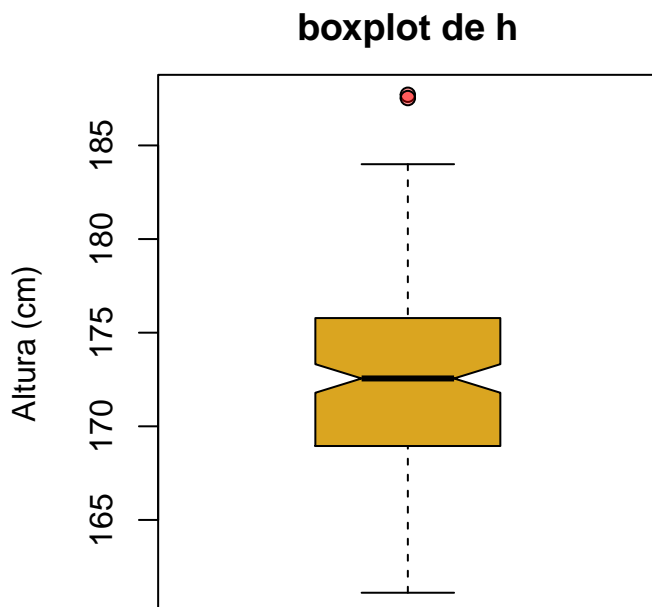


`col` es el color (hay muchísimos definidos en R), `xlab` el *label* (título) del eje x, e `ylab` el del eje y.

Un **diagrama de cajas** (*boxplot*) de la primera columna, con muescas (`notch = TRUE`), los valores atípicos con círculos sólidos con borde (`outpch=21`) de color rojo pero un 40% transparente

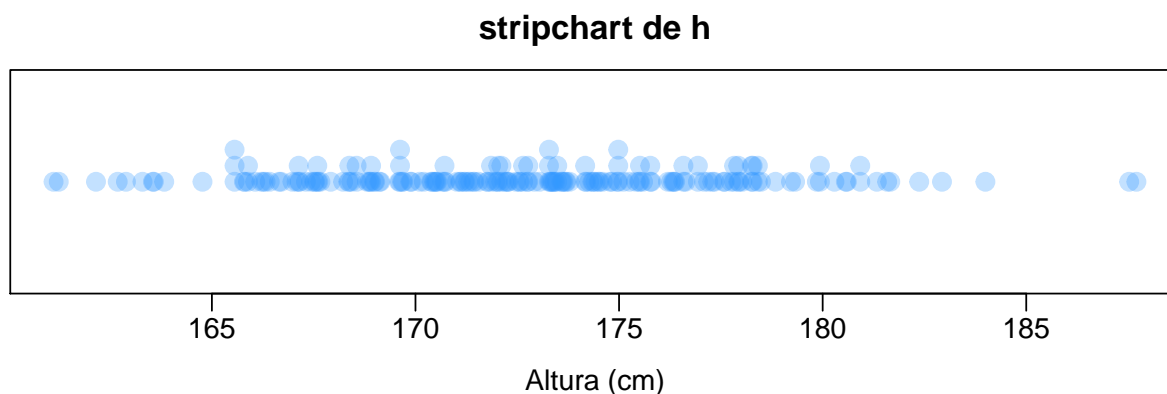
(`outbg=rgb(1,0,0,0.4)`) –¡si no fuera un poco transparente, casi no se vería que hay dos puntos!– se dibuja así:

```
h = datos$V1
boxplot(h, main="boxplot de h", notch = TRUE, ylab="Altura (cm)",
        col='goldenrod', outpch=21, outbg=rgb(1,0,0,0.4))
```



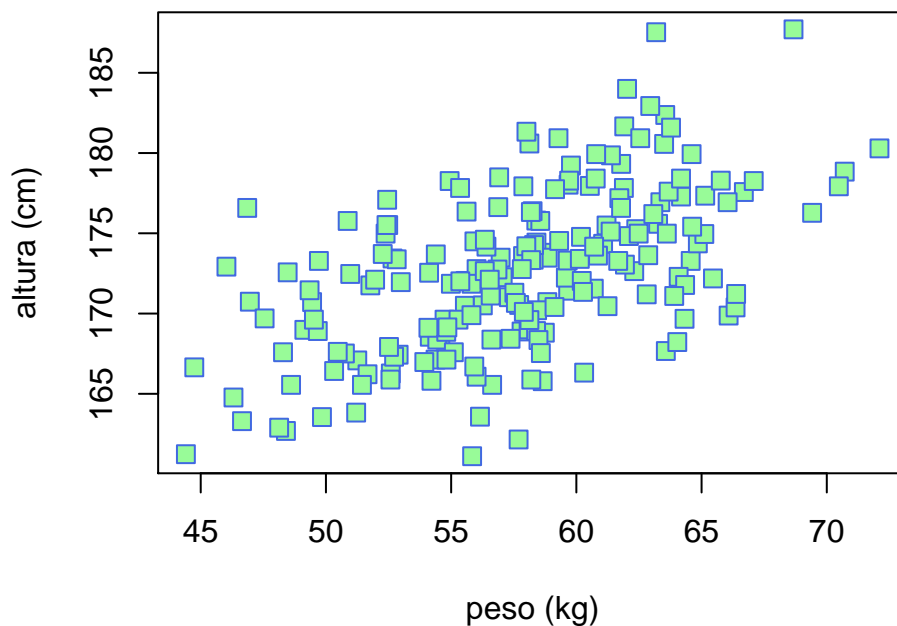
A continuación, un *stripchart* de `h` con círculos sólidos (`pch=19`) de color azulado (`rgb=[0.2, 0.6, 1]`) con un 30% de transparencia, para que se vean las acumulaciones (`rgb=[0.1, 0.6, 1, 0.3]`) de un tamaño un 40% más grande que el normal (`cex=1.4`), dibujado de modo que si varios puntos caen exactamente en el mismo valor, se dispongan en columna (`method="stack"`):

```
stripchart(h, main="stripchart de h", method="stack",
          xlab="Altura (cm)", pch=19, col=rgb(0.2,0.6,1,0.3), cex=1.4)
```



Para representar puntos de dos variables, lo más sencillo es un diagrama cartesiano:

```
plot(w,h,pch=22,col='royalblue',bg='palegreen',cex=1.4,
     xlab='peso (kg)',ylab='altura (cm)')
```



Obviamente, hay un sinnúmero de opciones gráficas para modificar la apariencia de la representación. Una función llamada `par` define los parámetros gráficos, y permite retocar las presentaciones de una manera muy flexible.

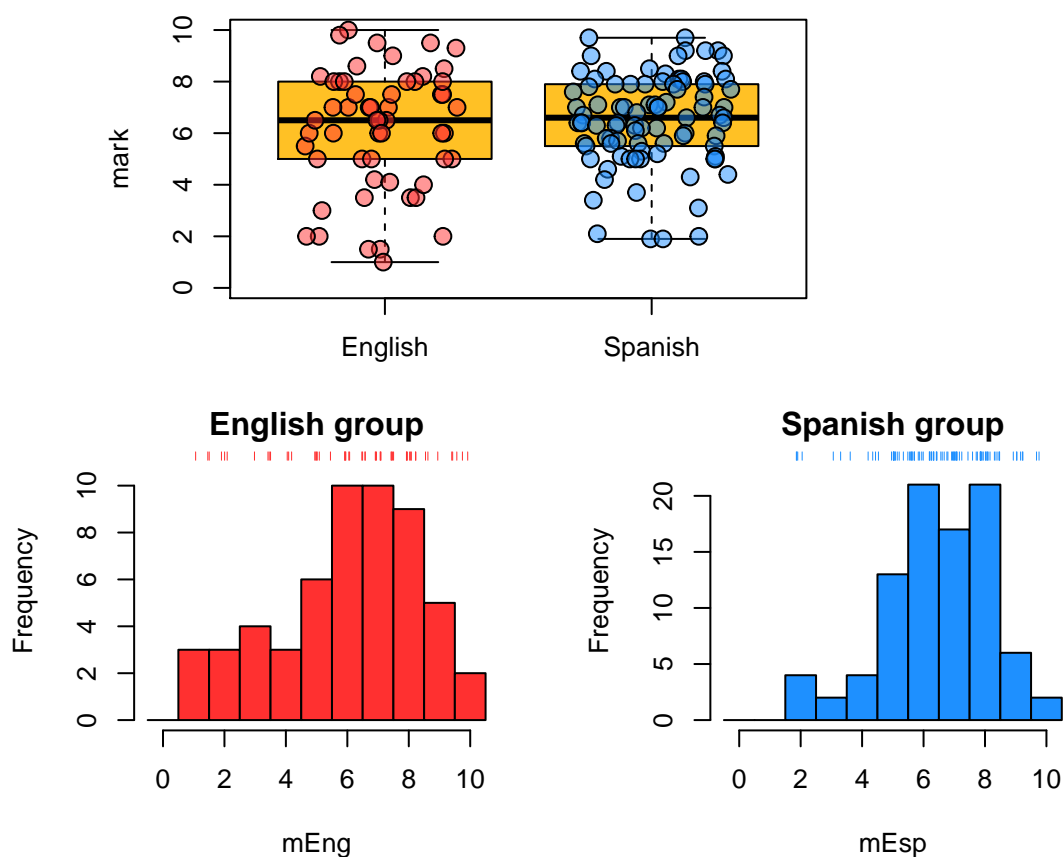
Nótese que `plot`, así como muchos otros comandos gráficos, crea una nueva figura. Para sobrescribir la representación sobre una anterior, se añade la opción `add=TRUE`. O bien, si se desea añadir puntos o líneas, se pueden utilizar los comandos `points` y `lines` respectivamente, que *añaden* esos elementos a un gráfico ya existente (si no existe uno, da error). Es también útil `abline`, que añade rectas a un gráfico (la sintaxis es un poco diferente a `lines`).

Inferencia Estadística

R es un lenguaje orientado a objeto. Es bueno saberlo, porque el resultado de un test contiene una gran cantidad de información organizada de manera que puede desconcertar a primera vista. Con un ejemplo se pueden explorar los conceptos básicos. Más adelante se dedica un capítulo a los tests de hipótesis; a continuación consideraremos por encima un ejemplo sencillo.

En las variables `mEng` y `mEsp` tenemos las notas de Bioestadística correspondientes a los grupos de inglés y español. ¿Son diferentes las medias?

```
marks = read.csv("notas_E_S.csv", sep=";")
mEng=marks$English
mEsp=marks$Spanish
MM=c(mEng,mEsp)
GG=c(rep('English',length(mEng)),rep('Spanish',length(mEsp)))
ms=data.frame(MM,GG)
```



A continuación se ejecuta un test para la igualdad de varianzas, y luego un test t de Student. La tilde debe interpretarse como *versus* (según, en función de).

```
resultado1=var.test(ms$MM~ms$GG)
resultado1
```

```
##
## F test to compare two variances
##
## data: ms$MM by ms$GG
## F = 1.7648, num df = 54, denom df = 89, p-value = 0.01734
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  1.105199 2.900217
## sample estimates:
## ratio of variances
##          1.764766
```

El resultado de un test suele contener: una *descripción*, en la que se incluye el método; el *nombre de los datos*; el *estadístico*; los *parámetros* (en este caso: los grados de libertad); el *p-valor*, la *hipótesis alternativa*; el *intervalo de confianza*; y la *estimación*.

Como se ve, las varianzas no pueden considerarse iguales. Eso debe indicarse como argumento del test t de Student.


```
resultado2=t.test(ms$MM~ms$GG,var.equal=FALSE)
resultado2
```

```
##
## Welch Two Sample t-test
##
## data: ms$MM by ms$GG
## t = -0.93337, df = 91.236, p-value = 0.3531
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -1.0648215  0.3840134
## sample estimates:
## mean in group English mean in group Spanish
##          6.161818          6.502222
```

Como ejercicio, intérpretese `resultado2` identificando todos los elementos. El resultado de un test es un objeto a cuyos componentes se accede mediante `$`. Por ejemplo,

```
resultado2$conf.int
```

```
## [1] -1.0648215  0.3840134
## attr("conf.level")
## [1] 0.95
```

Otros tests usados frecuentemente son `prop.test`, `binom.test` o `chisq.test`. El ANOVA y la regresión lineal se pueden considerar también tests con características específicas.

Control de flujo

Al escribir programas, con frecuencia es necesario **ramificar** las secuencias de comandos (*si se cumple una condición* ejecutar ciertas acciones). La palabra clave en este caso es `if`. O bien repetir determinados comandos un cierto número de veces, es decir, programar un **bucle**. Estos bucles pueden ejecutarse un número determinado de veces (bucle `for`, también llamado bucle con contador), o bien un número indeterminado de veces, mientras se cumpla una condición (bucle `while` o con condición).

La sintaxis, resumida, es así:

- `if (condición) expresión`
- `for (variable in secuencia) expresión`
- `while (condición) expresión`

Para las condiciones se emplean los operadores `<`, `>`, `>=`, `<=`, `==` y `!=`. Es importante recalcar que la pregunta *¿es A igual a B?* se implementa con `A==B`; la expresión `A=B` le asignaría a A el valor de B. La condición debe poder evaluarse dando un resultado lógico. La expresión puede ser un comando único o una secuencia de comandos; en este caso, deben escribirse entre llaves. Por ejemplo:

```
for(contador in 1:10){
  if (contador%%2==0){ # %% es el resto de la división
    print(contador^2) # solo muestra los cuadrados de los números pares
  } # para una sola instrucción, no harían falta llaves
}
```

```
## [1] 4
## [1] 16
## [1] 36
## [1] 64
## [1] 100
```

En los bucles con condición se puede usar también un contador, pero en este caso su gestión es responsabilidad del programador (ir aumentándolo, por poner un caso):

```
conta <- 1
while(conta<=3){
  print(conta)    # solo muestra los cuadrados de los números pares
  conta <- conta+1
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

Base R

Cheat Sheet

Getting Help

Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

More about an object

str(iris)

Return a summary of an object's structure.

class(iris)

Find the class an object belongs to.

Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

Load the package into the session, making all its functions available to use.

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

Working Directory

getwd()

Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')

Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)

Return x sorted.

table(x)

See counts of values.

rev(x)

Return x reversed.

unique(x)

See unique values.

Selecting Vector Elements

By Position

x[4]

The fourth element.

x[-4]

All but the fourth.

x[2:4]

Elements two to four.

x[-(2:4)]

All elements except two to four.

x[c(1, 5)]

Elements one and five.

By Value

x[x == 10]

Elements which are equal to 10.

x[x < 0]

All elements less than zero.

x[x %in% c(1, 2, 5)]

Elements in the set 1, 2, 5.

Named Vectors

x['apple']

Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
  Do something
}
```

Example

```
for (i in 1:4){
  j <- i + 10
  print(j)
}
```

While Loop

```
while (condition){
  Do something
}
```

Example

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

Functions

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

Example

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Output	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of readtable/write table.
load('file.Rdata')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions	a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	a < b	Less than	a <= b	Less than or equal to	is.na(a)	Is missing
	a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null				

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

ls() List all variables in the environment.

rm(x) Remove x from the environment.

rm(list = ls()) Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

m <- matrix(x, nrow = 3, ncol = 3)
Create a matrix from x.

 m[2,]	- Select a row	 m[, 1]	- Select a column	 m[2, 3]	- Select an element
Transpose t(m)			Matrix Multiplication solve(m, n)		
Find x in: m * x = n					

Lists

l <- list(x = 1:5, y = c('a', 'b'))
A list is a collection of elements which can be of different types.

l[[2]]	l[1]	l\$x	l[\"y\"]
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** package.

df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting






df\$x	df[[2]]
	

Understanding a data frame

View(df) See the full data frame.

head(df) See the first 6 rows.

Matrix subsetting

df[, 2]	nrow(df)	cbind	- Bind columns.
	Number of rows.		
df[2,]	ncol(df)	rbind	- Bind rows.
	Number of columns.		
df[2, 2]	dim(df)		
	Number of columns and rows.		

Strings

Also see the **stringr** package.

paste(x, y, sep = '')	Join multiple vectors together.
paste(x, collapse = '')	Join elements of a vector together.
grep(pattern, x)	Find regular expression matches in x.
gsub(pattern, replace, x)	Replace matches in x with a string.
toupper(x)	Convert to uppercase.
tolower(x)	Convert to lowercase.
nchar(x)	Number of characters in a string.

Factors

factor(x) Turn a vector into a factor. Can set the levels of the factor and the order.

cut(x, breaks = 4) Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

lm(y ~ x, data=df)	Linear model.	prop.test	Test for a difference between proportions.
glm(y ~ x, data=df)	Generalised linear model.	t.test(x, y)	Perform a t-test for difference between means.
summary	Get more detailed information out a model.	pairwise.t.test	Perform a t-test for paired data.
		aov	Analysis of variance.

Distributions

Random Variables	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm
Poisson	rpois	dpois	ppois
Binomial	rbinom	dbinom	pnbinom
Uniform	runif	dunif	punif

Plotting

Also see the **ggplot2** package.

plot(x)	Values of x in order.	plot(x, y)	Values of x against y.	hist(x)	Histogram of x.
----------------	-----------------------	-------------------	------------------------	----------------	-----------------

Dates

See the **lubridate** package.

Capítulo 3

Lectura y escritura de datos con R

Es posible importar datos a R desde cualquier archivo de texto, desde Excel o desde otros programas de estadística –como SPSS, STATA o SAS– y lo que aún es más útil cuando se manejan muchos datos, R se puede comunicar con bases de datos como Access o MySQL.

Funciones principales para lectura de datos en R

Estas son las funciones principales en R básico para leer datos:

- `read.table`, para archivos en formato de texto. El más flexible. los demás son casos particulares de este.
- `read.csv` y `read.csv2` Para archivos en formato de texto, con los datos separados por puntos o punto y coma.
- `read.delim` y `read.delim2`. Para datos separados por tabuladores.
- `read.fwf` para datos de ancho fijo.
- `readLines`, leer línea a línea. Útil cuando hay dificultades para leer un archivo, o si este no tiene una estructura definida.
- `load`, para datos binarios guardados en formato `.Rdata` (inverso de `save`)
- `readRDS` para datos guardados en formato RDS.

`read.table()`

La función `read.table` Es la función de R-base más importante para leer datos. Recomendamos leer la entrada correspondiente en la ayuda. Tiene los siguientes argumentos principales:

- `file`, El nombre de un fichero (o conexión – se verá más adelante)
- `header`, Valor lógico (toma valores TRUE o FALSE) que indica si los datos tienen encabezamiento.
- `sep`, caracteres que separan los datos
- `dec`, carácter de puntuación para los decimales (“.”, ó “,”)
- `encoding`, cómo están codificados los caracteres (opcional).
- `colClasses`, Un vector de caracteres que indica la clase de cada columna.
- `nrows`, número de filas en el conjunto de datos.
- `strip.white`, lógica, si TRUE quita los espacios en blanco por delante y detrás de las cadenas de caracteres (en las variables numéricas lo hace por defecto).
- `comment.char`, caracteres que indican los comentarios.
- `skip`, número de líneas que saltar antes del comienzo de la lectura de los datos.

- `stringsAsFactors`, lógico. ¿Una columna de caracteres debe ser transformado a factor automáticamente? Por defecto toma el valor `TRUE` pero es mejor especificar la variable como factor después de leerla. El paquete `readr` siempre mantiene el texto como cadenas de caracteres.

`write.table()`

Si queremos que nuestros datos sean legibles tenemos que tener cuidado con los nombres de fila y nombres de columna.

- `write.table(datos,file="misdatos.txt")`
- `write.table(datos,file="misdatos.txt", quote=FALSE)`
- `write.table(datos,file="misdatos.txt", quote=FALSE,row.names=FALSE)`

Ejemplo

Es fácil es importar datos desde un fichero de texto (.txt, .csv) donde tenemos los datos en columnas y como encabezamiento de cada columna el nombre de la variable correspondiente. Esto ocurrirá por ejemplo cuando hayamos guardado con `write.table()`

Vamos a ver como importar datos guardados desde una hoja de Excel y más en general desde un archivo .csv (*comma separated values*)

Para importar un *data frame* desde Excel nuestra hoja de cálculo debe a ser posible:

- tener los datos de cada variable en una columna,
- los nombres de la variable en la primera fila
- los datos en las filas siguientes.

Hemos de guardar el fichero Excel como un fichero csv (primero usar guardar como ... y despues seleccionar en guardar como tipo CSV (delimitado por comas) (*.csv)

Luego se puede leer con el formato adecuado entre los descritos arriba (en el caso de Excel en castellano, tenemos los decimales separados por comas y por tanto necesitamos las opciones `sep=";", dec=","`)

En general podemos importar datos separados por comas (extensión .csv) guardados desde Excel en español con:

```
misdatos <- read.table("mifichero.csv",
                      header=TRUE, sep=";", dec=",")
```

Esto corresponde a datos separados por punto y coma con decimales separados por comas (que es como exporta Excel los ficheros de datos “en español”)

En el archivo Excel `gimnasio.xlsx` se tienen los datos de peso, altura y ejercicio de 20 alumnos. En la columna *ejercicio* hemos usado los valores 1 para Nunca, 2 para Ocasional y 3 para Frecuente.

Una vez guardado como .csv desde Excel lo leemos con:

```
gimnasio <- read.csv("../DATA/gimnasio.csv",
                    header=TRUE, sep=";", dec=",")
```

Las ordenes `read.csv` y `read.csv2` son equivalentes a `read.table` con `sep=","`, `dec="."` en el primer caso y `sep=";", dec=","` en el segundo.

Optimizar la lectura

Si los documentos son muy grandes, especificar la clase de las columnas puede disminuir considerablemente el tiempo de lectura. Una manera de hacerlo es leer las primeras líneas para obtener los tipos de variables y después especificarlas con `colClasses`. Si nuestra base de datos no tiene comentarios, incluir `comment.char =` también aumentará la velocidad de lectura.

```
minidf <- read.table("datatable.txt", nrows = 20)
classes <- sapply(minidf, class)
df <- read.table("datatable.txt", colClasses = classes)
```

Hay que tener en cuenta que R leerá el fichero en la memoria RAM así que hemos de tener cuidado de no hacerle leer ficheros enteros que ocupen más que nuestra memoria RAM.

Calculando la memoria necesaria

Supóngase que un *data frame* con 1.500.000 filas y 120 columnas, todas numéricas. ¿Cuánta memoria se requiere para almacenar este *data frame*? En los sistemas operativos recientes los números en doble precisión usan 64 bits (8 bytes). Podemos hacer el siguiente cálculo:

$$\begin{aligned} 1.500.000 \times 120 \times 8 \text{ bytes/numeric} &= 1.440.000.000 \text{ bytes} \\ &= 1.440.000.000 / 2^{20} \text{ bytes/MB} \\ &= 1.373,29 \text{ MB} \\ &= 1,34 \text{ GB} \end{aligned}$$

Así que se requerirán aproximadamente 1.34 GB de RAM. Has de tener en cuenta:

- qué otros programas están usando RAM en el momento de la lectura
- qué otros objetos en el *Environment* de R están usando RAM.

Si intentamos leer un archivo y no tenemos suficiente RAM es posible que R se congele.

Escritura de archivos.

Para la escritura de archivos tenemos la contrapartida de cada uno de los métodos de lectura.

- `write.table`
- `write.csv`, `write.csv2`
- `save(x,df, file="nombrefichero.Rdata")` guarda en binario varios objetos de R de cualquier tipo (vector, data frame, lista, ...) se lee con `load("nombrefichero.Rdata")`
- `saveRDS(objeto)` como `save` pero para un único objeto. Para leerlo hemos de asignarlo pues al contrario que `save` no guarda los nombres `df <- readRDS(df.RDS)`

Leer archivos de texto con readr.

El paquete `readr` desarrollado por Hadley Wickam facilita la lectura de archivos para después utilizarlos dentro del `tidyverse`. Tiene alternativas a `read.table()`, `read.csv()` y `read.csv2()` llamadas `read_table()`, `read_csv()`, y `read_csv2()`. Las funciones de `readr` son mucho más rápidas que sus alternativas en R base, mientras la sintaxis es similar. La salida de la lectura es un `tibble` que es una modificación del `data.frame` con algunas ventajas sobre este.

El modo de lectura más genérico en `readr` es `read_delim`. Recomendamos leer la ayuda de `read_delim` para ver todas las opciones. En particular, las opciones relacionadas con el idioma (cómo están separados los decimales, el *encoding*, etc.) se especifican en la opción `locale`.

Leer archivos con el menú `import dataset`.

En la pestaña de `Environment` nos encontramos el botón *Import Dataset* que nos propone un interfaz visual para la lectura de datos. Es importante que al final de la lectura copiemos el código que aparece en nuestro script de R para que la lectura realizada sea reproducible. Vemos cómo podemos leer archivos no solo de ficheros de texto y archivos de Excel, sino también de otros programas estadísticos, utilizando el paquete `haven`.

Como ejemplo práctico, vamos a leer el archivo `gimnasio.csv` con este menú.

Conexiones

La lectura de datos se realiza mediante interfaces de conexión. Las conexiones más usuales se realizan a archivos, pero también pueden realizarse conexiones de otro tipo.

- `file`, abre una conexión a un archivo.
- `gzfile` (`bzfile`), abre una conexión a un archivo comprimido con `gzip` (`bzip2`).
- `url` abre una conexión a una página web.

Las conexiones son herramientas que nos permiten navegar entre archivos y otros objetos externos.

Conexiones a archivos

La función `file()` tiene los dos primeros argumentos comunes con otras conexiones: `description` es el nombre del fichero, y `open` es un código que indica en que modo se establecerá la conexión. Puede tomar las siguientes opciones:

- `"r"` abre un archivo en modo lectura.
- `"w"` abre un archivo para escribir (e inicia un nuevo archivo)
- `"a"` abre un archivo para añadir contenido (`append`)
- `"rb"`, `"wb"`, `"ab"` los tipos anteriores en modo binario (`Windows`)

En general las funciones de lectura que hemos visto se encargan de abrir y cerrar las conexiones.

```
## Creamos una conexión a 'mifichero.csv'
con <- file("mifichero.csv")
## Abrimos la conexión a 'mifichero.csv' en modo lectura
open(con, "r")
## Leemos de la conexión
misdatos <- read.csv(con)
## Cerramos la conexión
close(con)
```

Es lo mismo que

```
misdatos <- read.csv("mifichero.csv")
```


Lectura de líneas de una página web.

Veamos cómo usar una conexión junto con `readLines` para leer las primeras líneas de una página web.

```
## Abre una conexión a una URL para lectura
con <- url("https://ebird.org/home", "r")
## Leemos la página web
mitexto <- readLines(con)

## Warning in readLines(con): incomplete final line found on 'https://
## ebird.org/home'

## Escribimos las primeras líneas.
head(mitexto)

## [1] "<!-- ebird/recruitment.jsp -->"
## [2] ""
## [3] ""
## [4] "<!DOCTYPE html>"
## [5] "<html lang=\"en\" class=\"no-js\" dir=\"ltr\">"
## [6] ""

close(con)
```

R no puede tener más de una conexión abierta simultáneamente, así que hay que cerrar siempre las conexiones si no queremos tener efectos inesperados.

Otros tipos de datos.

Hay paquetes especializados para lectura de otras fuentes.

- `haven` SPSS, Stata y SAS.
- `readxl` archivos de Excel.
- ODBC Moderno paquete para bases de datos que interactúa con `dplyr`
- `jsonlite` JSON.
- `xml2` xml
- `httr` APIs web.
- `rvest` HTML para web scraping.
- `rio` R input output es un paquete que se autodenomina la navaja suiza de la lectura escritura de datos en R. Lee casi cualquier formato.

También determinadas bases de datos dan en sus propios paquetes funciones de lectura para los datos. Recomendamos examinar `Ropensci.org` para muchas de estas apis. Por ejemplo los paquetes `rebird` y `auk` para la conexión con la base de datos de observaciones de pájaros de `ebird`, o las múltiples herramientas relacionadas con la bioinformática que propone el metapaquete `bioconductor`.

Capítulo 4

Manipulación de bases de datos

Preliminares

- Modificar datos ya introducidos.
- Manejo de dplyr.

Tipos de Datos y asignaciones (recordatorio).

Objetos

Objetos atómicos

R tiene cinco clases básicas de variables:

- `character`
- `numeric` (real numbers)
- `integer`
- `complex`
- `logical` (True/False)

Los objetos en R son vectores o listas.

- Un vector solo puede contener objetos de la misma clase
- Una lista puede obtener objetos de distintas clases.

Números

- los números en R se suelen tratar como objetos de tipo `numeric`(doble precisión)
- Si se quiere trabajar específicamente con enteros añadiremos el sufijo `L` al número. Por ejemplo `1L` es un número de tipo `integer`.
- Hay un número especial `Inf` que representa infinito por ejemplo como resultado de `1/0`.
- El valor `NaN` representa un valor indefinido (`0/0`)

Atributos

Los objetos de R tienen atributos:

- names y dimnames
- dimensions (matrices, arrays)
- class
- length
- otros definidos por el usuario

Se pueden ver los atributos de un objeto usando `attributes()`

Asignaciones

Para hacer asignaciones con R se usa `<-` (también se puede usar `=` pero no se recomienda)

```
x <- 1 # Notar que no sale nada en la consola
print(x) # nos muestra el resultado almacenado.
```

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

```
x # nos muestra el resultado almacenado.
```

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

```
mensaje <- "Hola mundo"
```

```
y <- rnorm(50) #generamos 50 datos normales estándar
```

```
y
```

```
## [1] 0.01167439 0.73077814 0.73151128 -0.16208726 -0.76072621
## [6] 1.25513802 -0.61483446 -1.27680067 -0.79399872 0.21873722
## [11] -2.17878838 1.52966503 1.21448275 0.49066355 -0.77971823
## [16] 0.87887794 -0.54128166 -3.72613232 -0.53551983 0.72482418
## [21] -1.22292411 -0.06006843 0.74334262 -0.27070413 -1.13246122
## [26] -0.96841541 2.88629990 2.22670581 -1.74187542 0.53028209
## [31] 0.11233130 1.82143337 -0.98242064 0.79606317 -1.29459092
## [36] -0.48911339 1.22470902 -0.61892794 0.48164580 -0.72082651
## [41] -0.29361737 0.45421914 -1.45038313 -0.85554037 0.54015702
## [46] 0.80034126 0.54205117 -0.07437196 0.78585392 -0.36124548
```

Vectores

Sucesiones

```
x <- 1:10
```

```
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
y <- seq( from = -1, to = 1, by = 0.25)
```

```
y
```

```
## [1] -1.00 -0.75 -0.50 -0.25  0.00  0.25  0.50  0.75  1.00
```

Vectores

Usamos la función `c()` para crear vectores.

```
x <- c(0.5, 0.6)      # numeric
x <- c(TRUE, FALSE)  # logical
x <- c(T, F)         # logical
x <- c("a", "b", "c") # character
x <- 1:5             # integer
x <- c(1+0i, 1+4i)   # complex
```

Podemos inicializar los vectores con `vector()` o con `tipo()`

```
x <- vector("numeric", length=5)
x
y <- numeric(5)
y
```

La longitud de un vector se obtiene con la función `length()`

Matrices

Matrices son vectores con un atributo de dimensión. La dimensión es un vector de enteros de longitud 2 (`nrow`, `ncol`). Se construyen *por columnas*

```
m <- matrix( 1:6, nrow = 2, ncol = 3 )
m
```

```
##      [,1] [,2] [,3]
## [1,]   1   3   5
## [2,]   2   4   6
```

```
dim(m)
```

```
## [1] 2 3
```

#EJERCICIO: Usa `attributes()` para ver los atributos de esta matriz.

`cbind` y `rbind`

Se pueden crear matrices uniendo vectores por filas o columnas.

```
x <- 1:3
y <- 4:6
cbind(x, y)
```

```
##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
```

```
rbind(x,y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y    4    5    6
```

Listas

Las listas son un tipo especial de vectores donde se pueden mezclar objetos de distintas clases y de distinta longitud.

```
x <- list(1:3, c("a","b"), TRUE, 1L)
```

```
x
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1
```

Data Frames

El tipo de objeto con el que más vamos a trabajar. Son listas donde todos los objetos tienen la misma longitud.

```
df <- data.frame( x = c(2L,3L,1L), y = c(1.4, 1.5, 1.2), z = c("a", "b", "c") )
```

```
df
```

```
##   x   y z
## 1 2 1.4 a
## 2 3 1.5 b
## 3 1 1.2 c
```

Factores

Los factores serán de especial interés en modelización. Se especifican unos niveles o valores posibles que puede tomar la variable. Si no especificamos los niveles tomará los valores distintos ordenados (caso de ser texto) por orden alfabético.

```
x <- factor(c("sí", "no", "no", "sí", "no"))
```

```
x
```

```
## [1] sí no no sí no
## Levels: no sí
```

```
table(x)
```

```
## x
## no sí
## 3 2
```

Cambiar el orden de los niveles de un factor

Será muy útil en modelos lineales donde queremos tener un nivel de referencia.

```
x <- factor(x, levels = c("sí", "no"))
```

Niveles y etiquetas de un factor.

Es posible codificar los niveles con enteros a los que se asignan etiquetas. codificamos con 1 hombre y 2 mujer

```
sexo <- c(1,1,2,1,2,2,2)
sexo <- factor(sexo, levels = 1:2,
               labels = c("hombre", "mujer"))
```

Subconjuntos de datos

Operadores [], [[]] y \$

Hay distintos operadores que se pueden usar para extraer subconjuntos de objetos en R.

- [siempre devuelve un objeto de la misma clase (con una excepción)
- [[se usa para extraer elementos de una lista o un data frame. Se usa para extraer un único elemento de una lista o data frame. El objeto devuelto no necesariamente es una lista o data frame.
- \$ se usa para extraer elementos de una lista o data frame por nombre. Se comporta de manera análoga a [[

Ejemplos de subconjuntos

```
set.seed(123)
x <- round(rnorm(10),1)

x

## [1] -0.6 -0.2  1.6  0.1  0.1  1.7  0.5 -1.3 -0.7 -0.4
x[2]

## [1] -0.2
x[1:4]

## [1] -0.6 -0.2  1.6  0.1
```

```
x[c(1,3)]
```

```
## [1] -0.6  1.6
```

Subconjuntos por condicion.

```
x
```

```
## [1] -0.6 -0.2  1.6  0.1  0.1  1.7  0.5 -1.3 -0.7 -0.4
```

```
condicion <- x < 0
```

```
condicion
```

```
## [1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
```

```
x[condicion]
```

```
## [1] -0.6 -0.2 -1.3 -0.7 -0.4
```

```
x[x > 0]
```

```
## [1] 1.6 0.1 0.1 1.7 0.5
```

Subconjuntos de matrices

Elementos.

```
x <- matrix(1:6,2,3)
```

```
x
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    3    5
```

```
## [2,]    2    4    6
```

```
x[1,2]
```

```
## [1] 3
```

```
x[2,1]
```

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

Filas y columnas.

```
x <- matrix(1:6,2,3)
```

```
x
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    3    5
```

```
## [2,]    2    4    6
```

```
x[1, ]
```

```
## [1] 1 3 5
```

```
x[, 3]
```



```
## [1] 5 6
```

Subconjuntos de data.frames

```
df <- data.frame(x = c(1, 3, 4), y = c(2, 2, 5))
```

```
condicion <- df$x > 2
```

```
df[condicion, ]
```

```
##   x y
```

```
## 2 3 2
```

```
## 3 4 5
```

Subconjuntos de listas

```
x <- list(A=1:3, B=c("e","pi"), C=TRUE, D=rnorm(10))
```

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

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

```
x <- list(A=1:3, B=c("e","pi"), C=TRUE, D=rnorm(10))
```

```
x[1]
```

```
## $A
```

```
## [1] 1 2 3
```

```
x[[1]]
```

```
## [1] 1 2 3
```

```
# EJERCICIO intenta calcular con mean(),  
#la media de x[1], de x[[1]] y de x$A
```

Valores perdidos (NA)

Los valores perdidos se denominan NA

- `is.na()` se usa para ver si los objetos son NA
- `is.nan()` se usa para ver si los objetos son NaN
- NA tiene su clase puede ser NA entero, NA character, etc.
- Un NaN es también NA pero el recíproco no es cierto.

```
x <- c(1, 2, NA, 10, 3)
```

```
is.na(x)
```

```
## [1] FALSE FALSE TRUE FALSE FALSE
```

```
is.nan(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```
x <- c(1, 2, NaN, NA, 4)
is.na(x)

## [1] FALSE FALSE TRUE TRUE FALSE
is.nan(x)

## [1] FALSE FALSE TRUE FALSE FALSE
```

Eliminar NA

Con frecuencia deberemos quitar datos NA

```
x <- c(1, 2, NA, 4, NA, 5)
datosNA <- is.na(x)
datosNA

## [1] FALSE FALSE TRUE FALSE TRUE FALSE
x <- x[!datosNA]
```

Veamos como quitar todas las filas de un data.frame que tienen NA

```
df <- data.frame(x = c(2,NA,1,5), y=c(1,2,4,NA))
condicionsinNA <- complete.cases(df)
df[condicionsinNA, ]

##   x y
## 1 2 1
## 3 1 4
```

Introducción al Tidyverse, dplyr y tidyr

Podemos cargar todos los paquetes del tidyverse a la vez con

```
## -- Attaching packages ----- tidyverse 1.2.1 --

## v ggplot2 3.2.1    v purrr    0.3.2
## v tibble  2.1.3    v dplyr    0.8.3
## v tidyr   0.8.3    v stringr  1.4.0
## v readr   1.3.1    v forcats  0.4.0

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

dplyr o base R.

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data. (Dasu and Johnson, 2003)

Antes de cualquier análisis o gráfico necesitamos:

1. *Manipular* data frames, filtros, sumarios, y calculos por grupos.

2. *Limpiar y hacer ordenados* nuestros datos (tidy data).

Hay dos maneras de hacer esto en R

- Usar las funciones de R base para todo lo anterior sin emplear el `tidyverse` que usa sintaxis distinta que R.
- Usar cuando sea posible las herramientas del `tidyverse` porque son más fáciles de utilizar, y de interpretar que las de R base, y aceleran el proceso de limpieza de datos.

Veamos algunas de las herramientas del `tidyverse`

Datos de ejemplo

Trabajaremos con los datos de “Gapminder” que es un extracto de los datos de Gapminder.org. Para cada una de los 142 países, tenemos datos sobre la esperanza de vida, GDP per cápita y población cada cinco años desde 1952 hasta 2007. Podemos cargarlo con el paquete `gapminder` o con el archivo `gapminder-FiveYearData.csv`

```
gapminder <- read.csv("./DATA/gapminder-FiveYearData.csv",
  stringsAsFactors = TRUE)
head(gapminder)
```

```
##      country year      pop continent lifeExp gdpPercap
## 1 Afghanistan 1952  8425333      Asia  28.801  779.4453
## 2 Afghanistan 1957  9240934      Asia  30.332  820.8530
## 3 Afghanistan 1962 10267083      Asia  31.997  853.1007
## 4 Afghanistan 1967 11537966      Asia  34.020  836.1971
## 5 Afghanistan 1972 13079460      Asia  36.088  739.9811
## 6 Afghanistan 1977 14880372      Asia  38.438  786.1134
```

Manipulación de dataframes con dplyr

El paquete `dplyr` nos da muchas herramientas para manipular data frames que tiene una *gramática* bastante natural.

Cubriremos las 6 funciones más comunes así con el operador *pipe* (`%>%`) para combinarlas.

1. `select()`
2. `filter()`
3. `group_by()`
4. `summarize()`
5. `mutate()`
6. `arrange()`

Al cargar `tidyverse` ya hemos incluido `dplyr`, si no podríamos cargarlo con:

```
library(dplyr)
```

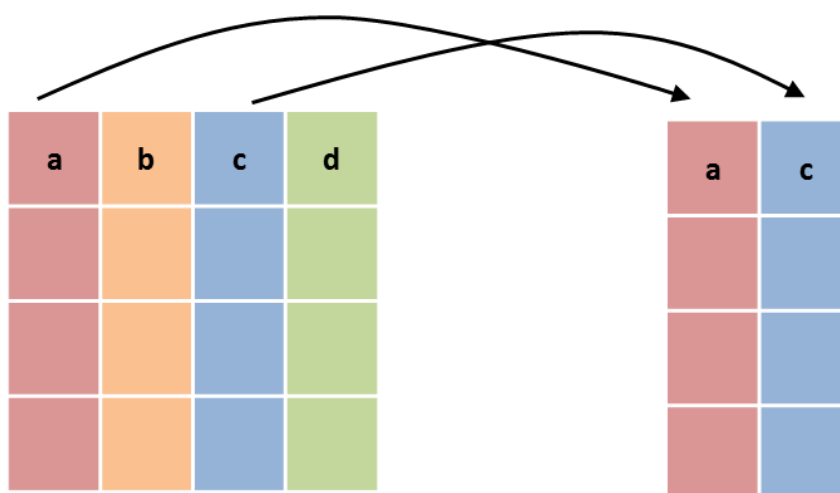
`dplyr select`

Tenemos un data.frame del que solo nos interesan unas variables. Podemos usar `select()` para quedarnos únicamente con las columnas de interés.

```
year_country_gdp <- select(gapminder, year, country, gdpPercap)
head(year_country_gdp)
```

```
##   year    country gdpPercap
## 1 1952 Afghanistan  779.4453
## 2 1957 Afghanistan  820.8530
## 3 1962 Afghanistan  853.1007
## 4 1967 Afghanistan  836.1971
## 5 1972 Afghanistan  739.9811
## 6 1977 Afghanistan  786.1134
```

select(data.frame, a, c)



Si miramos el contenido de `year_country_gdp`, veremos que solo están presentes las variables `year`, `country` y `gdpPercap`. Es equivalente a lo que podemos hacer con R base:

```
year_country_gdp <- gapminder[,c("year", "country", "gdpPercap")]
head(year_country_gdp)
```

```
##   year    country gdpPercap
## 1 1952 Afghanistan  779.4453
## 2 1957 Afghanistan  820.8530
## 3 1962 Afghanistan  853.1007
## 4 1967 Afghanistan  836.1971
## 5 1972 Afghanistan  739.9811
## 6 1977 Afghanistan  786.1134
```

`dplyr` facilita mucho la comprensión del código por el operador *pipe*.

pipes ‘%>%’

Las *pipes* toman el input a la izquierda del símbolo `%>%` y lo pasan como primer argumento a la función de la derecha.

Usemos una pipa para la asignación anterior.

```
year_country_gdp <- gapminder %>%  
  select(year, country, gdpPercap)
```

El `data.frame` de `gapminder` pasan por `%>%` a la función `select`. Así no hace falta especificar el `data` al aplicar `select()` porque lo hemos pasado por la *pipe*.

Filtros con dplyr

Supongamos que queremos centrar nuestro estudio en África. Podemos usar `select` y `filter` para seleccionar datos cuya variable `continent` tenga el valor `Africa`.

```
year_country_gdp_africa <- gapminder %>%  
  filter(continent == "Africa") %>%  
  select(year, country, gdpPercap)
```

Pasamos `gapminder` como primer argumento a `filter()` y el resultado como primer argumento a `select()`.

Notar que `select` y `filter` segmentan the data frame. La diferencia es que `select` extrae *columnas*, mientras que `filter` extrae *filas*.

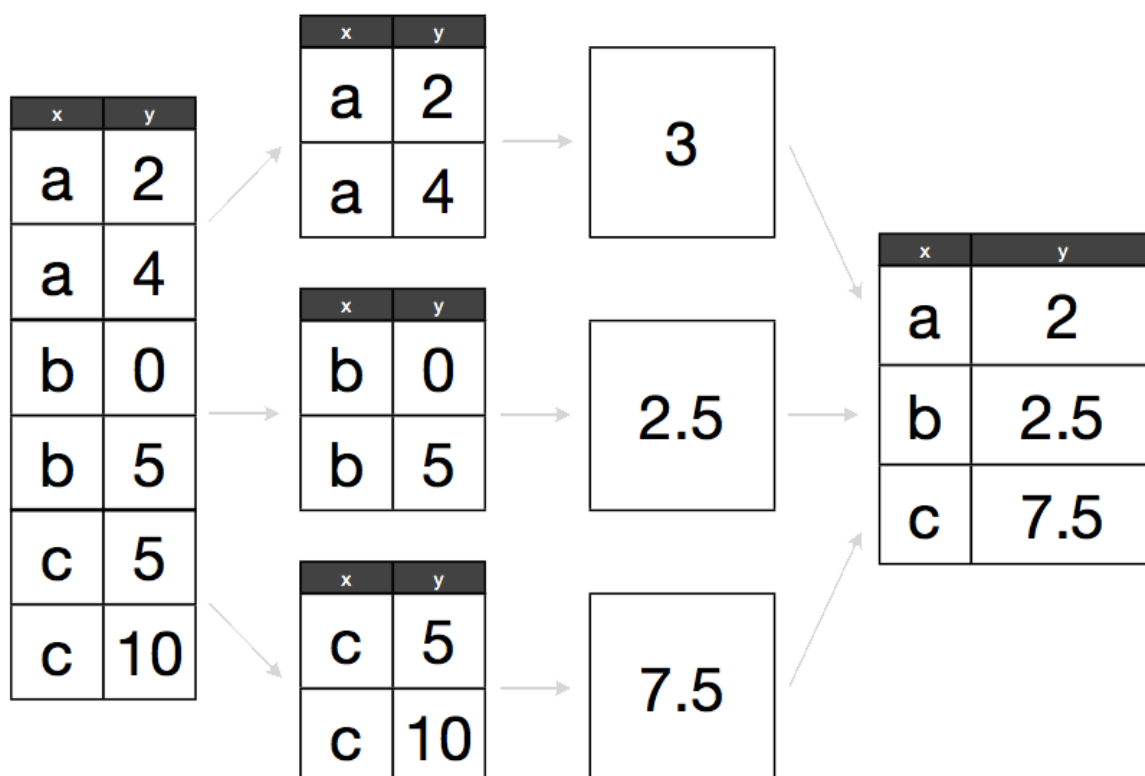
Nota: El orden de las operaciones es muy importante. Si hubiésemos usado ‘`select`’ primero, no hubiésemos tenido disponible la variable `continent` para usarla como argumento en `filter` ya que la hemos eliminado en el paso anterior.

Calculos en grupos con dplyr

Queremos calcular cual es el GPD medio en cada continente, y añadir los valores al `data.frame` original, esto es bastante complicado con base R pero muy fácil con `dplyr`.

split-apply-combine con dplyr

Hemos descrito un problema “split-apply-combine”:

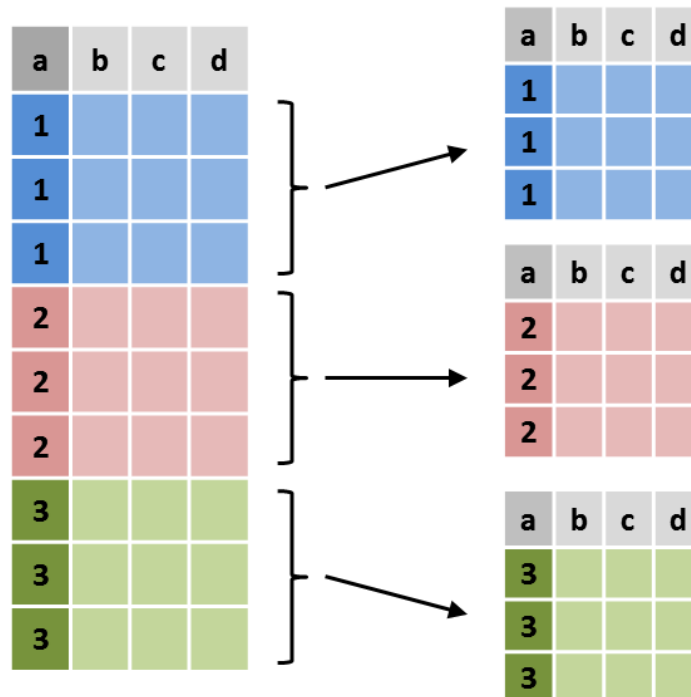


Queremos dividir *split* nuestros datos en grupos (en nuestro caso continentes), aplicar *apply* calculos en cada grupo, y finalmente combinar *combine* los resultados.

`dplyr group_by`

Hemos visto con los filtros con `filter()` nos pueden servir para seleccionar casos que cumplen ciertos criterios (por ejemplo: `continent == "Europe"`). La función agrupar por `group_by()`, nos separará en grupos como si hubiesemos filtrado por todos los posibles valores de continente con `filter()`.

Un data frame agrupado `grouped_df` se puede pensar como una lista `list` donde cada item en la lista es un `data.frame` que contiene solo las filas correspondientes a un valor particular de la variable de filtrado (en nuestro caso `continent`).

df %>% group_by(a)**summarize con dplyr**

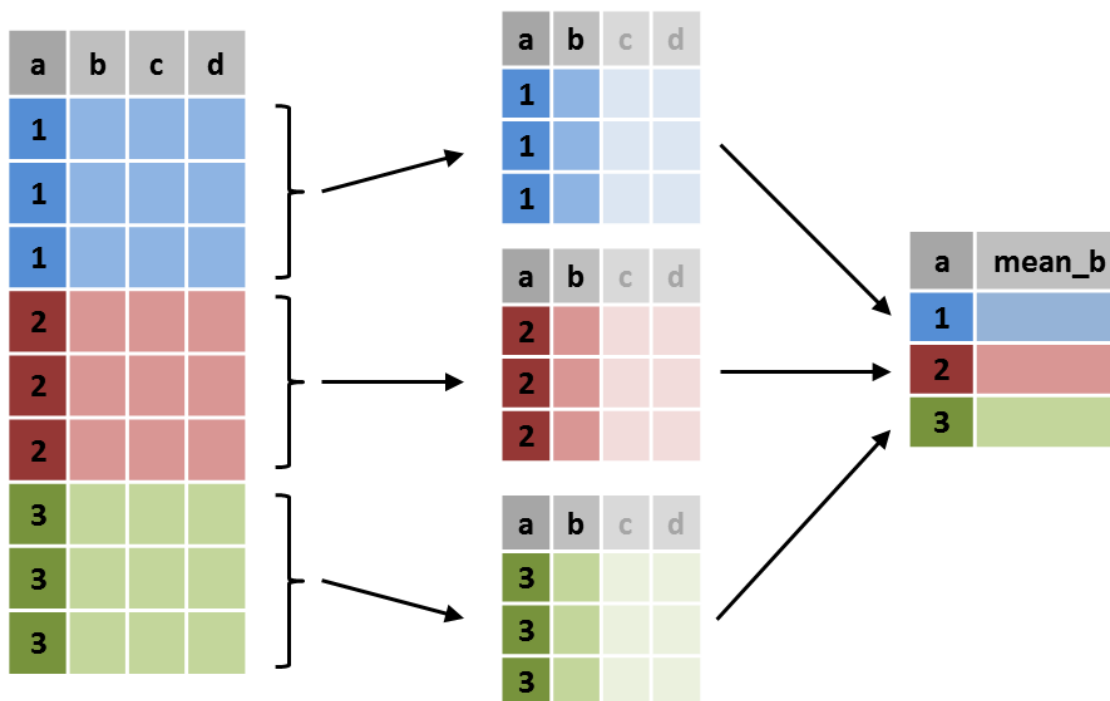
`group_by()` por si solo no es muy interesante. Si lo es combinado con la función `summarize()`. Esto nos permitirá crear nuevas variables empleando transformaciones de variables en cada uno de los data.frames que contienen información por continente.

En otras palabras con `group_by()` dividimos el dataframe original en partes, para después aplicar funciones a cada grupo (e.j. `mean()` o `sd()`) con `summarize()`. La salida es un nuevo dataframe con una fila por grupo.

```
gdp_bycontinents <- gapminder %>%
  group_by(continent) %>%
  summarize(mean_gdpPerCap = mean(gdpPerCap))
head(gdp_bycontinents)
```

```
## # A tibble: 5 x 2
##   continent mean_gdpPerCap
##   <fct>         <dbl>
## 1 Africa          2194.
## 2 Americas        7136.
## 3 Asia            7902.
## 4 Europe         14469.
## 5 Oceania        18622.
```

```
df %>% group_by(a) %>%
  summarize(mean_b=mean(b))
```



Hemos podido calcular así el GDP medio por continente. La función `group_by()` nos permite agrupar por varias variables. Hagámoslo por año y continente

```
gdp_bycontinents_byyear <- gapminder %>%
  group_by(continent, year) %>%
  summarize(mean_gdpPercap = mean(gdpPercap))
head(gdp_bycontinents_byyear)
```

```
## # A tibble: 6 x 3
## # Groups:   continent [1]
##   continent year mean_gdpPercap
##   <fct>      <int>      <dbl>
## 1 Africa    1952        1253.
## 2 Africa    1957        1385.
## 3 Africa    1962        1598.
## 4 Africa    1967        2050.
## 5 Africa    1972        2340.
## 6 Africa    1977        2586.
```

Podemos definir más de una variable con `summarize()`.

```
gdp_pop_bycontinents_byyear <- gapminder %>%
  group_by(continent, year) %>%
  summarize(mean_gdpPercap = mean(gdpPercap),
            sd_gdpPercap = sd(gdpPercap),
            mean_pop = mean(pop),
            sd_pop = sd(pop))
```



```
head(gdp_pop_bycontinents_byyear)
```

```
## # A tibble: 6 x 6
## # Groups:   continent [1]
##   continent year mean_gdpPercap sd_gdpPercap mean_pop sd_pop
##   <fct>      <int>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 Africa    1952        1253.        983.  4570010.  6317450.
## 2 Africa    1957        1385.       1135.  5093033.  7076042.
## 3 Africa    1962        1598.       1462.  5702247.  7957545.
## 4 Africa    1967        2050.       2848.  6447875.  8985505.
## 5 Africa    1972        2340.       3287.  7305376. 10130833.
## 6 Africa    1977        2586.       4142.  8328097. 11585184.
```

mutate con dplyr

La función `mutate()`, es similar a `summarize()` pero añade las nuevas variables al data frame de origen en lugar de crear uno nuevo. Muy útil para variables calculadas a partir de otras variables.

```
gapminder_with_extra_vars <- gapminder %>%
  group_by(continent, year) %>%
  mutate(mean_gdpPercap = mean(gdpPercap),
         sd_gdpPercap = sd(gdpPercap),
         mean_pop = mean(pop),
         sd_pop = sd(pop))
head(gapminder_with_extra_vars)
```

```
## # A tibble: 6 x 10
## # Groups:   continent, year [6]
##   country year pop continent lifeExp gdpPercap mean_gdpPercap
##   <fct> <int> <dbl> <fct>      <dbl>      <dbl>      <dbl>
## 1 Afghan~ 1952 8.43e6 Asia      28.8        779.        5195.
## 2 Afghan~ 1957 9.24e6 Asia      30.3        821.        5788.
## 3 Afghan~ 1962 1.03e7 Asia      32.0        853.        5729.
## 4 Afghan~ 1967 1.15e7 Asia      34.0        836.        5971.
## 5 Afghan~ 1972 1.31e7 Asia      36.1        740.        8187.
## 6 Afghan~ 1977 1.49e7 Asia      38.4        786.        7791.
## # ... with 3 more variables: sd_gdpPercap <dbl>, mean_pop <dbl>,
## #   sd_pop <dbl>
```

Podemos usar `mutate()` para crear variables nuevas antes de usar `summarize` (o después)

```
gdp_pop_bycontinents_byyear <- gapminder %>%
  mutate(gdp_billion = gdpPercap*pop/10^9) %>%
  group_by(continent, year) %>%
  summarize(mean_gdpPercap = mean(gdpPercap),
           sd_gdpPercap = sd(gdpPercap),
           mean_pop = mean(pop),
           sd_pop = sd(pop),
           mean_gdp_billion = mean(gdp_billion),
           sd_gdp_billion = sd(gdp_billion))
head(gdp_pop_bycontinents_byyear)
```

```
## # A tibble: 6 x 8
## # Groups:   continent [1]
##   continent year mean_gdpPercap sd_gdpPercap mean_pop sd_pop
##   <fct>      <int>      <dbl>         <dbl>      <dbl> <dbl>
## 1 Africa    1952         1253.          983. 4570010. 6.32e6
## 2 Africa    1957         1385.          1135. 5093033. 7.08e6
## 3 Africa    1962         1598.          1462. 5702247. 7.96e6
## 4 Africa    1967         2050.          2848. 6447875. 8.99e6
## 5 Africa    1972         2340.          3287. 7305376. 1.01e7
## 6 Africa    1977         2586.          4142. 8328097. 1.16e7
## # ... with 2 more variables: mean_gdp_billion <dbl>, sd_gdp_billion <dbl>
```

arrange (ordenar) con dplyr

Queremos ordenar las filas de un data.frame con respecto a los valores de una o varias columnas. Podemos usar la función `arrange()`. Por ejemplo ordenemos el data frame obtenido antes por año (reciente primero) y después por continente.

```
gapminder_with_extra_vars <- gapminder %>%
  group_by(continent, year) %>%
  mutate(mean_gdpPercap = mean(gdpPercap),
         sd_gdpPercap = sd(gdpPercap),
         mean_pop = mean(pop),
         sd_pop = sd(pop)) %>%
  arrange(desc(year), continent)
head(gapminder_with_extra_vars)
```

```
## # A tibble: 6 x 10
## # Groups:   continent, year [1]
##   country year   pop continent lifeExp gdpPercap mean_gdpPercap
##   <fct>   <int> <dbl> <fct>      <dbl>   <dbl>      <dbl>
## 1 Algeria  2007 3.33e7 Africa     72.3    6223.      3089.
## 2 Angola   2007 1.24e7 Africa     42.7    4797.      3089.
## 3 Benin    2007 8.08e6 Africa     56.7    1441.      3089.
## 4 Botswa~  2007 1.64e6 Africa     50.7   12570.      3089.
## 5 Burkin~  2007 1.43e7 Africa     52.3    1217.      3089.
## 6 Burundi  2007 8.39e6 Africa     49.6     430.      3089.
## # ... with 3 more variables: sd_gdpPercap <dbl>, mean_pop <dbl>,
## #   sd_pop <dbl>
```

Resumen de ventajas de dplyr

- Fácilmente interpretable.
- Pipes: nos permiten encadenar funciones especificando el proceso paso a paso.

Veamos el mismo calculo sin usar pipes:

```
gapminder_with_extra_vars <- arrange(
  mutate(
    group_by(gapminder, continent, year),
    mean_gdpPercap = mean(gdpPercap)
```

```
),
  desc(year), continent)
```

- Facilita las manipulaciones de tipo split-apply-combine.

Organizar datos de manera tidy.

Las dos propiedades fundamentales del tidy data son:

- 1) Cada columna es una variable.
- 2) Cada fila es una observación.

Se trabaja mejor con datos en este formato porque hay una manera consistente de referirse a variables (como nombres de columnas) y observaciones (como índice de filas) lo que facilita manipular, visualizar y modelar los datos.

Se recomienda leer el [artículo](#) de Hadley Wickham al respecto.

Tidying Data: Formatos ancho y largo.

“Tidy datasets are all alike but every messy dataset is messy in its own way.” –
Hadley Wickham

Tabular datasets can be arranged in many ways. For instance, consider the data below. Each data set displays information on heart rate observed in individuals across 3 different time periods. But the data are organized differently in each table.

```
ancho <- data.frame(
  nombre = c("Fulanez", "Menganez", "Zutanez"),
  v1 = c(67, 80, 64),
  v2 = c(56, 90, 50),
  v3 = c(70, 67, 101)
)
ancho
```

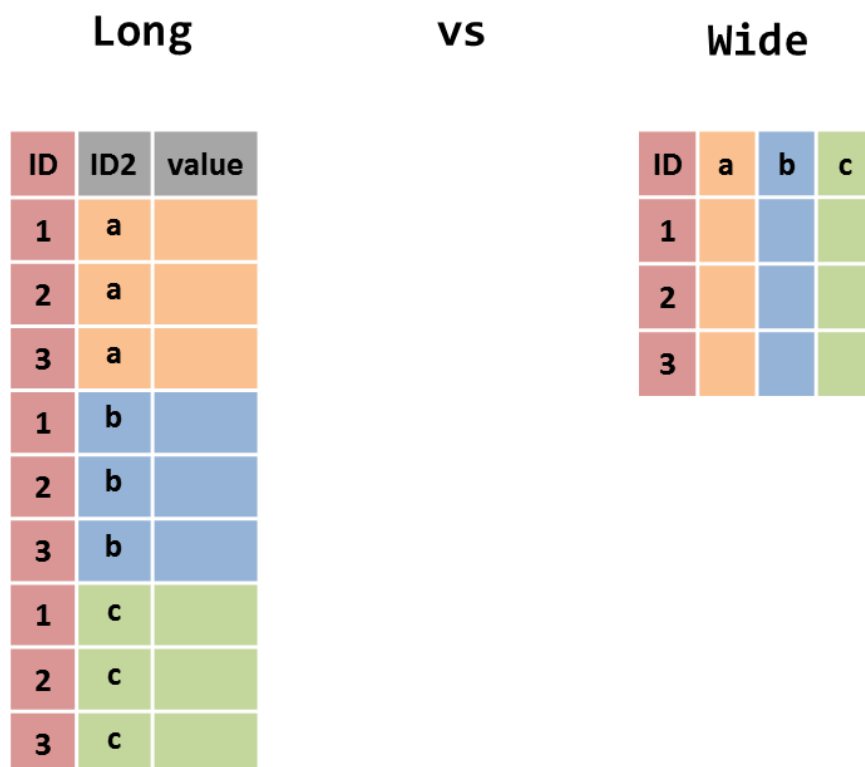
```
##      nombre v1 v2 v3
## 1 Fulanez 67 56 70
## 2 Menganez 80 90 67
## 3 Zutanez 64 50 101
```

```
largo <- data.frame(
  nombre = c("Fulanez", "Menganez", "Zutanez",
            "Fulanez", "Menganez", "Zutanez",
            "Fulanez", "Menganez", "Zutanez"),
  visita = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
  pulsaciones = c(67, 80, 64, 56, 90, 50, 70, 67, 10)
)
largo
```

```
##      nombre visita pulsaciones
## 1 Fulanez      1      67
## 2 Menganez      1      80
## 3 Zutanez      1      64
```

```
## 4 Fulanez      2      56
## 5 Menganez    2      90
## 6 Zutanez     2      50
## 7 Fulanez     3      70
## 8 Menganez    3      67
## 9 Zutanez     3      10
```

Pregunta:Cuál de los dos es el tidy data?



Muchas de las funciones de R prefieren el formato “largo” (pero no siempre es mejor)

Transformando Gapminder en tidy

Veamos la estructura original de gapminder:

```
head(gapminder)
```

```
##      country year      pop continent lifeExp gdpPercap
## 1 Afghanistan 1952  8425333      Asia   28.801   779.4453
## 2 Afghanistan 1957  9240934      Asia   30.332   820.8530
## 3 Afghanistan 1962 10267083      Asia   31.997   853.1007
## 4 Afghanistan 1967 11537966      Asia   34.020   836.1971
## 5 Afghanistan 1972 13079460      Asia   36.088   739.9811
## 6 Afghanistan 1977 14880372      Asia   38.438   786.1134
```

Pregunta: ¿Formato largo o ancho?

Respuesta: Ni uno ni otro. Tenemos 3 “variables ID” (continent, country, year) y 3 “variables observación” (pop, lifeExp, gdpPercap).

tidyr

Veamos como usar tidyr para transformar los datos como necesitamos.

```
# Si hemos cargado tidyverse no hace falta volver a cargar tidyr.
library(tidyr)
```

gather tidyr

Veamos una versión de los datos de gapminder en formato ancho.

```
gap_wide <- read.csv("./DATA/gapminder_wide.csv",
                    stringsAsFactors = FALSE)
head(gap_wide)
```

```
##   continent      country gdpPercap_1952 gdpPercap_1957 gdpPercap_1962
## 1   Africa      Algeria      2449.0082      3013.9760      2550.8169
## 2   Africa      Angola       3520.6103      3827.9405      4269.2767
## 3   Africa      Benin        1062.7522       959.6011       949.4991
## 4   Africa      Botswana      851.2411       918.2325       983.6540
## 5   Africa Burkina Faso     543.2552       617.1835       722.5120
## 6   Africa      Burundi      339.2965       379.5646       355.2032
##   gdpPercap_1967 gdpPercap_1972 gdpPercap_1977 gdpPercap_1982
## 1      3246.9918      4182.6638      4910.4168      5745.1602
## 2      5522.7764      5473.2880      3008.6474      2756.9537
## 3      1035.8314      1085.7969      1029.1613      1277.8976
## 4      1214.7093      2263.6111      3214.8578      4551.1421
## 5       794.8266       854.7360       743.3870       807.1986
## 6       412.9775       464.0995       556.1033       559.6032
##   gdpPercap_1987 gdpPercap_1992 gdpPercap_1997 gdpPercap_2002
## 1      5681.3585      5023.2166      4797.2951      5288.0404
## 2      2430.2083      2627.8457      2277.1409      2773.2873
## 3      1225.8560      1191.2077      1232.9753      1372.8779
## 4      6205.8839      7954.1116      8647.1423      11003.6051
## 5       912.0631       931.7528       946.2950      1037.6452
## 6       621.8188       631.6999       463.1151       446.4035
##   gdpPercap_2007 lifeExp_1952 lifeExp_1957 lifeExp_1962 lifeExp_1967
## 1      6223.3675       43.077       45.685       48.303       51.407
## 2      4797.2313       30.015       31.999       34.000       35.985
## 3      1441.2849       38.223       40.358       42.618       44.885
## 4     12569.8518       47.622       49.618       51.520       53.298
## 5      1217.0330       31.975       34.906       37.814       40.697
## 6       430.0707       39.031       40.533       42.045       43.548
##   lifeExp_1972 lifeExp_1977 lifeExp_1982 lifeExp_1987 lifeExp_1992
## 1       54.518       58.014       61.368       65.799       67.744
## 2       37.928       39.483       39.942       39.906       40.647
## 3       47.014       49.190       50.904       52.337       53.919
## 4       56.024       59.319       61.484       63.622       62.745
```

```
## 5      43.591      46.137      48.122      49.557      50.260
## 6      44.057      45.910      47.471      48.211      44.736
##  lifeExp_1997 lifeExp_2002 lifeExp_2007 pop_1952 pop_1957 pop_1962
## 1      69.152      70.994      72.301  9279525 10270856 11000948
## 2      40.963      41.003      42.731  4232095  4561361  4826015
## 3      54.777      54.406      56.728  1738315  1925173  2151895
## 4      52.556      46.634      50.728   442308   474639   512764
## 5      50.324      50.650      52.295  4469979  4713416  4919632
## 6      45.326      47.360      49.580  2445618  2667518  2961915
##  pop_1967 pop_1972 pop_1977 pop_1982 pop_1987 pop_1992 pop_1997 pop_2002
## 1 12760499 14760787 17152804 20033753 23254956 26298373 29072015 31287142
## 2  5247469  5894858  6162675  7016384  7874230  8735988  9875024 10866106
## 3  2427334  2761407  3168267  3641603  4243788  4981671  6066080  7026113
## 4  553541   619351   781472   970347  1151184  1342614  1536536  1630347
## 5  5127935  5433886  5889574  6634596  7586551  8878303 10352843 12251209
## 6  3330989  3529983  3834415  4580410  5126023  5809236  6121610  7021078
##  pop_2007
## 1 33333216
## 2 12420476
## 3  8078314
## 4  1639131
## 5 14326203
## 6  8390505
```

La función `gather()` tiene el mismo efecto que `stack()` en base R. Pone las observaciones en un único vector.

```
gap_long <- gap_wide %>%
  gather(obstype_year, obs_values, 3:38)
head(gap_long)
```

```
##  continent      country  obstype_year  obs_values
## 1  Africa      Algeria  gdpPercap_1952  2449.0082
## 2  Africa      Angola   gdpPercap_1952  3520.6103
## 3  Africa      Benin   gdpPercap_1952  1062.7522
## 4  Africa      Botswana gdpPercap_1952  851.2411
## 5  Africa  Burkina Faso gdpPercap_1952  543.2552
## 6  Africa      Burundi  gdpPercap_1952  339.2965
```

Hemos usado tres argumentos en `gather()`:

1. El nombre de la nueva columna que tendrá la variable ID (`obstype_year`),
2. El nombre de la nueva columna que tendrá la variable con todas las observaciones (`obs_value`),
3. los índices de las variables observación (`3:38`, columnas 3 a 38) que queremos unir en una variable. Notar que las columnas 1 y 2, son nuestras variables “ID”.

select en tidyr

Si hay muchas columnas o están nombradas de forma consistente, podemos usar información contenida en los nombres.

Podemos seleccionar variables con:

- índices de las variables
- nombres de las variables (sin comillas)
- x:z para seleccionar todas las variables entre x y z
- -y para *excluir* y
- `starts_with(x, ignore.case = TRUE)`: todos los nombres que comienzan por x
- `ends_with(x, ignore.case = TRUE)`: todos los nombres que terminan con x
- `contains(x, ignore.case = TRUE)`: todos los nombres que contienen x

Veamos un ejemplo

```
# con starts_with()
gap_long <- gap_wide %>%
  gather(obstype_year, obs_values, starts_with('pop'),
        starts_with('lifeExp'), starts_with('gdpPercap'))
head(gap_long)
```

```
##   continent      country obstype_year obs_values
## 1   Africa      Algeria      pop_1952    9279525
## 2   Africa      Angola       pop_1952    4232095
## 3   Africa      Benin        pop_1952    1738315
## 4   Africa      Botswana     pop_1952     442308
## 5   Africa Burkina Faso     pop_1952    4469979
## 6   Africa      Burundi     pop_1952    2445618
```

```
# con el operador -
gap_long <- gap_wide %>%
  gather(obstype_year, obs_values, -continent, -country)
head(gap_long)
```

```
##   continent      country  obstype_year obs_values
## 1   Africa      Algeria  gdpPercap_1952  2449.0082
## 2   Africa      Angola   gdpPercap_1952  3520.6103
## 3   Africa      Benin    gdpPercap_1952  1062.7522
## 4   Africa      Botswana gdpPercap_1952   851.2411
## 5   Africa Burkina Faso  gdpPercap_1952   543.2552
## 6   Africa      Burundi  gdpPercap_1952   339.2965
```

Trata de identificar las nuevas variables ID y observaciones.

Ejercicio: Usar `gather` para transformar la mini base de datos ancho en algo similar a largo.

separate en tidyr

En el dataset largo `obstype_year` contiene dos tipos de variables, el tipo de observación (`pop`, `lifeExp`, o `gdpPercap`) y el año `year`.

Podemos usar `separate()` para dividir en distintas variables:

```
gap_long_sep <- gap_long %>%
  separate(obstype_year, into = c('obs_type', 'year'), sep = "_") %>%
  mutate(year = as.integer(year))
head(gap_long_sep)
```

```
##   continent      country  obs_type year obs_values
## 1   Africa      Algeria  gdpPercap 1952  2449.0082
```

```
## 2 Africa Angola gdpPercap 1952 3520.6103
## 3 Africa Benin gdpPercap 1952 1062.7522
## 4 Africa Botswana gdpPercap 1952 851.2411
## 5 Africa Burkina Faso gdpPercap 1952 543.2552
## 6 Africa Burundi gdpPercap 1952 339.2965
```

spread en tidyr

Es el opuesto a `gather()`. Divide observaciones en variables según los valores de un factor. veamos como pasar del formato largo al inicial.

```
gap_medium <- gap_long_sep %>%
  spread(obs_type, obs_values)
head(gap_medium)
```

```
## continent country year gdpPercap lifeExp pop
## 1 Africa Algeria 1952 2449.008 43.077 9279525
## 2 Africa Algeria 1957 3013.976 45.685 10270856
## 3 Africa Algeria 1962 2550.817 48.303 11000948
## 4 Africa Algeria 1967 3246.992 51.407 12760499
## 5 Africa Algeria 1972 4182.664 54.518 14760787
## 6 Africa Algeria 1977 4910.417 58.014 17152804
```

Con unos pocos cambios conseguimos los datos originales.

```
gapminder <- read.csv("./DATA/gapminder-FiveYearData.csv")
head(gap_medium)
```

```
## continent country year gdpPercap lifeExp pop
## 1 Africa Algeria 1952 2449.008 43.077 9279525
## 2 Africa Algeria 1957 3013.976 45.685 10270856
## 3 Africa Algeria 1962 2550.817 48.303 11000948
## 4 Africa Algeria 1967 3246.992 51.407 12760499
## 5 Africa Algeria 1972 4182.664 54.518 14760787
## 6 Africa Algeria 1977 4910.417 58.014 17152804
```

```
head(gapminder)
```

```
## country year pop continent lifeExp gdpPercap
## 1 Afghanistan 1952 8425333 Asia 28.801 779.4453
## 2 Afghanistan 1957 9240934 Asia 30.332 820.8530
## 3 Afghanistan 1962 10267083 Asia 31.997 853.1007
## 4 Afghanistan 1967 11537966 Asia 34.020 836.1971
## 5 Afghanistan 1972 13079460 Asia 36.088 739.9811
## 6 Afghanistan 1977 14880372 Asia 38.438 786.1134
```

```
# ordenamos columnas
```

```
gap_medium <- gap_medium[,names(gapminder)]
head(gap_medium)
```

```
## country year pop continent lifeExp gdpPercap
## 1 Algeria 1952 9279525 Africa 43.077 2449.008
## 2 Algeria 1957 10270856 Africa 45.685 3013.976
## 3 Algeria 1962 11000948 Africa 48.303 2550.817
```



```
## 4 Algeria 1967 12760499 Africa 51.407 3246.992
## 5 Algeria 1972 14760787 Africa 54.518 4182.664
## 6 Algeria 1977 17152804 Africa 58.014 4910.417
```

```
# arrange por país continente y año.
gap_medium <- gap_medium %>%
  arrange(country,continent,year)
head(gap_medium)
```

```
##      country year      pop continent lifeExp gdpPercap
## 1 Afghanistan 1952 8425333      Asia 28.801 779.4453
## 2 Afghanistan 1957 9240934      Asia 30.332 820.8530
## 3 Afghanistan 1962 10267083     Asia 31.997 853.1007
## 4 Afghanistan 1967 11537966     Asia 34.020 836.1971
## 5 Afghanistan 1972 13079460     Asia 36.088 739.9811
## 6 Afghanistan 1977 14880372     Asia 38.438 786.1134
```

Ejercicio: Usar spread para transformar la mini base de datos largo en algo similar a ancho.

Unión de bases de datos.

Hay cuatro tipo de uniones principales que podemos hacer entre dos bases de datos.

- Full join
- inner join
- left join
- right join

Veremos los cuatro tipo para dos pequeñas bases de datos con las notas de alumnos en estadística y matemáticas.

EST

Nombre	Apellidos	Nota	Sexo
Juan	Sanchez	7	Hombre
Pepa	Perez	9	Mujer
Pepa	Fernandez	8	Mujer

MAT

Nombre	Apellido	Nota	Sexo
Juan	Sanchez	8	Hombre
Pepa	Fernandez	6	Mujer
Luis	Martinez	7	Hombre

Full join

```
full_join(EST, MAT,
          by=c("Nombre", "Apellidos"="Apellido", "Sexo"),
          suffix =c("_EST", "_MAT"))
```

EST

Nombre	Apellidos	Nota	Sexo
Juan	Sanchez	7	Hombre
Pepa	Perez	9	Mujer
Pepa	Fernandez	8	Mujer

MAT

Nombre	Apellido	Nota	Sexo
Juan	Sanchez	8	Hombre
Pepa	Fernandez	6	Mujer
Luis	Martinez	7	Hombre

Nombre	Apellidos	Nota_EST	Nota_MAT	Sexo
Juan	Sanchez	7	8	Hombre
Pepa	Perez	9	NA	Mujer
Pepa	Fernandez	8	6	Mujer
Luis	Martinez	NA	7	Hombre

inner join

```
inner_join(EST, MAT,
           by=c("Nombre", "Apellidos"="Apellido", "Sexo"),
           suffix =c("_EST", "_MAT"))
```

EST				MAT			
Nombre	Apellidos	Nota	Sexo	Nombre	Apellido	Nota	Sexo
Juan	Sanchez	7	Hombre	Juan	Sanchez	8	Hombre
Pepa	Perez	9	Mujer	Pepa	Fernandez	6	Mujer
Pepa	Fernandez	8	Mujer	Luis	Martinez	7	Hombre

Nombre	Apellidos	Nota_EST	Nota_MAT	Sexo
Juan	Sanchez	7	8	Hombre
Pepa	Fernandez	8	6	Mujer

right join

```
right_join(EST, MAT,
           by=c("Nombre", "Apellidos"="Apellido", "Sexo"),
           suffix =c("_EST", "_MAT"))
```

EST				MAT			
Nombre	Apellidos	Nota	Sexo	Nombre	Apellido	Nota	Sexo
Juan	Sanchez	7	Hombre	Juan	Sanchez	8	Hombre
Pepa	Perez	9	Mujer	Pepa	Fernandez	6	Mujer
Pepa	Fernandez	8	Mujer	Luis	Martinez	7	Hombre

Nombre	Apellidos	Nota_EST	Nota_MAT	Sexo
Juan	Sanchez	7	8	Hombre
Pepa	Fernandez	8	6	Mujer
Luis	Martinez	NA	7	Hombre

left join

```
left_join(EST, MAT,
          by=c("Nombre", "Apellidos"="Apellido", "Sexo"),
          suffix =c("_EST", "_MAT"))
```

EST

Nombre	Apellidos	Nota	Sexo
Juan	Sanchez	7	Hombre
Pepa	Perez	9	Mujer
Pepa	Fernandez	8	Mujer

MAT

Nombre	Apellido	Nota	Sexo
Juan	Sanchez	8	Hombre
Pepa	Fernandez	6	Mujer
Luis	Martinez	7	Hombre

Nombre	Apellidos	Nota_EST	Nota_MAT	Sexo
Juan	Sanchez	7	8	Hombre
Pepa	Perez	9	NA	Mujer
Pepa	Fernandez	8	6	Mujer

Capítulo 5

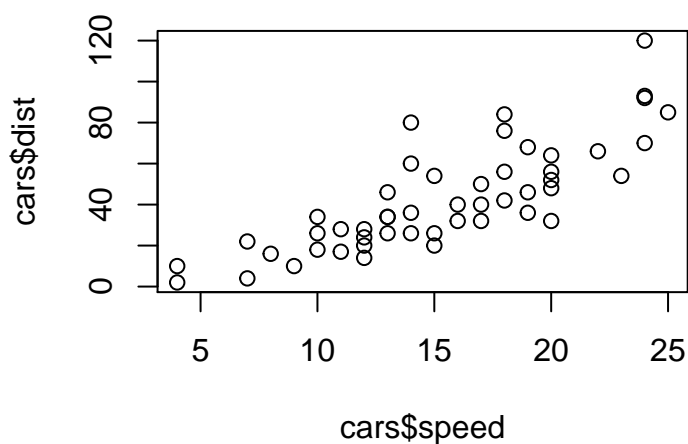
Gráficos

El medio *no* es el mensaje, pero el mensaje es un gráfico. Con **R** es fácil obtener excelentes gráficos de calidad profesional para incorporar en publicaciones. El tema podría por sí solo constituir la materia de un curso, y de hecho hay libros enteros dedicados a ello. Aparte, hay paquetes que amplían las posibilidades todavía más. Aquí consideraremos los aspectos básicos de los gráficos de **R**. Se dividen las funciones en tres categorías: *de alto nivel* (para crear gráficos), *de bajo nivel* (para modificarlos), y las que controlan los *parámetros gráficos*.

Funciones de alto nivel

Están pensadas para generar un gráfico nuevo con los datos que se les proporcionan. Una de estas funciones es `plot(x,y)`, que toma como argumento dos vectores numéricos y representa los puntos. Representemos los datos `cars`, que contienen la distancia de frenado (*dist*, en pies) y la velocidad del coche (*speed*, en millas por hora):

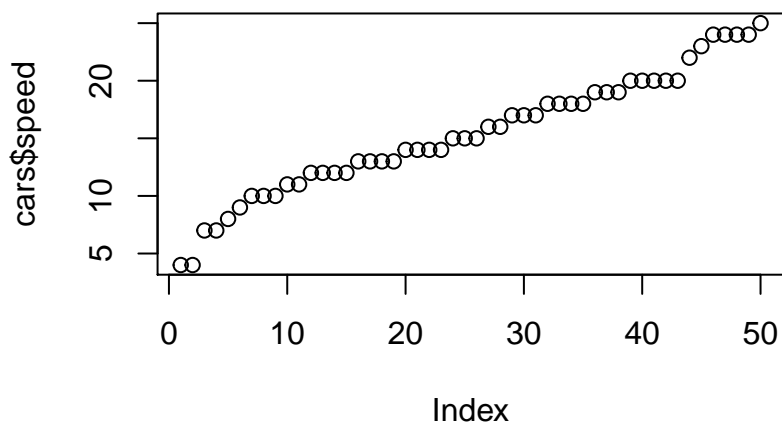
```
plot(cars$speed,cars$dist)
```



Este gráfico se llama *scatterplot* en inglés. Nótese que `plot` admite dos sintaxis: `plot(x,y)` o bien `plot(y~x)` (en R, *y~x* se lee *y en función de x*). Ambas son equivalentes.

Si sólo se le pasa un argumento, entonces `plot(x)` representa la serie `x`, es decir, los valores de `x` en el eje de ordenadas en función del índice del componente:

```
plot(cars$speed)
```

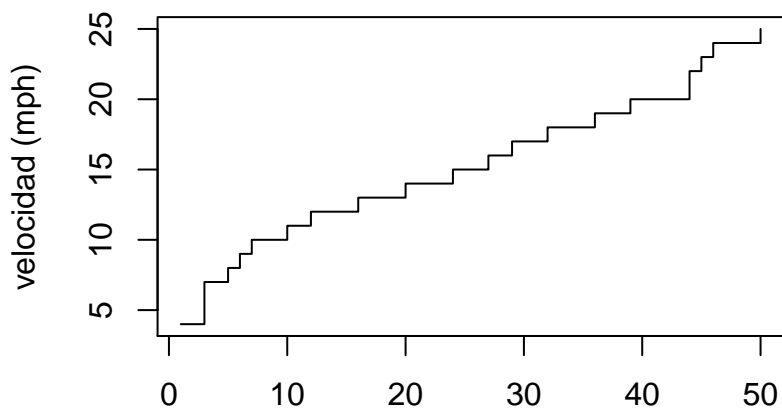


Argumentos que se pueden pasar a las funciones de alto nivel (El símbolo `|` significa `o` : escoger entre una de las posibilidades).

- `add = TRUE` dibuja el gráfico encima del que había [solo válido para algunos gráficos].
- `log = {"x" | "y" | "xy"}` pone en escala logarítmica el eje indicado.
- `type = {"p" | "l" | "b" | "s"}` representa (x,y) con puntos, líneas, ambos, o escalones.
- `xlab = "..."`, `ylab = "..."`, `main = "..."`, ponen títulos a los ejes y al gráfico.

Por ejemplo:

```
plot(cars$speed,type="s",xlab="",ylab="velocidad (mph)")
```

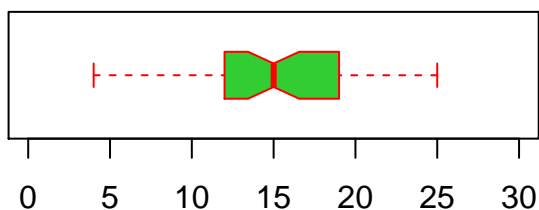


Los parámetros gráficos se pueden definir de otra manera, con la función `par()`, que se explica más adelante. Hay además algunos parámetros específicos de cada función (otras funciones de alto nivel son `boxplot`, `hist` y `stripchart`, por ejemplo). Es recomendable leer en la ayuda de R las opciones que admiten.

Ejemplos:

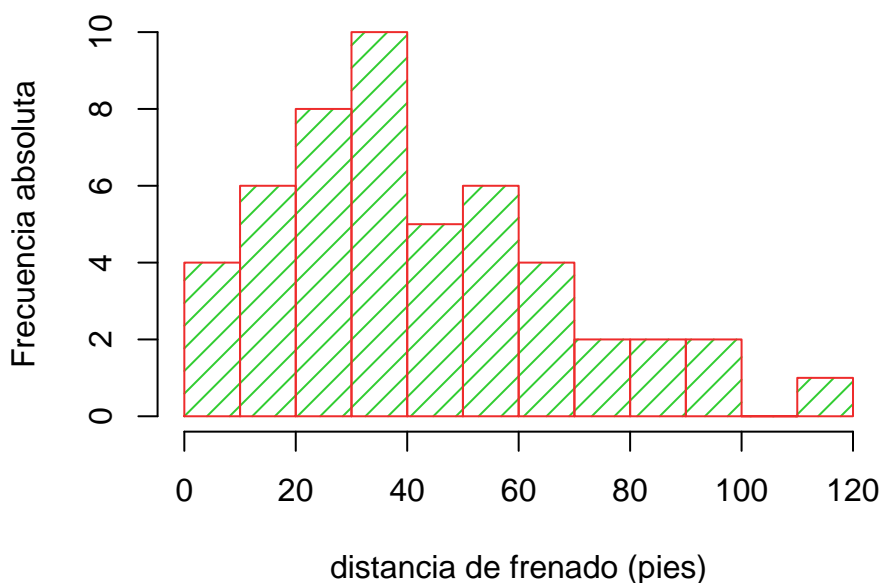
```
boxplot(cars$speed, main='diagrama de cajas horizontal', notch = TRUE,
        ylim=c(0,30),horizontal = TRUE, border='red', col='limegreen')
```

diagrama de cajas horizontal



```
hist(cars$dist, main='histograma', breaks=seq(from=0,to=120,by=10),density=12,
     col='limegreen',border='firebrick2', xlab='distancia de frenado (pies)',
     ylab='Frecuencia absoluta')
```

histograma



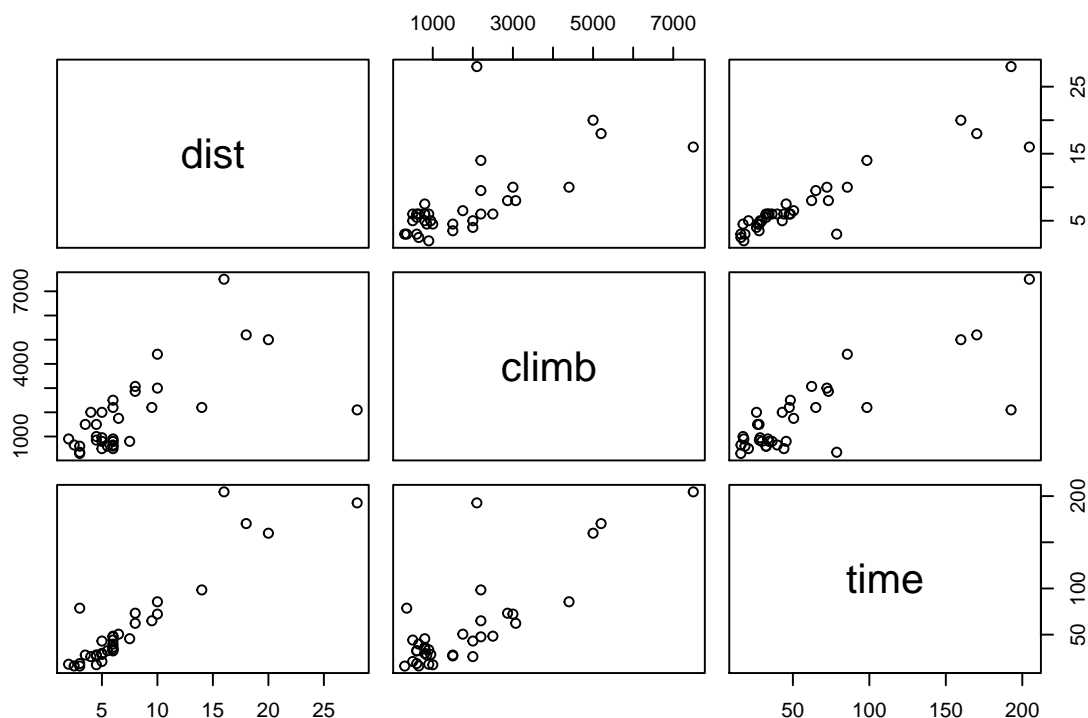
Para exportar los gráficos e incluirlos en otro documento, lo más cómodo suele ser utilizar el menú **Export** desde la pestaña **Plots**. Si el gráfico es sencillo, un formato vectorial (como PNG o EPS) suele ser el mejor compromiso entre la calidad y el tamaño del archivo.

Si se trata de datos multivariados, hay una función que permite explorar visualmente las correlaciones entre las variables: `pairs`. Por ejemplo, `hills` (dentro de la librería `MASS`) contiene los mejores tiempos para 35 carreras de montaña en Escocia, donde las variables son: `dist`, la distancia en millas ; `climb`, el desnivel acumulado, en pies; y `time`, la mejor marca, en minutos. A continuación se representan las variables por pares (*más adelante se explica cómo representar varios gráficos en la misma figura; `pairs` lo hace automáticamente*).

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.3.3
```

```
pairs(hills)
```



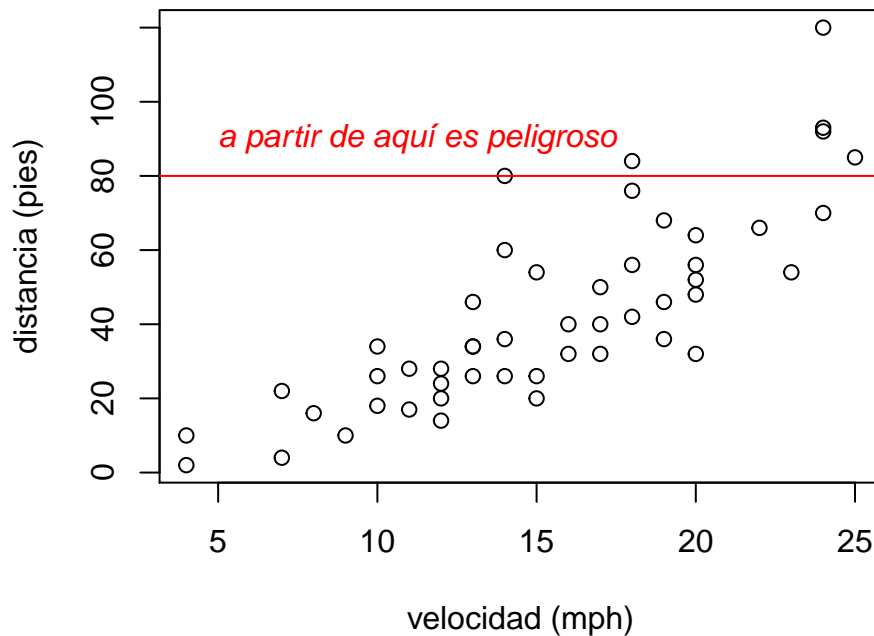
Funciones de bajo nivel

Se usan en general para añadir elementos a un gráfico ya existente.

- `points(x,y)` añade los puntos indicados
- `lines(x,y)` dibuja líneas
- `text(x,y, "texto")`
- `abline` dibuja líneas:
 - `abline(a,b)` con pendiente a y ordenada en el origen b
 - `abline(h=valor)` línea horizontal en $y=valor$
 - `abline(v=valor)` línea vertical en $x=valor$
- `polygon(x,y)` dibuja un polígono con los vértices ordenados $\{(x,y)\}$
- `legend` añade una leyenda

Ejemplo:


```
plot(cars$speed,cars$dist,xlab="velocidad (mph)",ylab="distancia (pies)")
abline(h=80, col="red")
text(5,90,"a partir de aquí es peligroso",font=3, adj=c(0,NA), col="red")
```



Parámetros gráficos

Permiten cambiar un gran número de características: aspecto de los símbolos, las marcas de los ejes, dimensiones del gráfico, etc. En general, la sintaxis es `propiedad = valor`. Hay dos maneras de hacerlo: de manera que afecte solo a una función gráfica (y entonces se le pasa a esa función `propiedad = valor` como argumento), o bien de manera que el cambio afecte ya a todo lo que se represente a partir de ese momento. Esto último se logra con la función `par(propiedad = valor)`. En este último caso, para revertir los cambios a los parámetros originales, hay que tener la precaución de guardar los valores de las propiedades originales y luego restaurarlos. Eso se hace así:

```
par_originales <- par( no.readonly = TRUE )
...
par(par_originales)
```

Consultar la ayuda de `par` para una lista exhaustiva de propiedades (aquí sólo se dan algunas).

Propiedades de los elementos gráficos (símbolos, líneas, texto ...)

- Tipo de símbolo: `pch`. Puede tomar como valor cualquier carácter (entre comillas), o un número del 1 al 25, correspondiente a una serie de símbolos predefinidos.
- Tamaño del símbolo: `cex = número` (multiplicador: 1 es el valor por defecto).

- Tipo de línea: `lty`. El valor puede ser especificado como un número o una cadena de caracteres: 0 = "blank", 1 = "solid" (default), 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash". La primera (0 ó "blank") no dibuja nada. El ancho de línea se indica con `lwd = número`.
- Color: `col` (para el relleno de símbolos es `bg`). Véase la lista de colores.

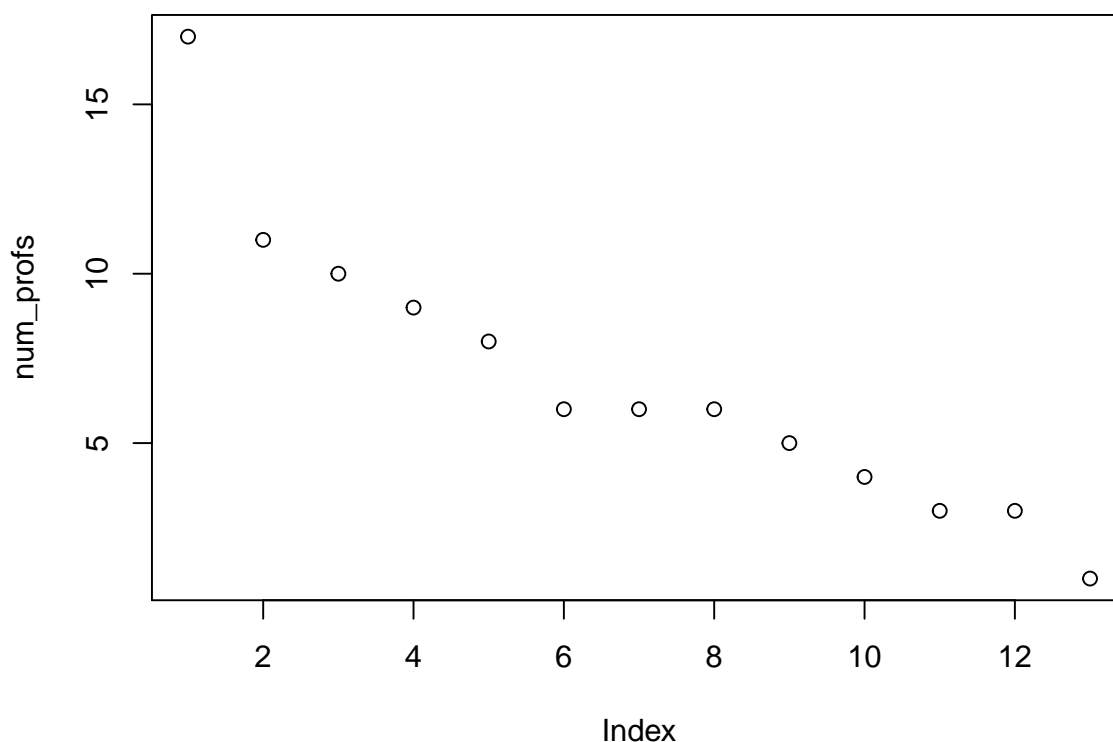
Ejemplo:

```
# type = "n" no dibuja nada; pero abre un nuevo gráfico.
plot(1,1,xlim=c(0,25),ylim=c(0,4),type = "n",xlab = "",ylab = "")
# fondo gris claro
rect(par("usr")[1], par("usr")[3], par("usr")[2], par("usr")[4], col = "snow2")
points(0,0.5,pch="$",col="firebrick2") # el símbolo es el carácter $
Colorettes=colors() # lista de los colores definidos en R, unos 500
for (contador in 1:25){
  points(contador,0.5, pch=contador,col="black",bg="limegreen")
  points(contador,1.3, pch=contador, col=Colorettes[contador*4],cex=2, lwd=2)
}
for (contador in 1:6){
  lines(c(4*(contador-1),4*(contador-1)+3),c(2,2),lty=contador, col="red")
  lines(c(4*(contador-1),4*(contador-1)+3),c(3,3),lty=contador,lwd=2)
}
text(3,2.3,adj=0,"diferentes tipos de línea, ancho 1")
text(3,3.3,adj=0,"diferentes tipos de línea, ancho 2",font=4) # negrita cursiva
```



Propiedades de los ejes, márgenes, dimensiones

```
profs=c("admin./func.", "comerciales", "agroalimentación", "prof.salud", "leyes",
"gremios", "autónomos", "directivos", "ingeniería", "enseñanza", "emplead.hogar",
"estudiantes", "fuerzas.armadas" )
num_profs <- c(17, 11, 10, 9, 8, 6, 6, 6, 5, 4, 3, 3, 1)
plot(num_profs) # gráfico con los valores por defecto
```



A continuación, algunas modificaciones posibles:

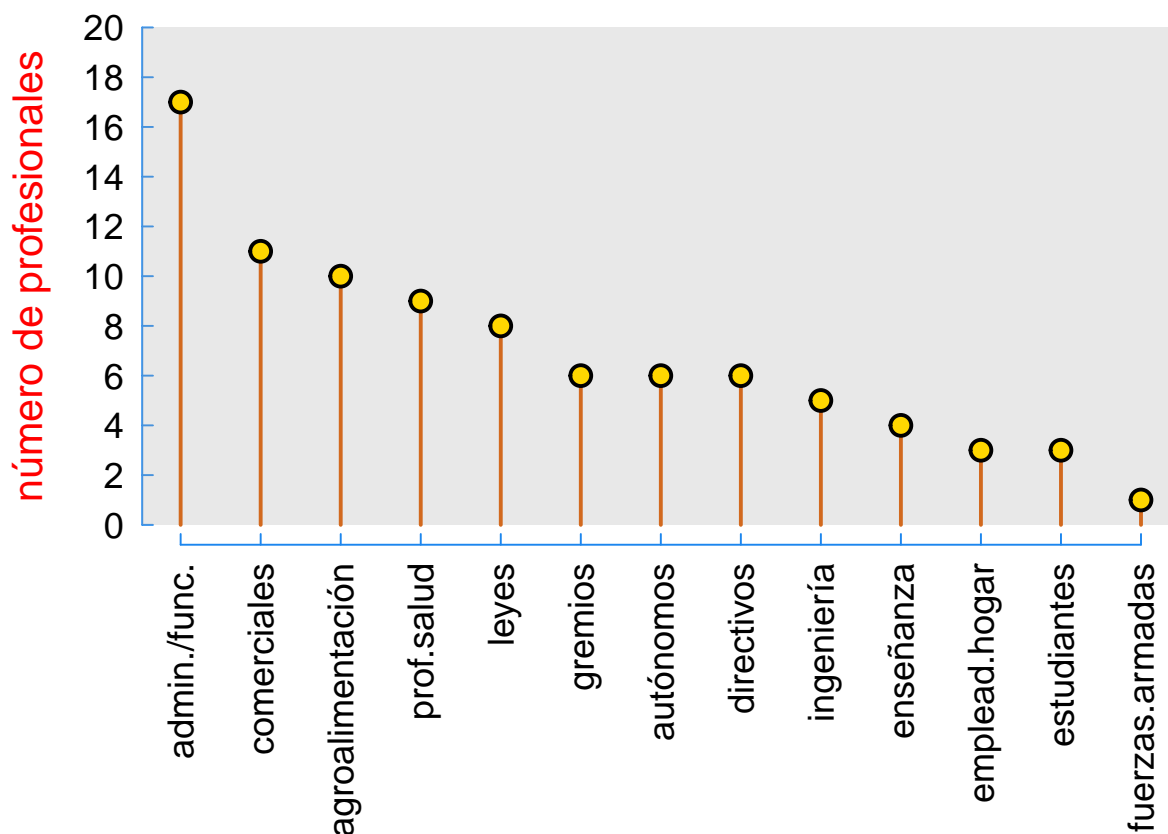
```
op_orig <- par(no.readonly = TRUE) # se guardan los parámetros originales

par(mai=c(2,0.8,0.1,0.1)) # margen interno: abajo/izda./arriba/dcha.
par(las=2) # orientación de las etiquetas
par(lab=c(length(num_profs),8,2)) # etiquetas de las marcas
par(mgp=c(2.5,0.5,0),tck=0.02) # posición de títulos, marcas y etiquetas
par(fg="dodgerblue2",bty="n",cex.axis=1.2) # color ejes, sin caja, ampliación

# type = "n" no dibuja nada ... Sirve para construir los ejes
plot(num_profs,type= "n",ylim=c(0,20),xlim=c(1,length(num_profs)),
      xlab=" ",xaxt="n", # sin etiquetas ni títulos en x
      ylab = "número de profesionales",col.lab="red", cex.lab=1.3)

recgraf = par("usr")
# rectángulo con el color de fondo gris
rect(recgraf[1],0,recgraf[2],20,col=rgb(0.7,0.7,0.7,0.3),border=NA)

points(num_profs,type= "h",col="chocolate",lwd=2) # líneas verticales
points(num_profs,pch=21,cex=1.5,col="black",lwd=2,bg="gold") # puntos
axis(1,at=1:13, labels = profs) # etiquetas en x
```



```
par(op_orig) # se vuelven a dejar los parámetros como estaban
```

En R, hay muchos otros tipos de gráficos, por ejemplo `qqplot` (gráficos cuantil-cuantil), `stripcharts` o `rug plots`.

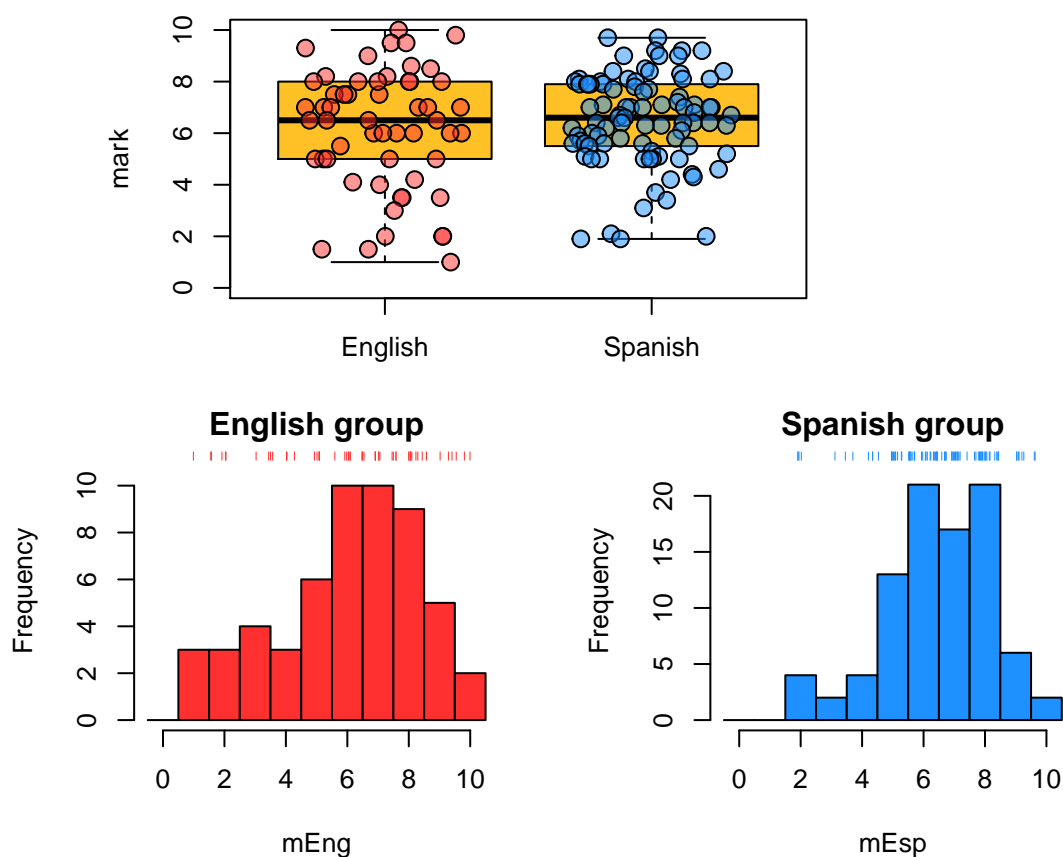
Varios gráficos en la misma figura Se pueden representar varios gráficos en la misma figura con `mfrow`, `mfg` y `layout`. La primera define los paneles; la segunda escoge el panel para usar. La última permite un mayor control sobre la disposición.

En el ejemplo siguiente, se han tomado las notas de los grupos de inglés y castellano (primer curso) para compararlas. Nótese los siguientes aspectos:

- El empleo de `add = TRUE` para evitar la creación de un gráfico nuevo. Con eso se dibujan varios gráficos uno encima de otro.
- La **transparencia** está definida en R dentro del color; en vez de un simple vector de tres componentes (*red*, *green* y *blue*), la función `rgb` puede definir colores con *cuatro* componentes; el último es el llamado *alpha*: un valor entre 0 y 1 que indica la transparencia. En esta figura, los `stripcharts` permiten ver los `boxplots` que quedan por debajo.
- El uso de `layout` hace que cada uno de los gráficos que se necesite crear ocupe el lugar asignado. El comando `layout(matrix(c(1,1,2,3),2,2,byrow=TRUE), widths=c(1,1), heights=c(0.9,1.1))` se interpreta considerando que se crea una matriz de 2x2, el primer gráfico toma los dos primeros paneles (o sea: el superior izquierdo [1,1] y el superior derecho [1,2], ya que se avanza por filas: `byrow = TRUE`); el siguiente, el panel inferior izquierdo [2,1]; y tercero y último, el elemento [2,2] de la matriz. Las anchuras y alturas se definen con `widhts = ...` y `heights =`

- El parámetro `jitter` añade ruido a los datos para evitar que coincidan en el mismo punto exactamente.

```
marks = read.csv("notas_E_S.csv", sep=";")
mEng=marks$English
mEsp=marks$Spanish
MM=c(mEng,mEsp)
GG=c(rep('English',length(mEng)),rep('Spanish',length(mEsp)))
ms=data.frame(MM,GG)
par(mai=c(0.5,0.5,0.3,0.1))
layout(matrix(c(1,1,2,3),2,2,byrow=TRUE),widths=c(1,1),heights=c(0.9,1.1))
par(mai=c(0.5,1.5,0.3,1.5))
boxplot(ms$MM~ms$GG,ylim=c(0,10),col='goldenrod1',ylab='mark')
stripchart(mEng,at=1,vertical = TRUE,add = TRUE, jitter=0.3,
           method = "jitter",pch=21,bg=rgb(1,0.19,0.19,0.5),cex=1.4)
stripchart(mEsp,at=2,vertical = TRUE,add = TRUE, jitter=0.3,
           method = "jitter",pch=21,bg=rgb(0.12,0.56,1,0.5),cex=1.4)
par(mai=c(1,1,0.3,0.1))
hist(mEng,main = "English group",breaks=seq(from=-0.5,to=10.5,by=1),
     col='firebrick1',ylim=c(0,11))
rug(jitter(mEng,amount=0.1),col='firebrick1',side=3)
hist(mEsp,main = "Spanish group",breaks=seq(from=-0.5,to=10.5,by=1),
     col="dodgerblue",ylim=c(0,23) )
rug(jitter(mEsp,amount=0.1),col='dodgerblue',side=3)
```



Consideraciones finales

Hay muchas más posibilidades para representaciones gráficas, por supuesto. Puede ser útil visitar [A compendium of clean graphs in R](#), donde se muestran los gráficos *y el código empleado para elaborarlos*, listo para copiar y pegar.

Es recomendable el empleo de **ggplot2**, un paquete de gráficos que permite ampliar notablemente las posibilidades. Más adelante se describe su uso.

Capítulo 6

Introducción a *ggplot2*

Preliminares

El paquete `ggplot2` se encuentra incluido en los que se cargan con el meta-paquete `tidyverse`. Si no está cargado ese paquete, úsese `library(ggplot2)` (e instalarlo o actualizarlo si fuese necesario).

`ggplot2` es un paquete creado por Hadley Wickam para producir gráficos de datos o estadísticos basado en la gramática de gráficos desarrollada por Wilkinson. Encontrarás mucha más información sobre este paquete en el libro online de su autor [ggplot2: Elegant graphs for data analysis](#), del que hemos extraído información para este guión.

Partes de un gráfico en `ggplot`

Todos los gráficos están compuestos de:

- **Datos** que queremos visualizar (han de estar en forma de `data.frame` o `tibble`) y un conjunto de correspondencias estéticas (*aesthetic mappings* `aes(x=,y=, color=,...)`) que describen la correspondencia entre variables del conjunto de datos y atributos estéticos que se pueden percibir percibir. Así, una variable puede corresponder al eje X, otra al eje Y, otra al color, otra al tamaño, etc.
- **Capas** hechas de elementos geométricos y transformaciones estadísticas. Los objetos geométricos `geom_` representan los elementos que se ven en el gráfico: puntos `geom_point()`, líneas `geom_line()`, diagramas de cajas `geom_boxplot()`, etc. Las transformaciones estadísticas, `stat_`, resumen los datos de alguna manera útil para transmitir información. Dividir en bins `stat_bin()` y contar observaciones `stat_n()` para hacer un histograma, o resumir la relación bidimensional entre dos variables con un modelo lineal. Son especialmente útiles cuando se desea cambiar el comportamiento estándar de una geometría para producir otro tipo de gráfico.
- **Escalas** `scale_` especifica la correspondencia entre valores de los datos y la estética correspondiente. Las escalas añaden una leyenda o ejes que permiten interpretar el gráfico y recobrar los datos originales. Por ejemplo `scale_x_log10()`
- **Sistemas de coordenadas** `coord_` Sistemas de coordenadas (cartesiano, polar,..) ejes etc.

- **Paneles** Permite subdividir los datos en subgrupos y representarlos en distintos paneles para establecer comparaciones.
- **Temas theme_** Algunos aspectos secundarios del gráfico (como el tamaño de las fuentes, el color de fondo, etc.) se pueden agrupar en temas y reutilizarlos para dar una cierta uniformidad a las figuras de un trabajo.

Ejemplo básico

Utilizaremos los datos de [gapminder](#)

Vamos a establecer el lienzo sobre el que hacemos el gráfico.

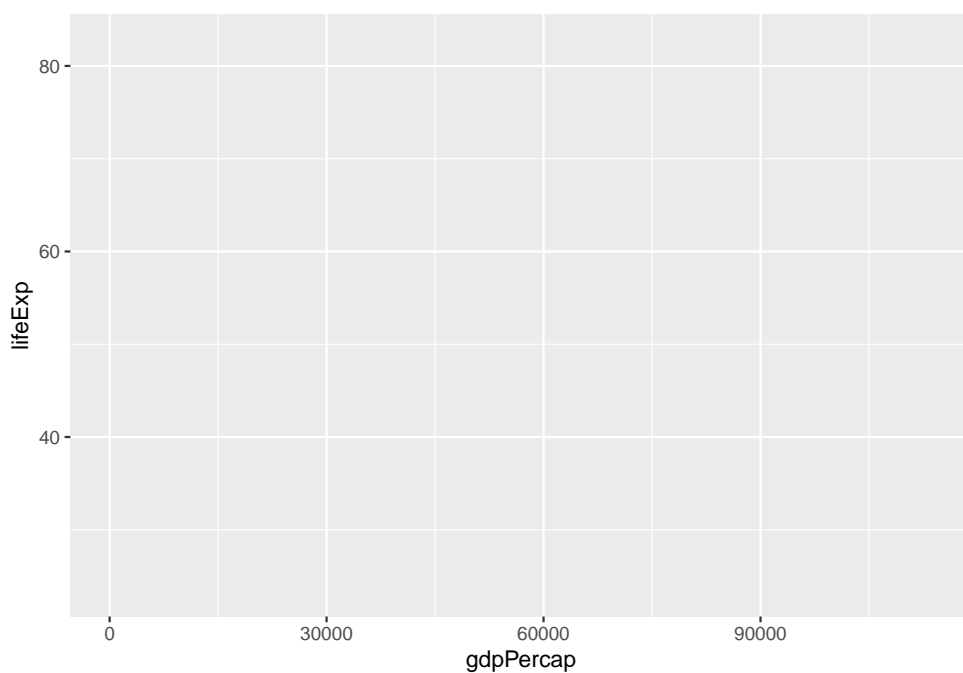
```
library(gapminder)
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.2.1 --

## v ggplot2 3.2.1      v purrr  0.3.2
## v tibble  2.1.3      v dplyr  0.8.3
## v tidyr   0.8.3      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.4.0

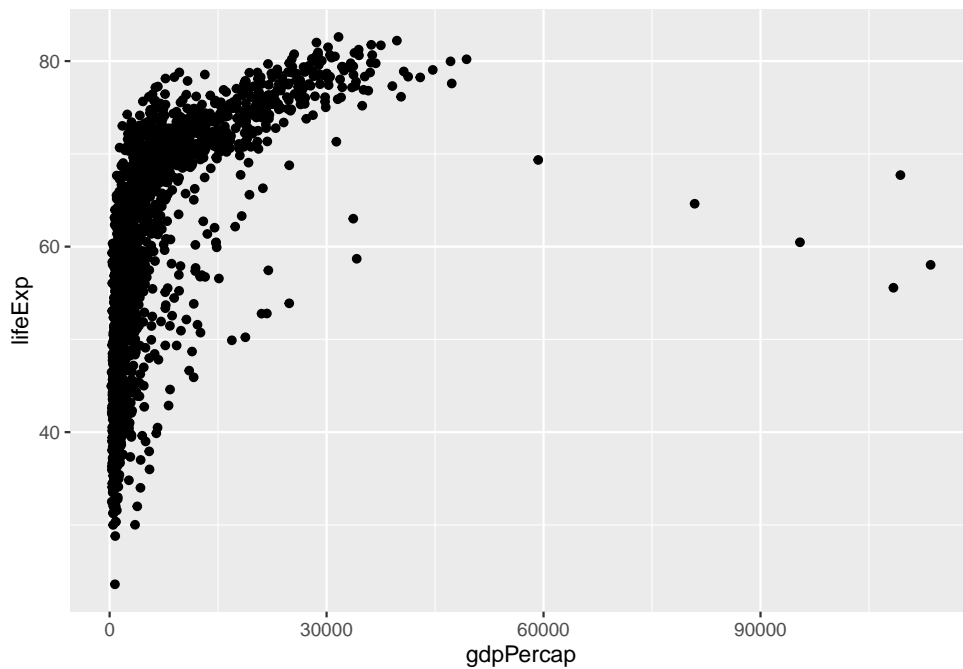
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))
```



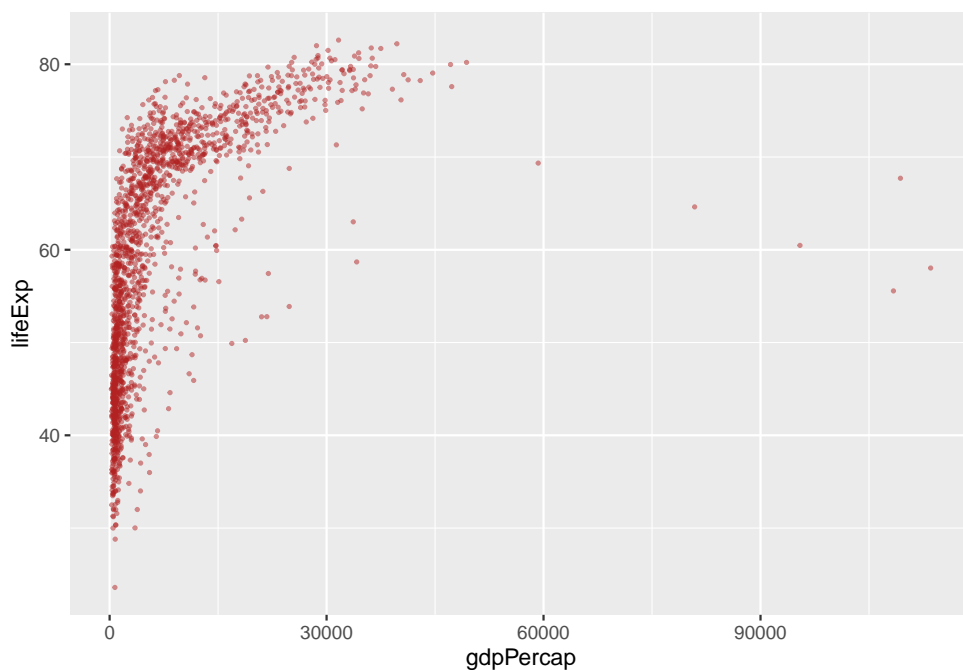
Añadimos ahora una capa con `geom_point`

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

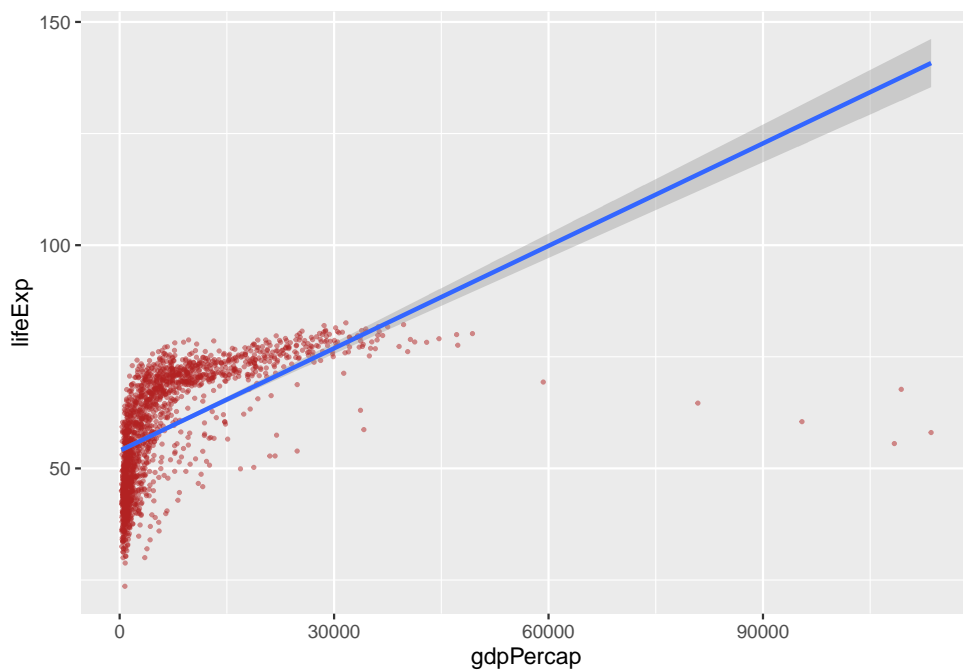
Podemos hacer alguna modificación dentro de la geometría (cambiar el color, tamaño del punto, transparencia,..)

```
ggplot(gapminder, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(col = "firebrick", size = 0.5, alpha=0.5)
```



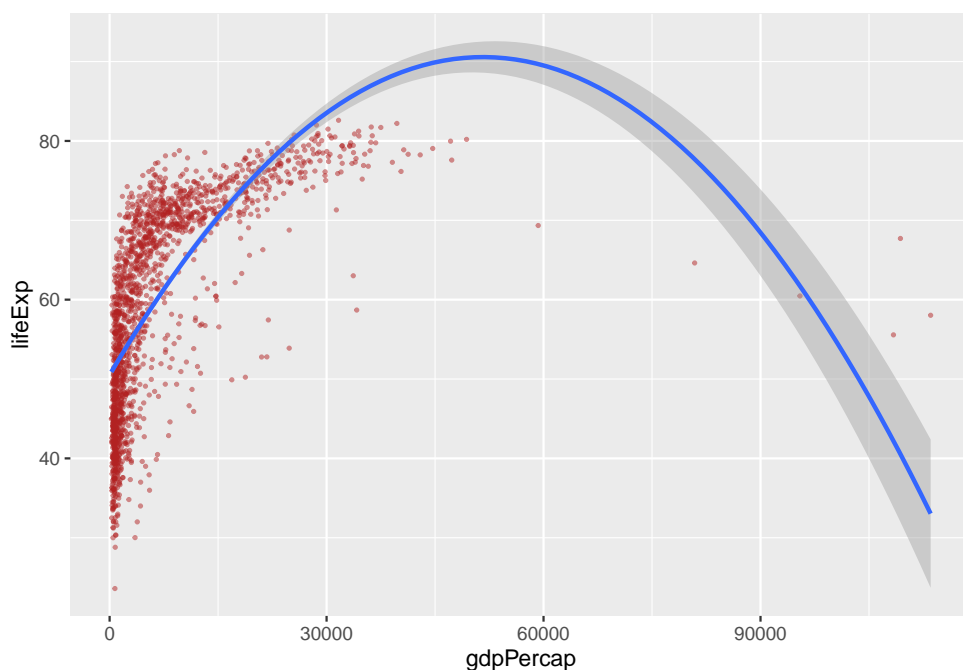
Añadimos ahora otra capa. Incluimos otra geometría que representa un ajuste lineal.

```
ggplot(gapminder, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(col = "firebrick", size = 0.5, alpha=0.5)+
  geom_smooth(method="lm", formula = y~x)
```



Este ajuste es bastante malo; probemos un ajuste cuadrático.

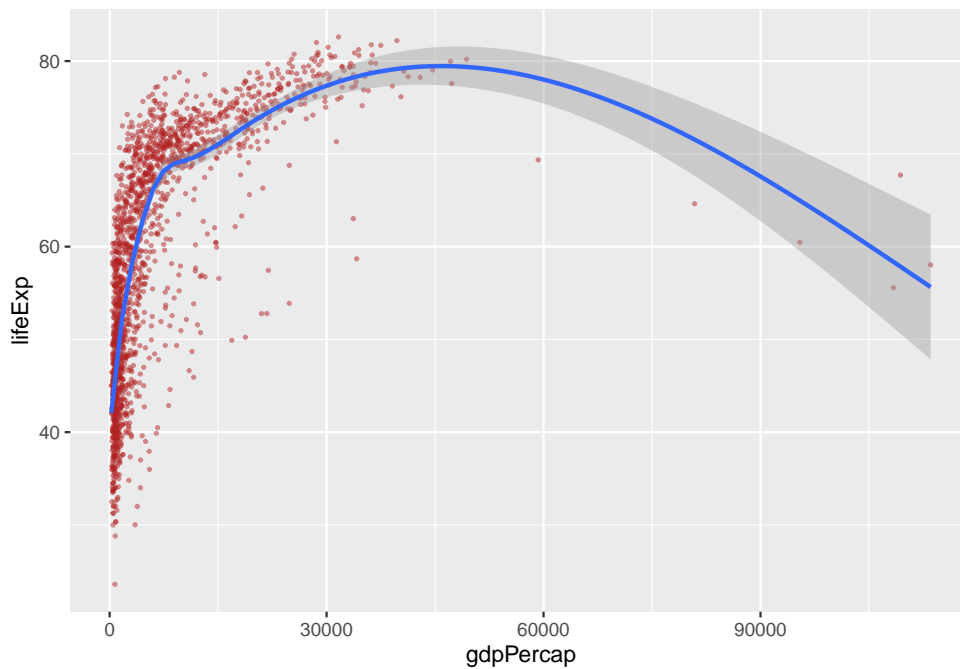
```
ggplot(gapminder, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(col = "firebrick", size = 0.5, alpha=0.5)+
  geom_smooth(method="lm", formula = y~x + I(x^2))
```



Probemos el ajuste *smooth* con las opciones por defecto (en este caso realiza un spline cúbico)

```
ggplot(gapminder, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(col = "firebrick", size = 0.5, alpha=0.5)+
  geom_smooth()
```

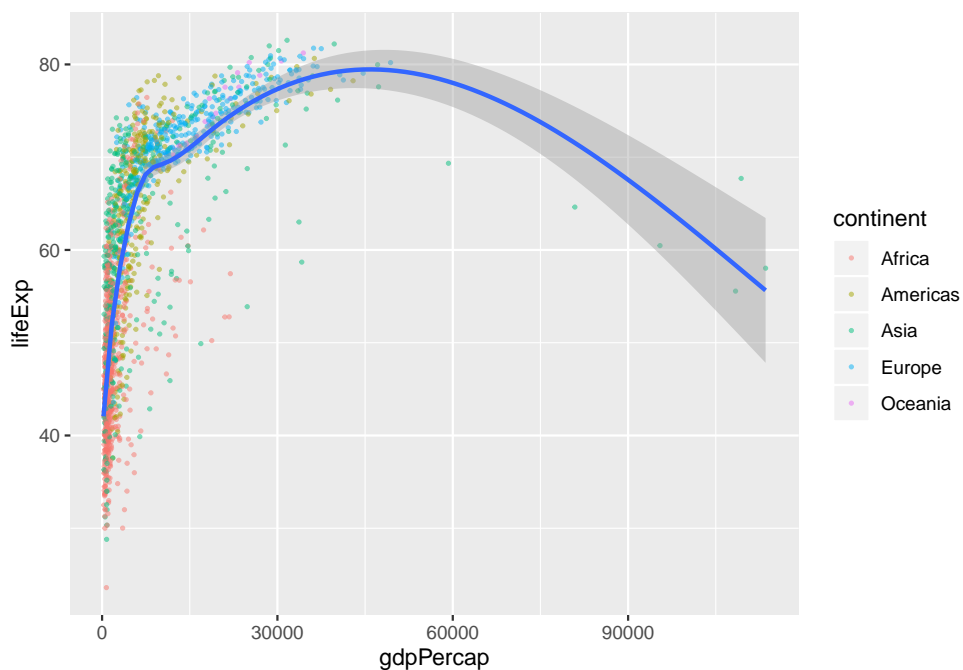
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Queremos que el color represente la variable `continent`. Para ello hemos de incluirlo dentro de la estética `aes()`. Podemos incluirlo en `geom_point` afectando únicamente a esta geometría. Nótese que se omite la opción `col="firebrick"`, ya que ahora el color será variable.

```
gapminder %>%
  ggplot(aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(size = 0.5, alpha=0.5, aes(color=continent))+
  geom_smooth()
```

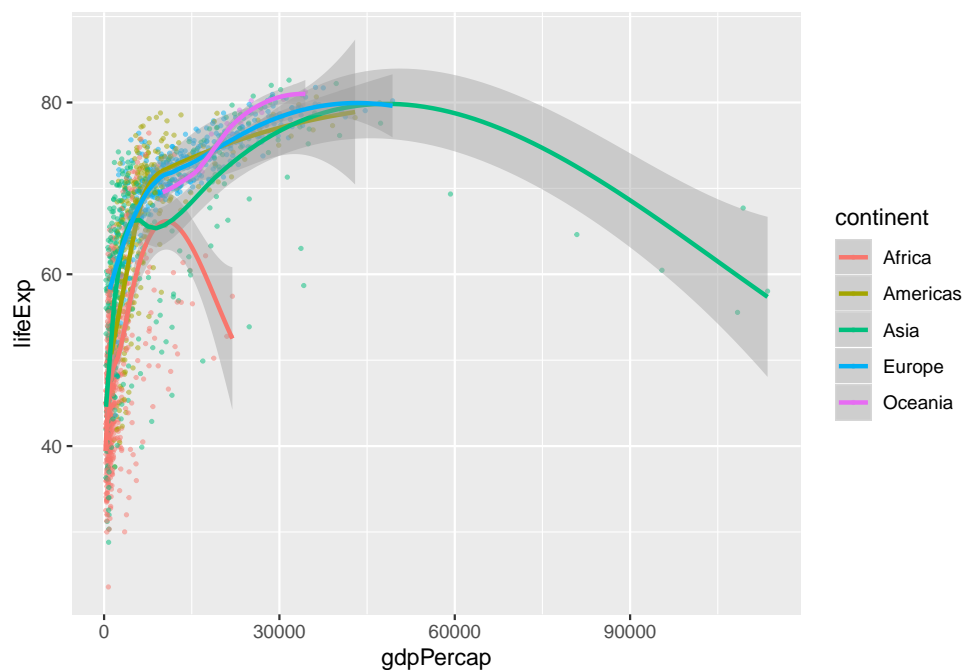
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



O incluirlo en `ggplot` afectando a todas las geometrías.

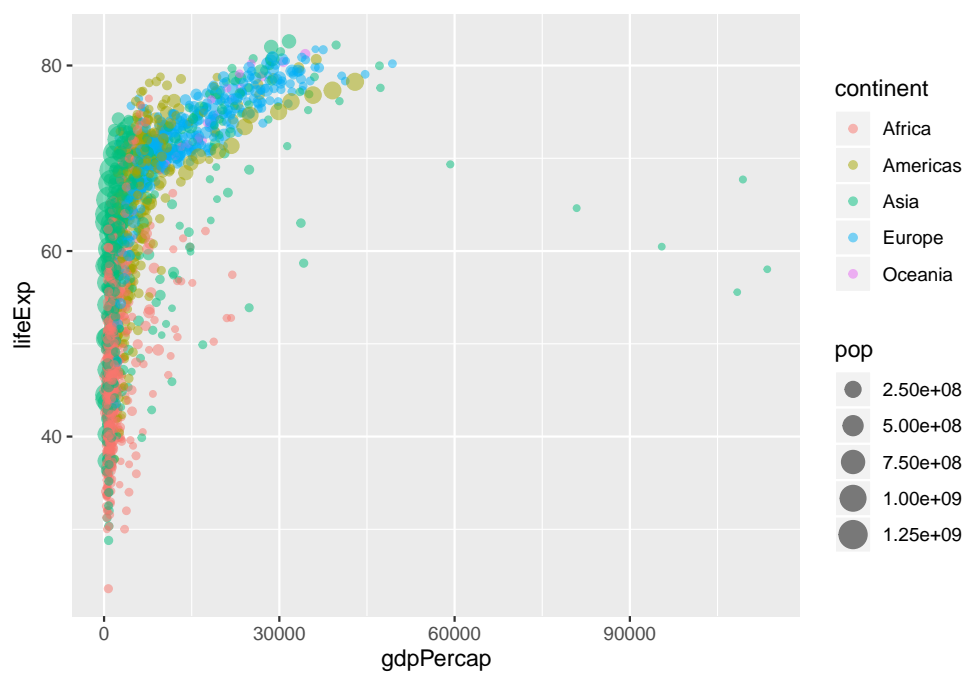
```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, color=continent)) +
  geom_point(size = 0.5, alpha=0.5)+
  geom_smooth()
```

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



Dejemos de momento el suavizado. Hagamos que el tamaño represente otra variable.

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, color=continent, size=pop)) +
  geom_point(alpha=0.5)
```

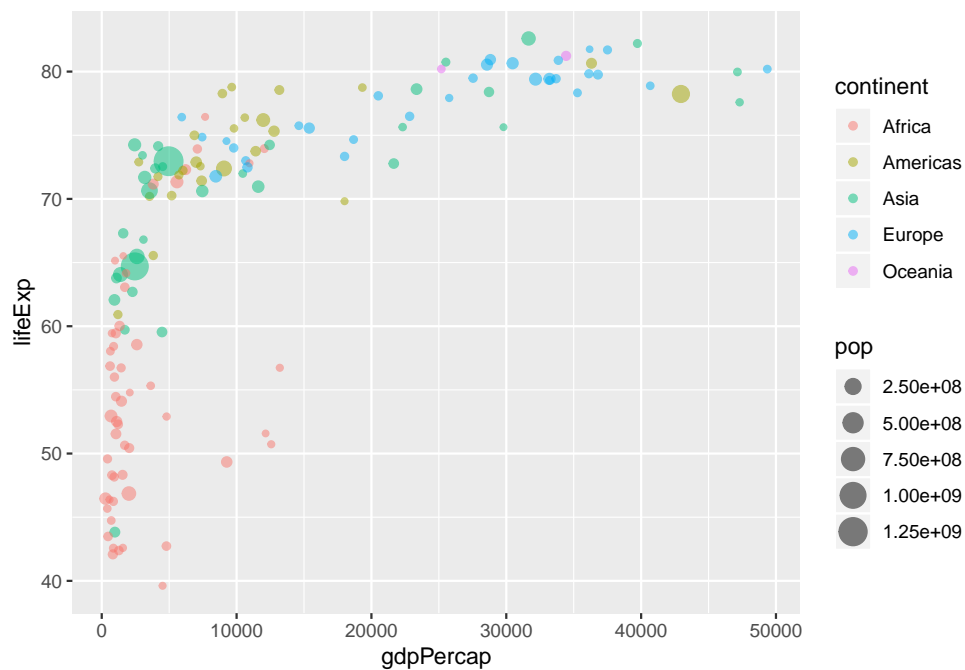


Nos centramos en un año (usando dplyr)

```

this_year <- 2007
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             color = continent, size = pop)) +
  geom_point(alpha = 0.5)

```

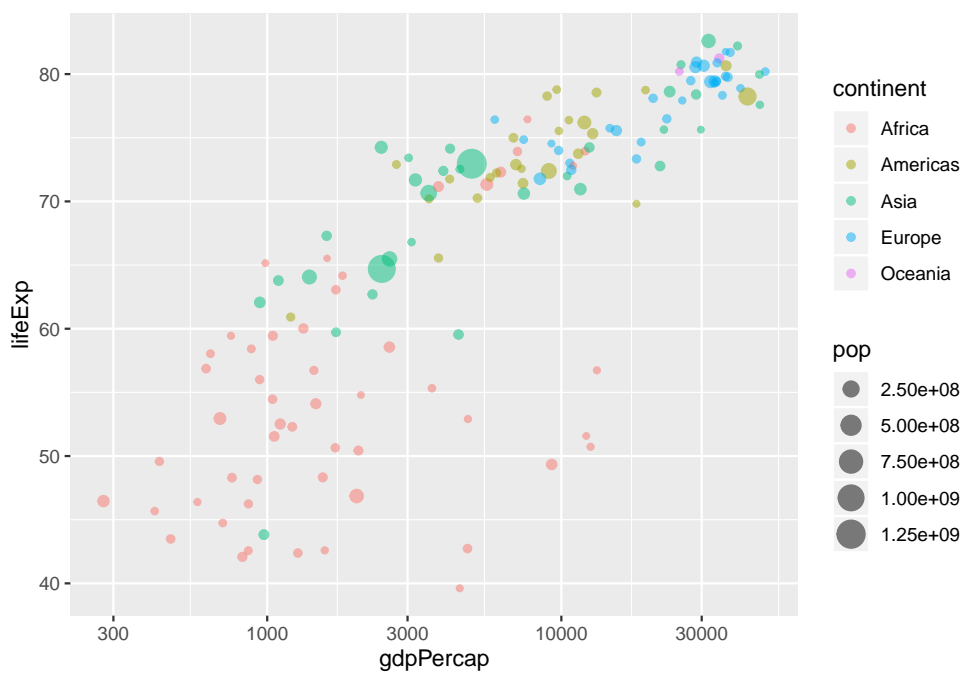


Pongamos en escala logarítmica el eje X.

```

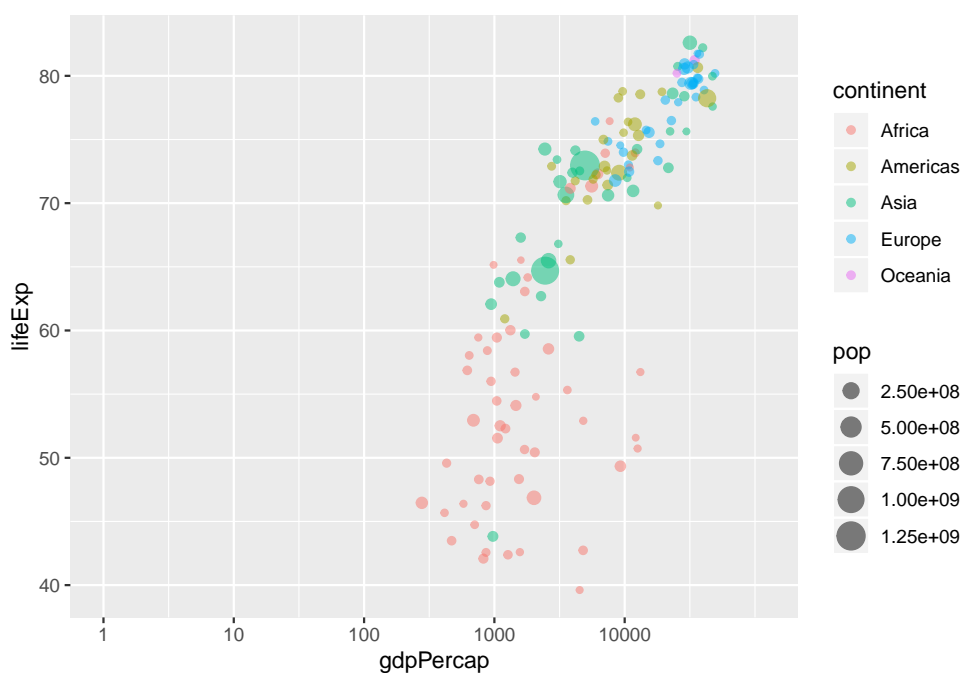
this_year <- 2007
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10()

```



Se puede personalizar esta escala

```
this_year <- 2007
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  # clean the x-axis breaks
  scale_x_log10(breaks = c(1, 10, 100, 1000, 10000),
               limits = c(1, 120000))
```

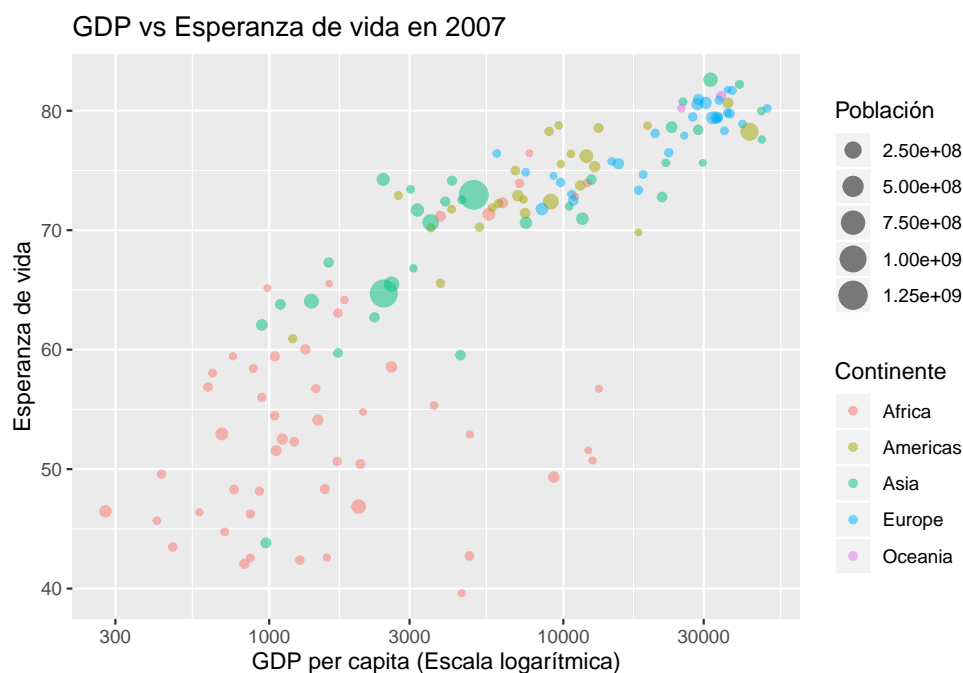


Incluir etiquetas:

```

this_year <- 2007
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Población",
       color = "Continente")

```

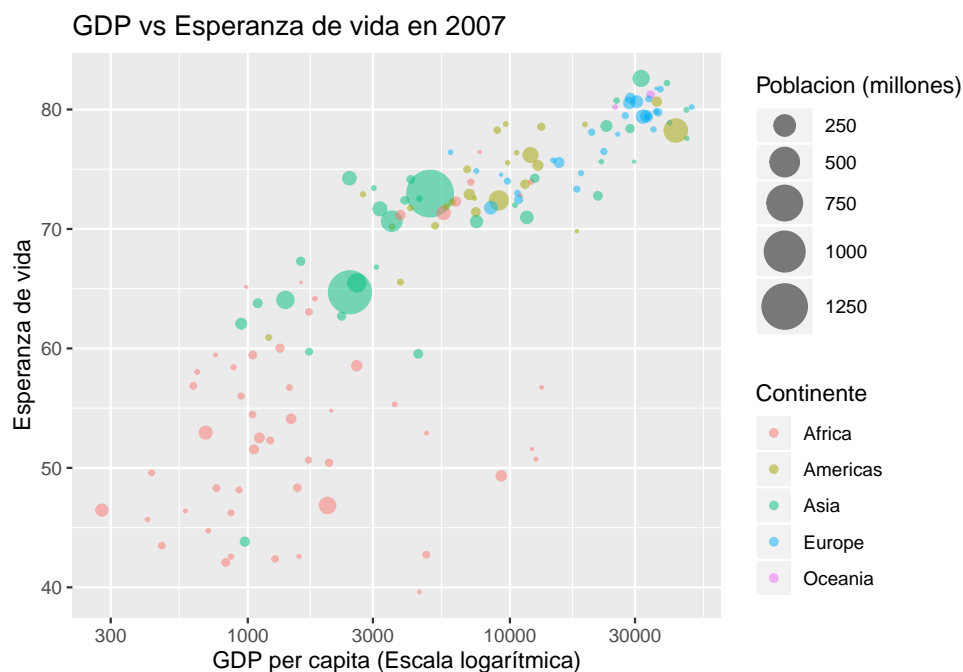


Vamos a modificar la escala del tamaño. Consideramos la variable `pop` continua ya que tiene muchos valores posibles y por tanto usamos `scale_size_continuous`

```

this_year <- 2007
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
            color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "Continente") +
  scale_size(range = c(0.1, 10),
            breaks = 1000000 * c(250, 500, 750, 1000, 1250),
            labels = c("250", "500", "750", "1000", "1250"))

```

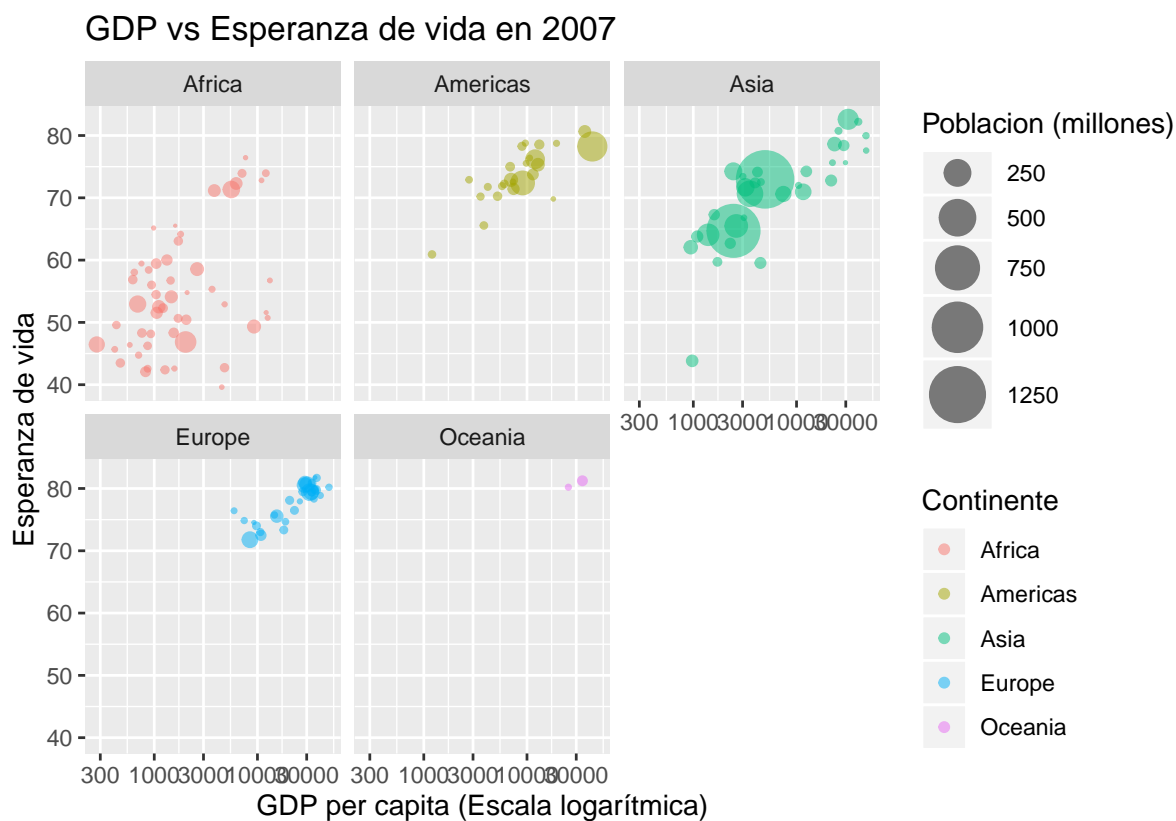


Con este ejemplo se han mostrado algunas de las posibilidades que tiene este paquete en cuanto a la presentación gráfica de los datos. Los gráficos científicos deben presentar idealmente una gran cantidad de información, fácil de digerir, de manera fidedigna, con el mínimo esfuerzo para el lector. Los buenos gráficos deben ser autoexplicativos.

Paneles

Podemos dividir en paneles (`facet_wrap`)

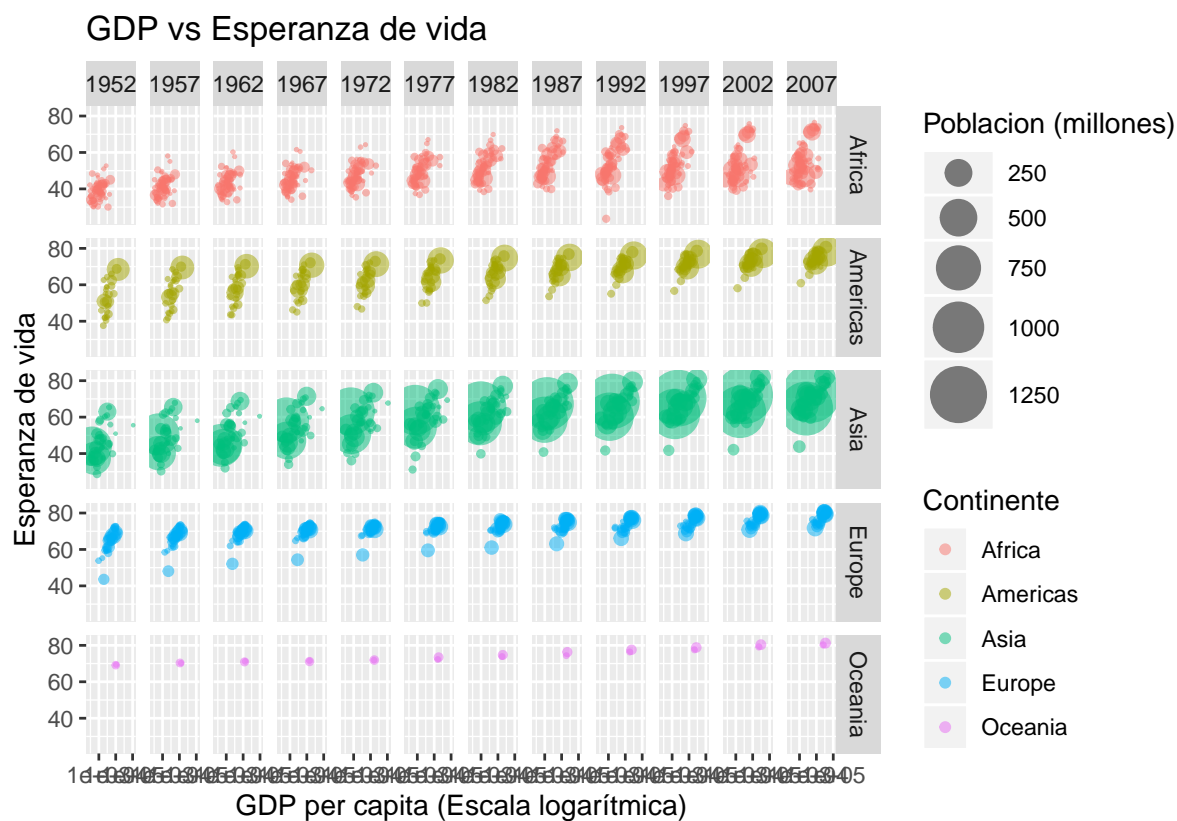
```
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "Continente") +
  scale_size(range = c(0.1, 10),
            breaks = 1000000 * c(250, 500, 750, 1000, 1250),
            labels = c("250", "500", "750", "1000", "1250")) +
  facet_wrap(~continent)
```

Paneles con facet-grid

```

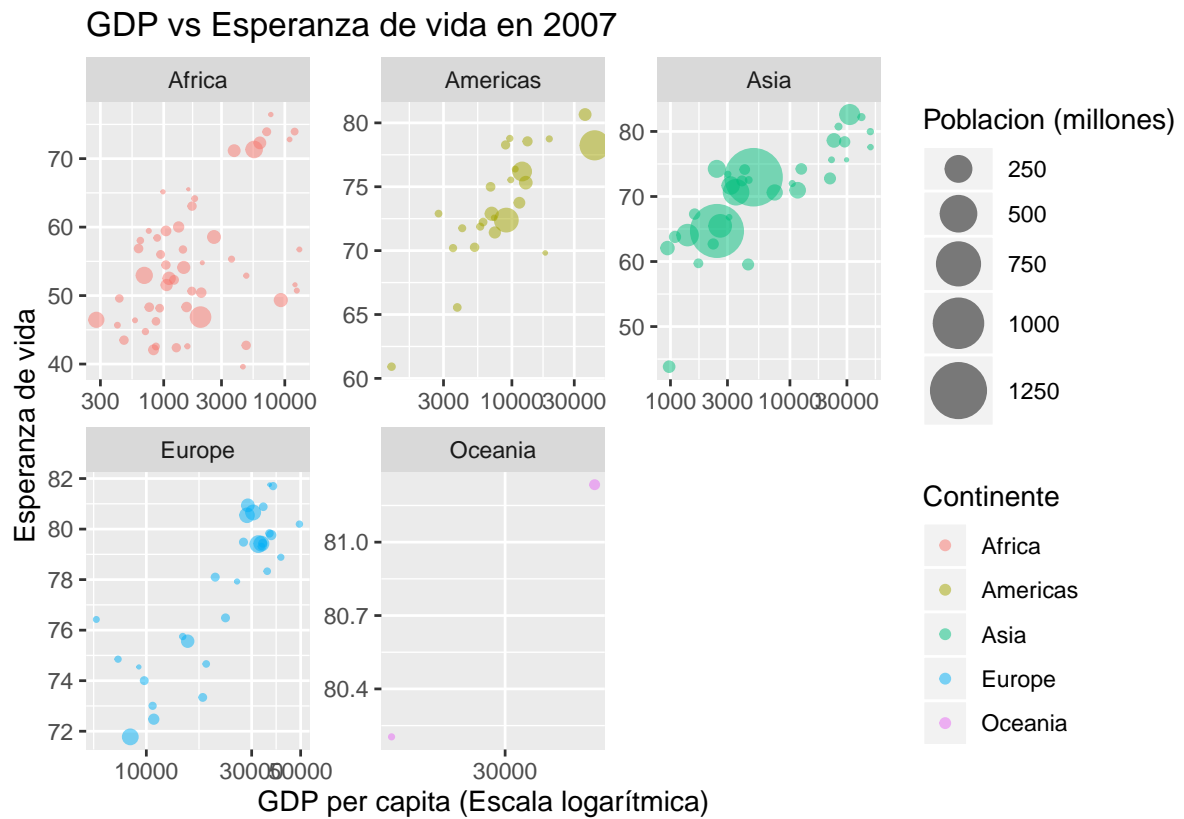
gapminder %>%
  # filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "Continente") +
  scale_size(range = c(0.1, 10),
            breaks = 1000000 * c(250, 500, 750, 1000, 1250),
            labels = c("250", "500", "750", "1000", "1250"))+
  facet_grid(continent~year)
  
```



Escalas

Es posible *liberar* las escalas de los paneles para ajustarse mejor al rango de los datos, pero esto puede ser contraproducente si se quiere comparar datos entre paneles.

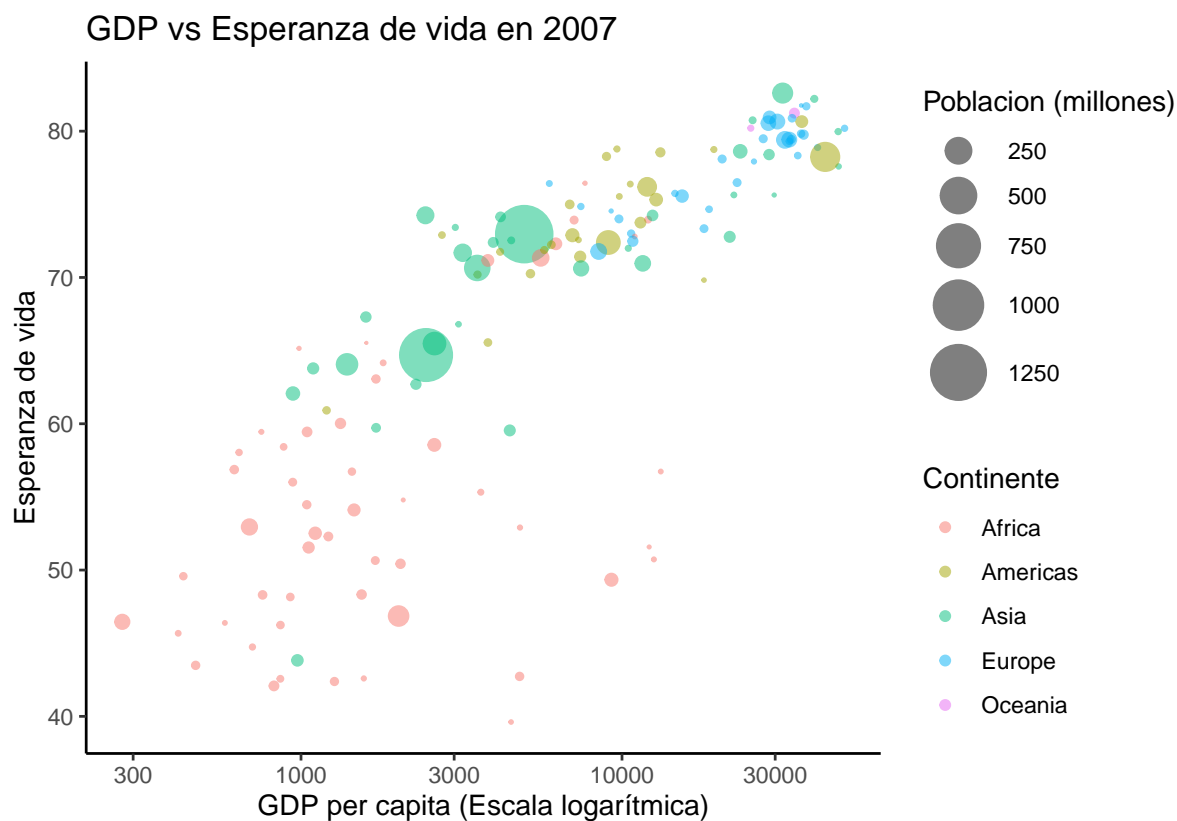
```
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "Continente") +
  scale_size(range = c(0.1, 10),
            breaks = 1000000 * c(250, 500, 750, 1000, 1250),
            labels = c("250", "500", "750", "1000", "1250"))+
  facet_wrap(~continent, scales="free")
```



Temas

Los temas se emplean para dar un aspecto uniforme a un conjunto de gráficos.

```
gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "Continente") +
  scale_size(range = c(0.1, 10),
            breaks = 1000000 * c(250, 500, 750, 1000, 1250),
            labels = c("250", "500", "750", "1000", "1250"))+
  theme_classic()
```



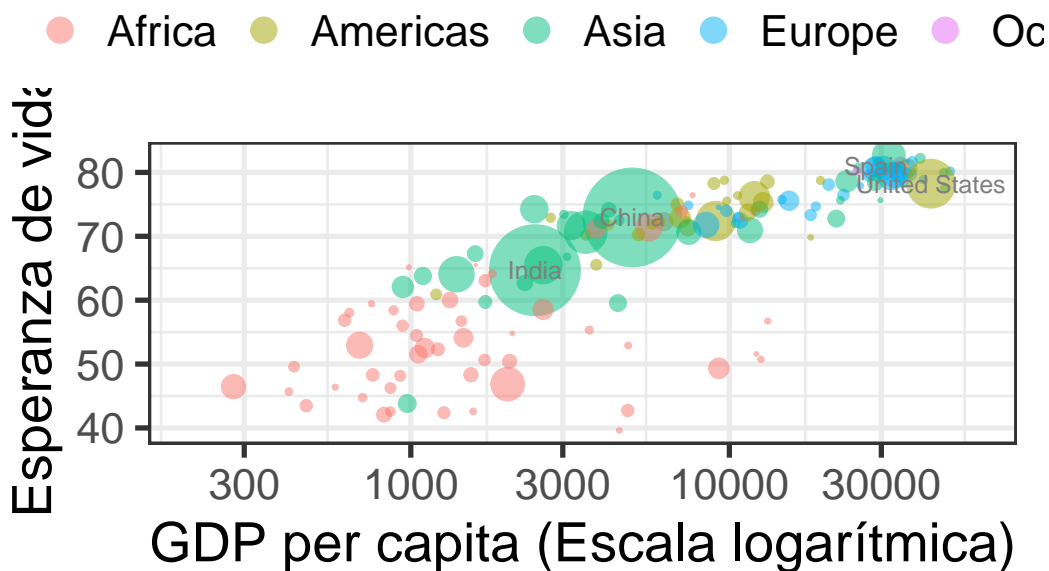
Es posible modificar un tema ya creado o incluso crear nuestros propios temas.

```
migrafico <- gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             color = continent, size = pop)) +
  geom_point(alpha = 0.5) +
  labs(title = "GDP vs Esperanza de vida en 2007",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "") +

  geom_text(aes(x = gdpPercap, y = lifeExp, label = country),
           color = "grey50", size=4,
           data = filter(gapminder, year==2007,
                        pop > 1000000000 | country %in% c("Spain", "United States"))) +
  scale_x_log10(limits = c(200, 60000)) +
  scale_size(range = c(0.2, 20),
            # quitamos la leyenda de tamaño
            guide = "none") +
  theme_bw(base_size = 24) +
  #theme_classic(base_size = 24) +
  # Leyenda arriba y ejes grises
  theme(legend.position = "top"
        #,axis.line = element_line(color = "grey85"),
        #axis.ticks = element_line(color = "grey85")
        )+
```

```
guides(color = guide_legend(override.aes = list(size=5)))
show(migrafico)
```

GDP vs Esperanza de vida en



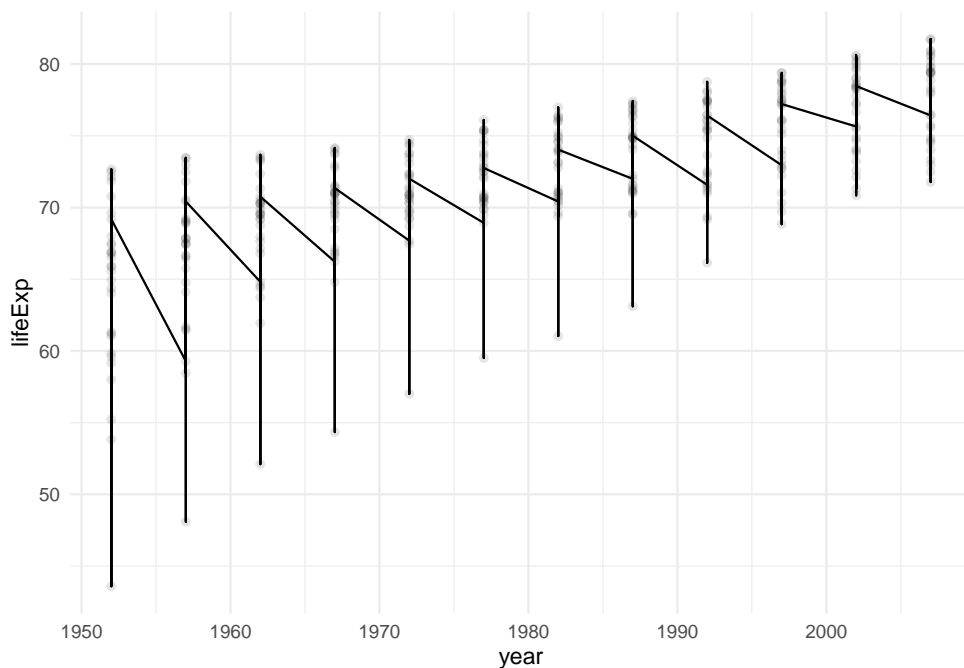
Otras geometrías

Veamos algunas de las geometrías más usadas.

geom_line

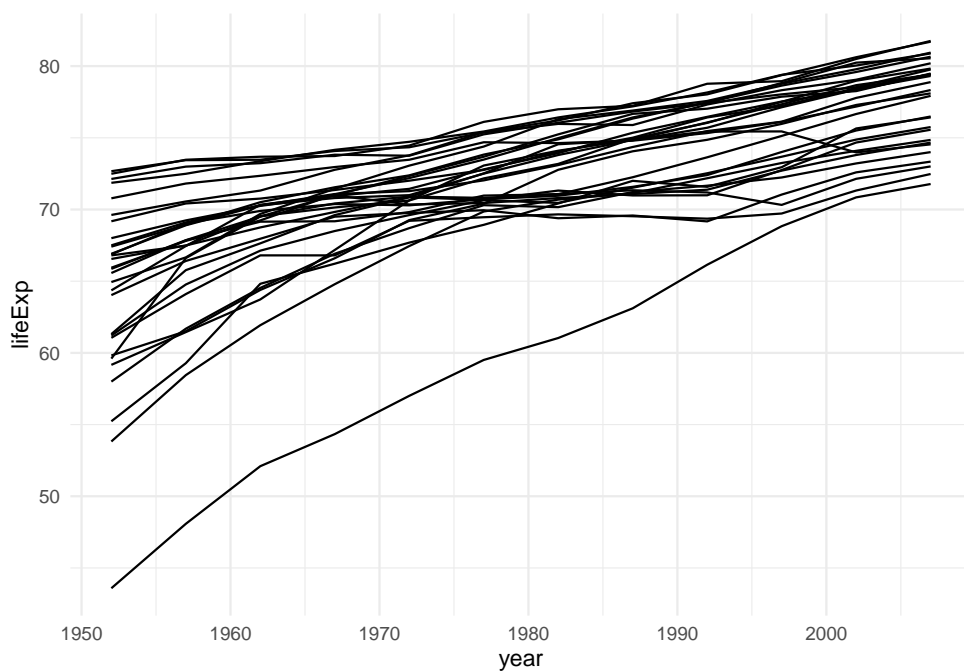
Intentemos estudiar la evolución temporal de la esperanza de vida en los países europeos.

```
gapminder %>%
  filter(continent=="Europe") %>%
  ggplot(aes(x=year,y=lifeExp)) +
    geom_point(alpha=0.1)+
    geom_line()+
    theme_minimal()
```



Obviamente no es el resultado esperado. Esto ocurre porque lo que en realidad queremos es dibujar una línea para cada país mientras que lo que hemos pedido es dibujar una sola línea. La opción `group` de `ggplot2` realizará esta agrupación.

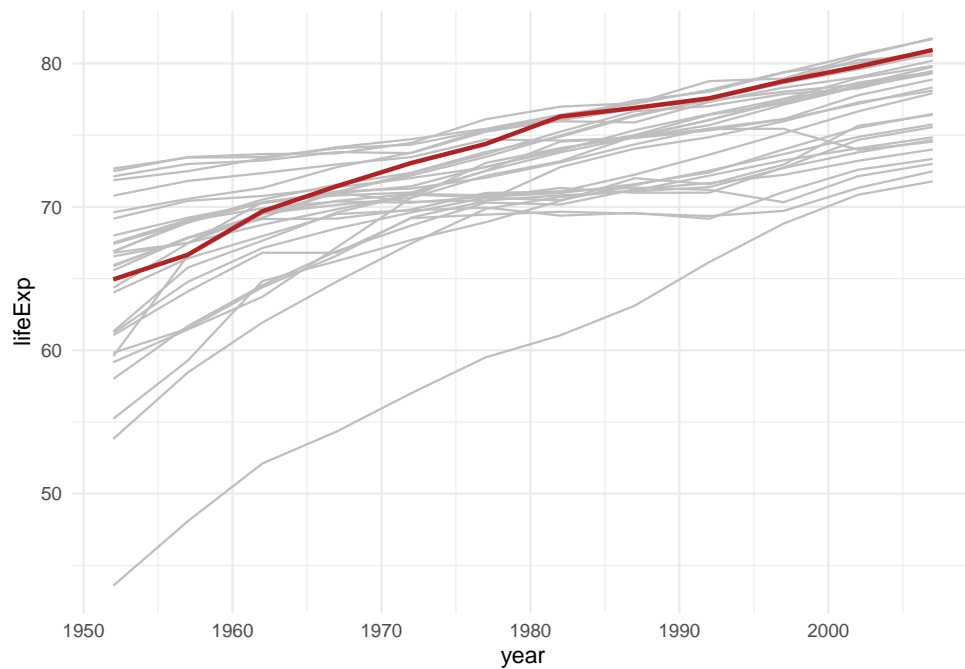
```
gapminder %>%
  filter(continent=="Europe") %>%
  ggplot(aes(x=year,y=lifeExp, group = country)) +
  geom_line()+
  theme_minimal()
```



¿Cuál de ellas representa España?

Podemos filtrar dentro de una geometría.

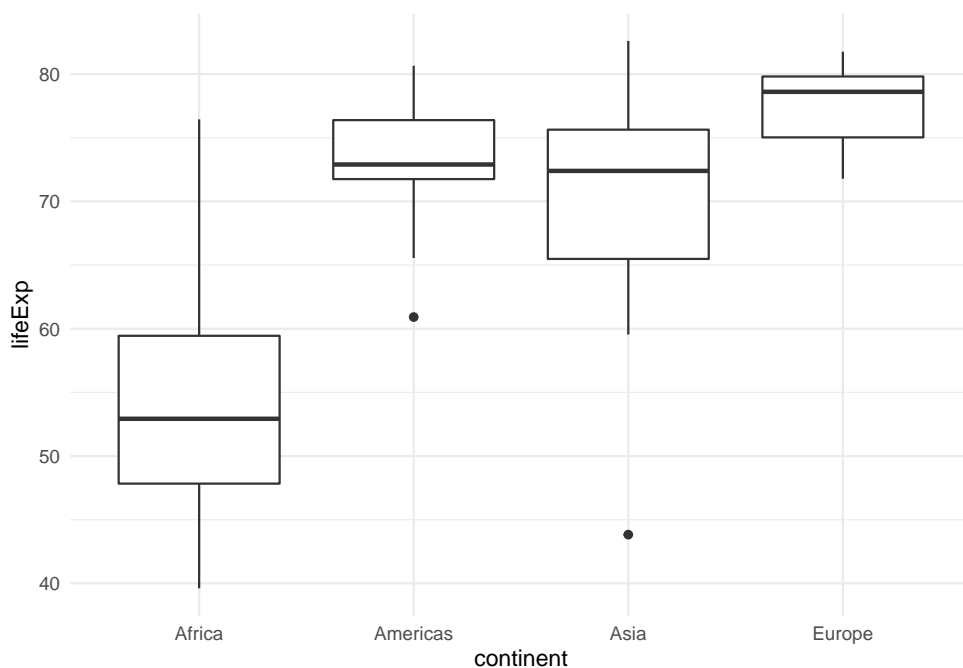
```
gapminder %>%
  filter(continent=="Europe") %>%
  ggplot(aes(x=year,y=lifeExp, group = country)) +
    #geom_point(alpha=0.1)+
    geom_line(color="grey")+
    geom_line(data=filter(gapminder,country=="Spain"),color="firebrick", size=1)+
    theme_minimal()
```



Boxplots

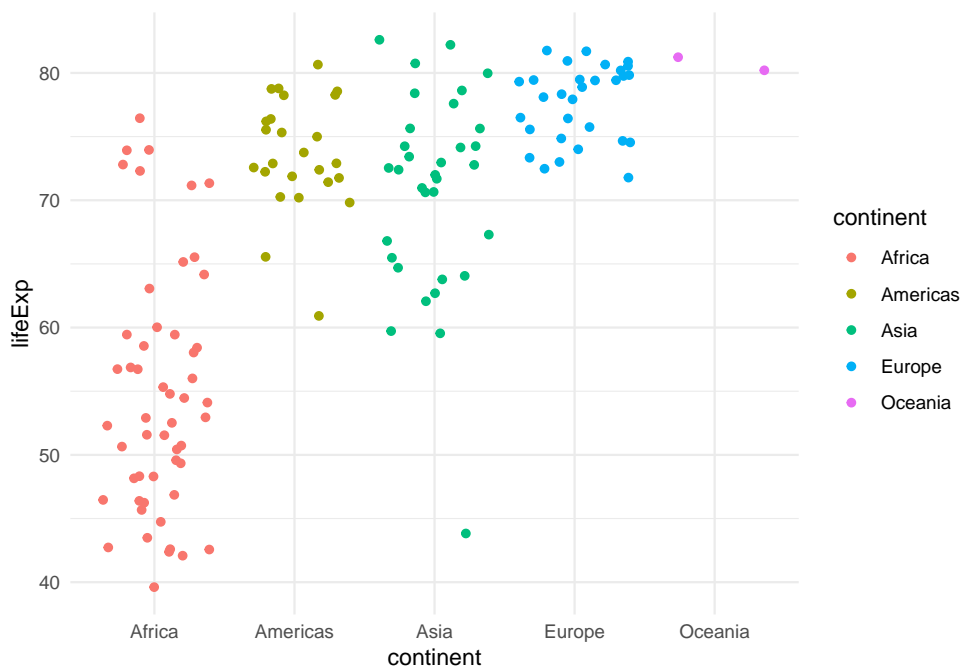
Para comparar variables cualitativas en función de variables cuantitativas, los boxplots son una muy buena herramienta. (Como Oceanía tiene solo dos valores en este gráfico se omite).

```
gapminder %>%
  filter(year==this_year, continent != "Oceania") %>%
  droplevels() %>%
  ggplot(aes(x=continent, y=lifeExp))+
  geom_boxplot()+
  theme_minimal()
```



jitter

```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=continent, y=lifeExp, color=continent))+
  geom_jitter()+
  theme_minimal()
```

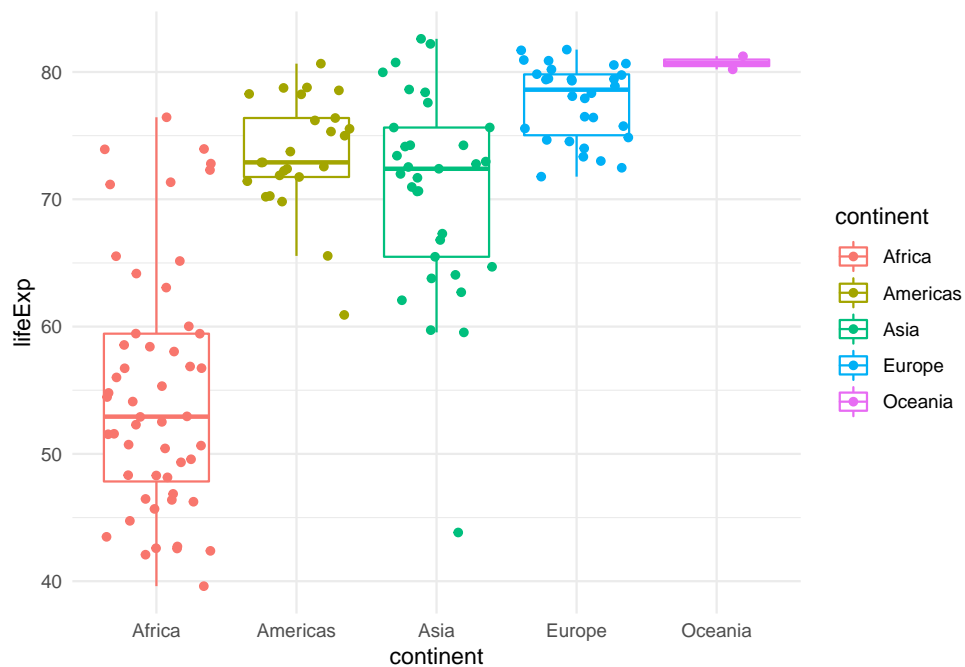


Podemos combinar boxplot y jitter. Nótese cómo hemos quitado los outliers del boxplot con `outlier.shape=NA`, para que no aparezcan dos veces:

```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=continent, y=lifeExp, color=continent))+
```



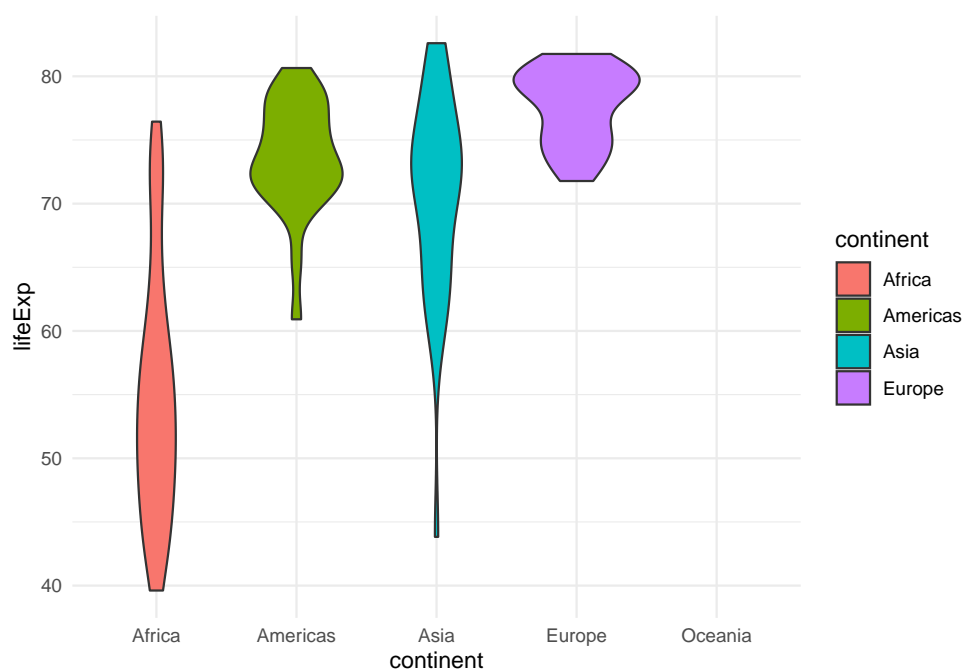
```
geom_boxplot(outlier.shape=NA)+
geom_jitter()+
theme_minimal()
```



Violin Plots

Son una alternativa a los diagramas de cajas que proporcionan la forma de la distribución.

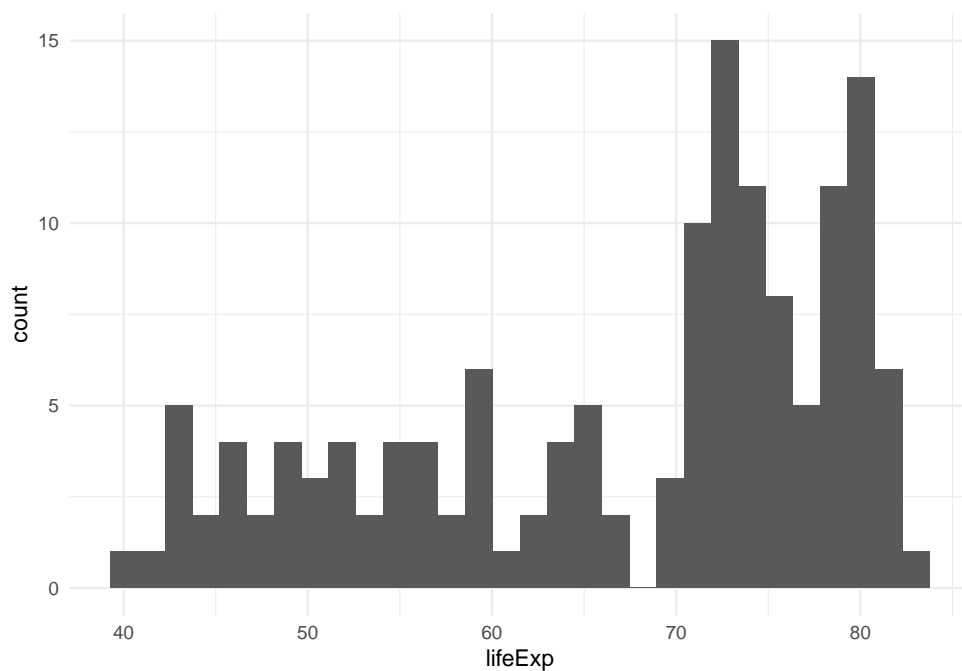
```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=continent, y=lifeExp, fill=continent))+
  geom_violin()+
  theme_minimal()
```



histogramas

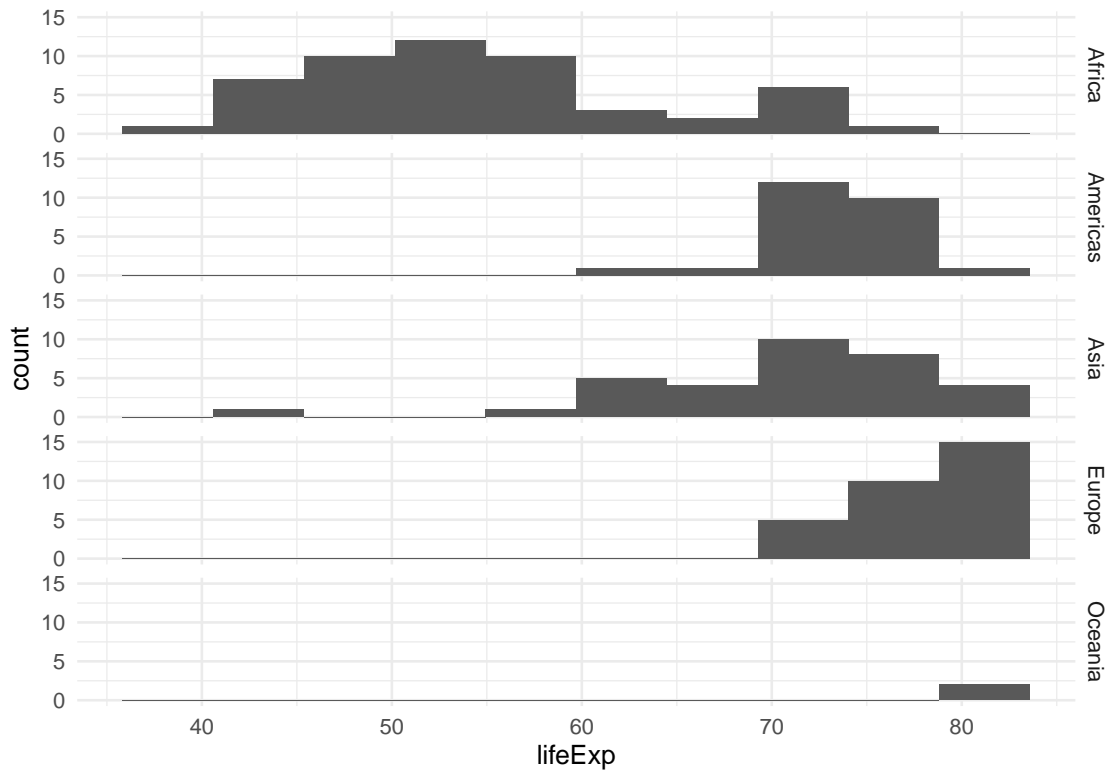
```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=lifeExp))+
  geom_histogram()+
  theme_minimal()
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.



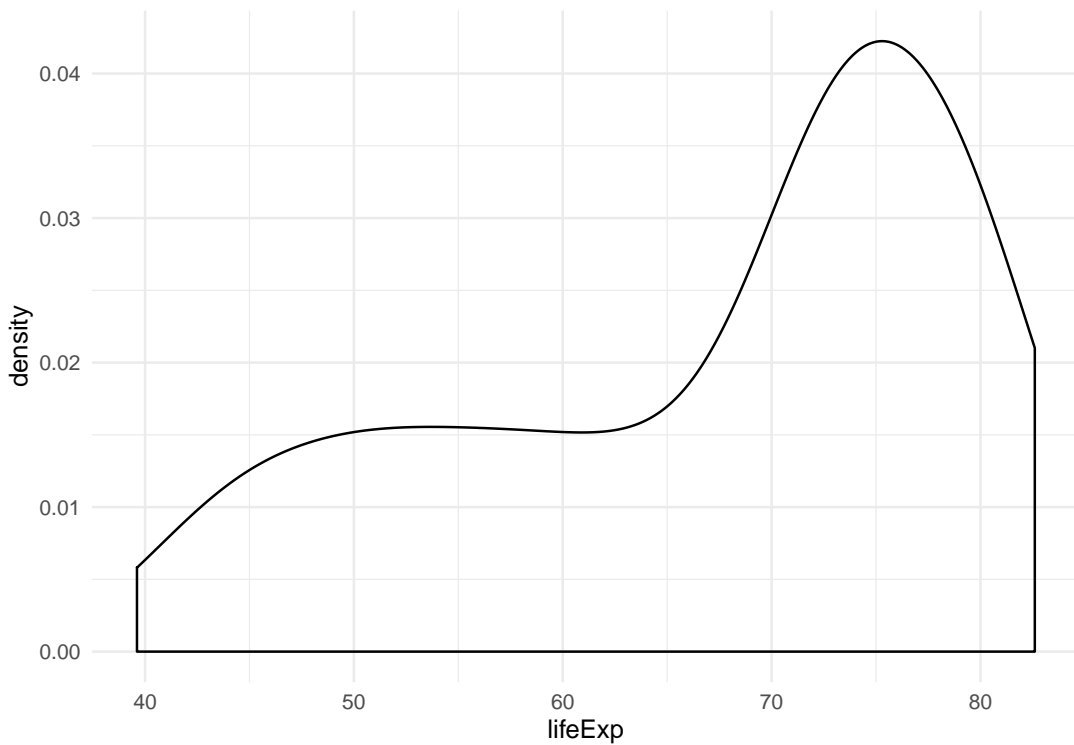
Si queremos comparar las distribuciones podemos hacerlo con facets

```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=lifeExp))+
  geom_histogram(bins=10)+
  facet_grid(continent~.)+
  theme_minimal()
```



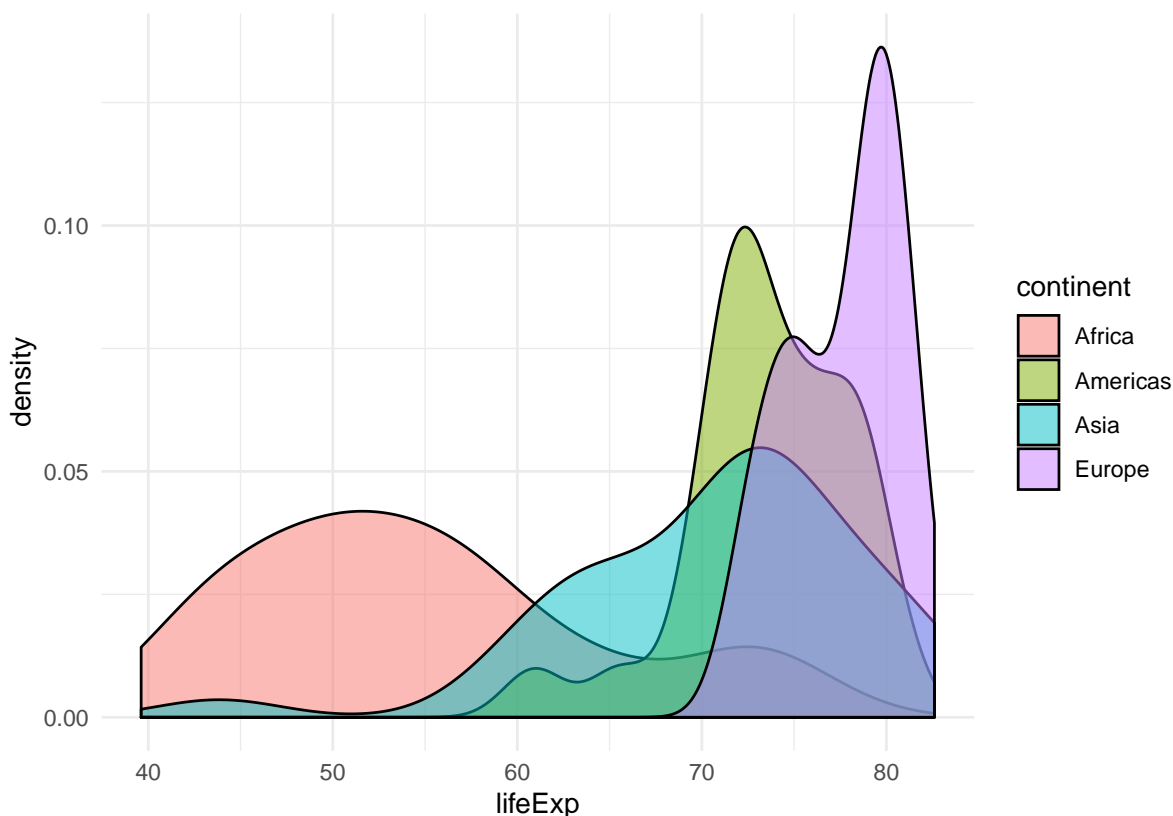
Estimaciones de la curva de densidad

```
gapminder %>%
  filter(year==this_year) %>%
  ggplot(aes(x=lifeExp))+
  geom_density()+
  theme_minimal()
```



Podemos comparar distribuciones con las curvas de densidad y transparencia. De nuevo se omite Oceanía.

```
gapminder %>%
  filter(year==this_year,
         continent!="Oceania") %>%
  droplevels() %>%
  ggplot(aes(x=lifeExp, fill=continent))+
  geom_density(alpha=0.5)+
  theme_minimal()
```

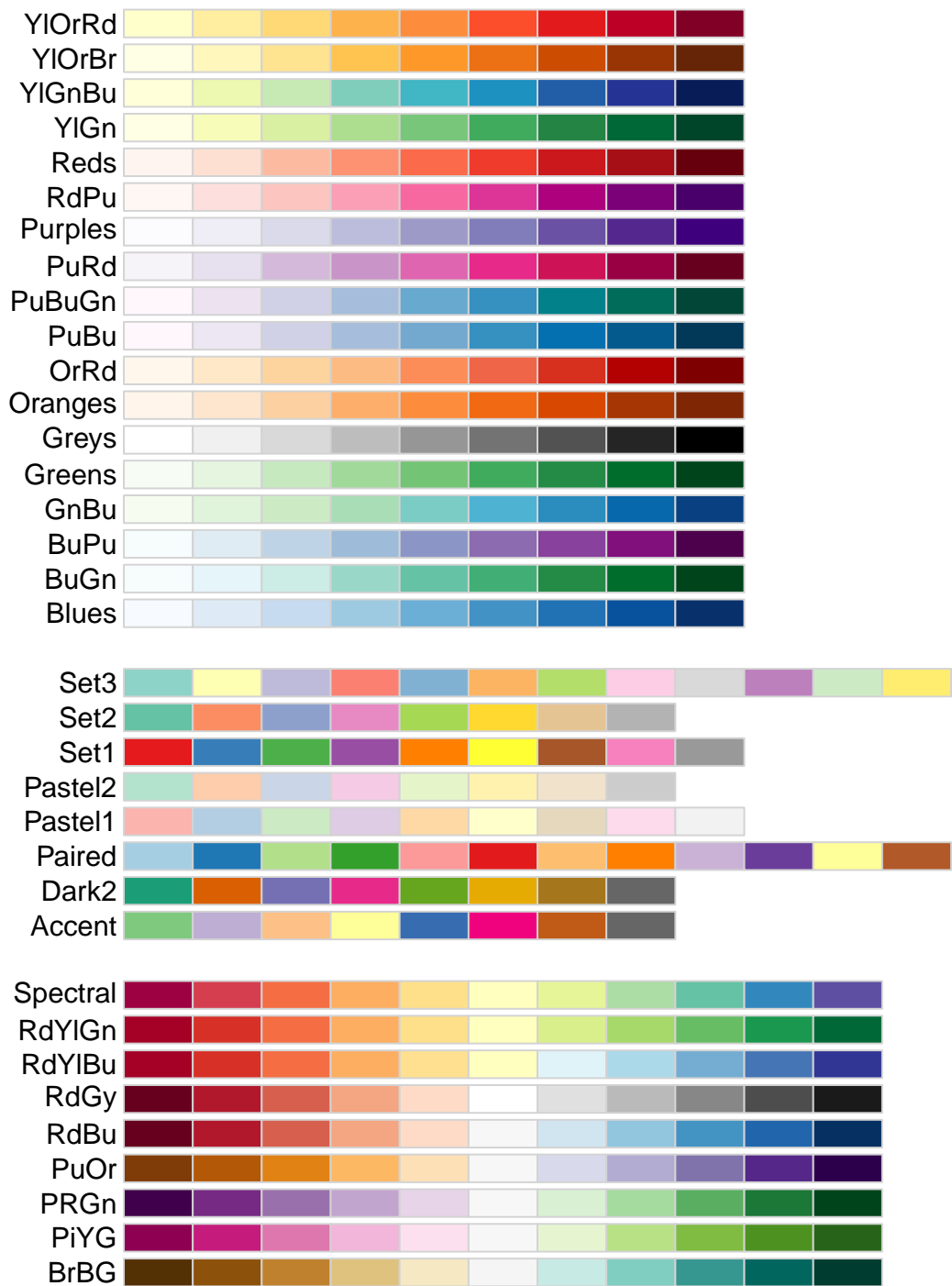


Paletas de colores. El paquete RcolorBrewer

La librería RColorBrewer nos propone tres tipos de paletas fáciles de usar. De arriba a abajo:

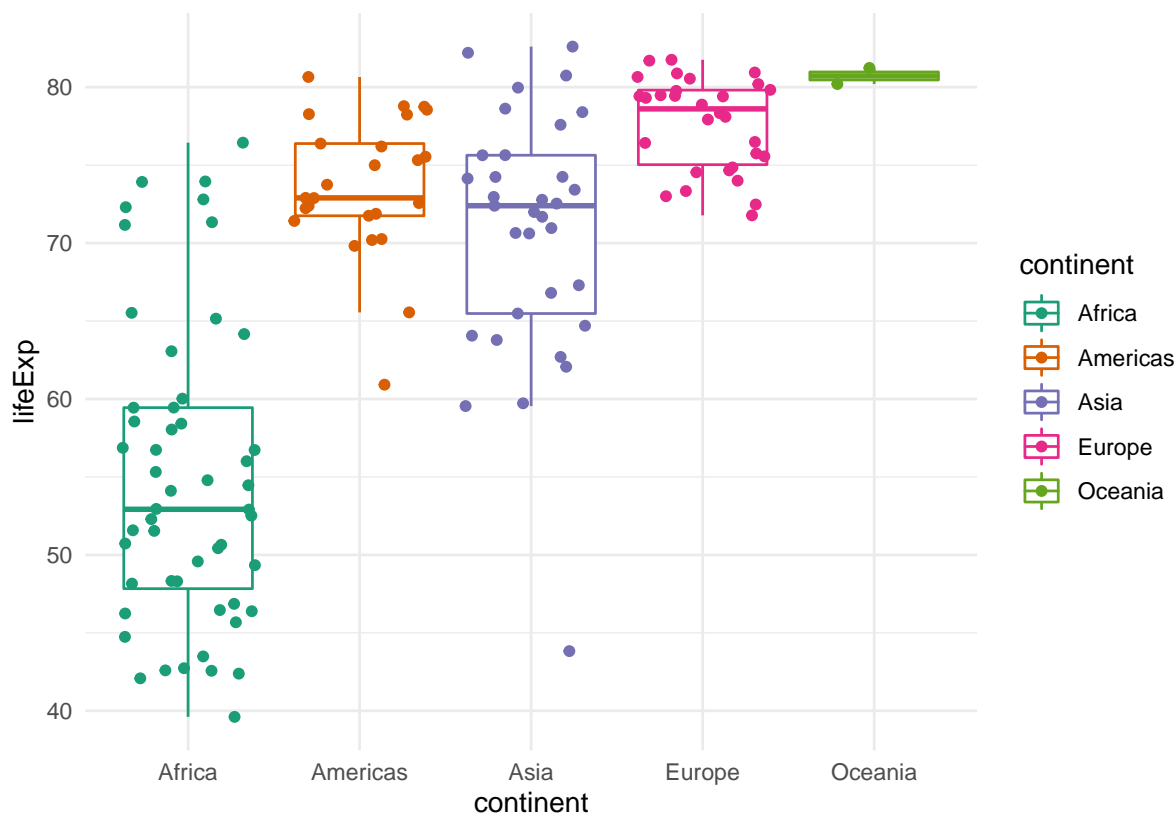
- *Secuencial*: adecuadas para variables numéricas positivas. Enfatizan diferencias entre lo pequeño y lo grande.
- *Cualitativa*: útil para distinguir valores cualitativos.
- *Divergente*: indicadas para variables numéricas que pueden tomar valores positivos y negativos.

```
library(RColorBrewer)
display.brewer.all()
```



```
gapminder %>%
  filter(year==2007) %>%
  ggplot(aes(x=continent, y=lifeExp,color=continent))+
  geom_boxplot(outlier.shape=NA)+
  geom_jitter()+

  scale_color_brewer(type = qual, palette="Dark2")+
  theme_minimal()
```



Cómo guardar gráficos con ggplot2

Hay dos procedimientos principales: usar la ventana gráfica para exportar el gráfico en el formato deseado, o hacerlo dentro de un script con `ggsave`.

```
migraf<- gapminder %>%
  filter(year==2007) %>%
  ggplot(aes(x=continent, y=lifeExp,color=continent))+
  geom_boxplot(outlier.shape=NA)+
  geom_jitter()+

  scale_color_brewer(type = qual, palette="Dark2")+
  theme_minimal()
ggsave(plot=migraf,file="migraf.jpg",height=60,width=120,units="mm",dpi=600)
```

Gráficos interactivos con manipulate

Dentro de Rstudio hay una herramienta rápida para crear gráficos (y análisis) interactivos básicos utilizando el paquete `manipulate`. Se pueden crear aplicaciones interactivas más sofisticadas usando Shiny.

```
library(manipulate)
manipulate(
  gapminder %>%
    filter(year == this_year) %>%
    ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
    geom_point(alpha = 0.5) +
    scale_x_log10() +
    labs(title = "paste(GDP vs Esperanza de vida en", this_year),
         x = "GDP per capita (Escala logarítmica)",
         y = "Esperanza de vida",
         size = "Poblacion (millones)",
         color = "Continente") +
    scale_size(range = c(0.1, 10),
              breaks = 1000000 * c(250, 500, 750, 1000, 1250),
              labels = c("250", "500", "750", "1000", "1250")),
  this_year = slider(min(gapminder$year), max(gapminder$year),
                    step=5, initial = min(gapminder$year))
)
```

html widgets. Exportando gráficos en plotly

plotly es una herramienta en javascript para la realización de gráficos interactivos. Es muy fácil transformar gráficos de `ggplot2` en gráficos de plotly.

```
library(plotly,quietly = TRUE, warn.conflicts = FALSE)
p <- migrafico <- gapminder %>%
  filter(year == this_year) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
            color = continent, size = pop, text=country)) +
  geom_point(alpha = 0.5) +

  labs(title = "",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "") +
  scale_x_log10(limits = c(200, 60000)) +
  scale_size(range = c(0.1, 10),
            guide = "none") +
  theme_classic() +
  theme(legend.position = "top",
        axis.line = element_line(color = "grey85"),
        axis.ticks = element_line(color = "grey85"))
ggplotly(p, tooltip="country")
```

Podemos crear gráficos interactivos con `plotly` incluyendo una estética que no existe en `ggplot` antes de transformarlo `frame=year`.

```
p <- migrafico <- gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             color = continent, size = pop, text=country, frame = year)) +
  geom_point(alpha = 0.5) +
  labs(title = "",
       x = "GDP per capita (Escala logarítmica)",
       y = "Esperanza de vida",
       size = "Poblacion (millones)",
       color = "") +
  scale_x_log10(limits = c(200, 60000)) +
  scale_size(range = c(0.1, 10),
            guide = "none") +
  theme_classic() +
  theme(legend.position = "top",
       axis.line = element_line(color = "grey85"),
       axis.ticks = element_line(color = "grey85"))
ggplotly(p, tooltip="country")
```


Capítulo 7

Markdown y R

Markdown es un lenguaje de edición sencillo. El texto se escribe con un editor cualquiera (se podría usar el Bloc de Notas de Windows), y se incluyen algunos caracteres para formatear el texto. Por ejemplo: una palabra entre asteriscos (así: ****palabra****), saldrá en negrita (**palabra**). A continuación, el texto se pasa por un ‘compilador’ que produce un documento en Word, PDF o HTML. Luego, se puede seguir editando el documento con el programa correspondiente (MS Word, Adobe Reader, Firefox u otro navegador, etc.). El lenguaje Markdown es liviano y sencillo, además de ser completamente gratuito (de código abierto), y por eso se usa mucho para componer páginas Web. Lógicamente, [la página Web del creador de Markdown](#) está hecha con Markdown. Este mismo documento está escrito con Markdown. Y también se pueden crear presentaciones (diapositivas) en distintos formatos.

El lenguaje Markdown permite incluir código escrito en R, que se ejecuta a compilar el archivo de texto y cuyo resultado se muestra en el documento final. Esa característica es extraordinariamente útil. Por ejemplo, al escribir $2 + 2$ de cierta forma se obtiene esto al compilar:

```
2 + 2
```

```
## [1] 4
```

En RStudio está incluido el *compilador* (se llama **Knit**) que compone documentos a partir de un texto en Markdown. **Para crear un documento Markdown en RStudio** basta abrir un archivo R Markdown: *File > New File > RMarkdown . . .*, escoger *Document*, poner título y autor (opcional, se puede añadir más tarde), el tipo de documento que se desee crear (empezar con HTML, que seguro que funciona) y con eso se abre un archivo nuevo en el editor.

La mejor manera de aprender es ir probando. Por ejemplo, escribe debajo de donde pone **## R Markdown** algo así:

```
**Soy Sauce**  
*Hola, Sauce; soy Angel*
```

Guarda el archivo con el nombre que quieras (la extensión *Rmd* se añadirá por defecto). Al hacer *click* en el botón **Knit** se ejecuta el ‘compilador’. Los mensajes del proceso aparecen en la consola *R Markdown* (no hace falta prestarles atención). La salida aparecerá en una ventana nueva.

Si todo ha ido bien, verás esto en el documento resultante:

```
Soy Sauce  
Hola, Sauce; soy Angel
```

A continuación se describe el procedimiento para componer documentos. En primer lugar, se

indica la manera de formatear el texto (por ejemplo, cómo insertar un hipervínculo o crear una tabla). En segundo lugar, se explica cómo incluir comandos de R que se ejecuten al compilar el documento.

Formateo del texto

Markdown es muy simple; la mayoría de las instrucciones que admite se proporcionan a continuación. Si se escribe al pie de la letra esto de aquí abajo, al compilar se obtiene el resultado que se muestra en la página siguiente:

```
Texto sencillo; para saltar de línea, acabarla con dos espacios
*cursiva* (o _cursiva_, da igual), **negrita** o __negrita__, superíndice2,
~~tachado~~, [enlace a mi página Web](http://fisica.unav.es/~angel/)
guión largo: -- , guión más largo: ---, puntos suspensivos ...
Una ecuación en línea:  $A = \pi \times r^2$ . Las ecuaciones se escriben
con el código \LaTeX \ y se puede hacer que ocupen una línea exclusiva:

$$\mu = E(x) = \int_{-\infty}^{\infty} x f(x) dx$$

```

```
# Título 1 (el encabezado de una sección)
```

```
## Encabezado 2 (un encabezado exige un párrafo nuevo)
```

```
### Encabezado 3 (un párrafo se consigue con dos líneas nuevas)
```

```
#### etcétera
```

```
línea horizontal (o salto de diapositiva en presentaciones):
```

```
*****
```

```
Foto de Homer Simpson: ![] (D:\docencia\master\cursoR\clases\3_markdown\HS.png)
```

```
* primer elemento de una lista
```

```
* y este es el segundo
```

```
  + el cual tiene a su vez (_indentado con varios espacios ..._)
```

```
  + otros subapartados (... *o con un par de tabuladores* )
```

```
1. Esta, en cambio, es una lista numerada
```

```
2. item 2
```

```
  + sub-item 1
```

```
  + sub-item 2
```

```
He aquí una tabla:
```

```
Chiste | Malo
```

```
----- | -----
```

```
Casilla 1 | Casilla 2
```

```
Iker | Casillas
```

Tras apretar el botón **Knit**, se obtiene lo siguiente:

Texto sencillo; para saltar de línea, acabarla con dos espacios *cursiva* (o *cursiva*, da igual), **negrita** o **negrita**, superíndice², tachado, [enlace a mi página Web](#) guión largo: – , guión más largo: —, puntos suspensivos ... Una ecuación en línea: $A = \pi \times r^2$. Las ecuaciones se escriben con el código L^AT_EX y se puede hacer que ocupen una línea exclusiva:

$$\mu = E(x) = \int_{-\infty}^{\infty} x f(x) dx$$

Título 1 (el encabezado de una sección)

Encabezado 2 (un encabezado exige un párrafo nuevo)

Encabezado 3 (un párrafo se consigue con dos líneas nuevas)

etcétera

línea horizontal (o salto de diapositiva en presentaciones):



Foto de Homer Simpson:

- primer elemento de una lista
 - y este es el segundo
 - el cual tiene a su vez (*indentado con varios espacios ...*)
 - otros subapartados (*... o con un par de tabuladores*)
1. Esta, en cambio, es una lista numerada
 2. item 2
 - sub-item 1
 - sub-item 2

He aquí una tabla:

Chiste	Malo
Casilla 1	Casilla 2
Iker	Casillas

Cómo insertar código en R

Un comando se puede ejecutar en línea con el texto. Para ello, se debe escribir el comando precedido por `r` (con un espacio detrás) entre acentos abiertos. Para producir esto: el coseno de $\pi/4$ es 0.7071068, lo que se escribió literalmente fue: el coseno de $\pi/4$ es `r cos(pi/4)`. El comando que calcula el coseno se ejecuta al apretar el botón **Knit** que compone el documento.

Si son varios comandos (lo que se llama un *code chunk*), deben escribirse en un párrafo propio, el cual debe empezarse con una nueva línea; en ella se ponen tres acentos abiertos ````` (puede que haya que apretar alguna vez la barra espaciadora para que salgan todos) seguidos de `{r}`; en sucesivas líneas se escriben las instrucciones de R; y en la última línea se ponen otros tres acentos abiertos. Por poner un caso, si se escribe esto (al pie de la letra):

```
``` {r}
x = 3
y = 2
x+y
```
```

al apretar el botón **Knit** se obtiene lo siguiente en el documento generado:

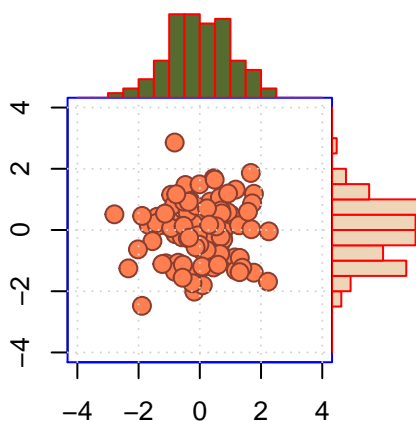
```
x = 3
y = 2
x+y
```

```
## [1] 5
```

(nótese que la salida queda marcada por `##`). Dentro de las llaves de inicio (`{r}`) pueden incluirse varias opciones. Así, `{r , echo = FALSE}` en el encabezado del *chunk* ejecuta el código que viene detrás pero no lo muestra; por ejemplo, para el código precedente da este resultado:

```
## [1] 5
```

Hay varias opciones más que se pueden añadir –consúltese la documentación. Una muy útil es el ancho y alto de las figuras, en pulgadas. A continuación se muestra un gráfico obtenido con las opciones `echo = FALSE, fig.height=2.5, fig.width=2.5, fig.align='center'`:



Encabezado. Tipos de documento.

Al abrir un documento nuevo, RStudio automáticamente incluye un encabezamiento parecido a este:

```
---
title: "Untitled"
author: "A. Garcimartín"
date: "19 de septiembre de 2018"
output: html_document
---
```

y justo a continuación las opciones globales de **Knit** (mejor no tocarlas)

```
``` {r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

En el encabezado, se pueden añadir o quitar líneas (basta añadir un # al comienzo de la línea para que no se tenga en cuenta). Por ejemplo, en este documento el encabezado es:

```
title: "R Markdown"
subtitle: "Documentos científicos que contienen código y gráficos en R"
author: "A. Garcimartín"
# date: "18 de septiembre de 2018"
output:
  pdf_document:
    keep_tex: yes
  html_document: default
  word_document: default

fontsize: 11pt
spacing: double
urlcolor: blue
```

(y esas son solo algunas posibilidades). Saca el autor, pero no la fecha, y se puede compilar con **Knit** para producir un fichero PDF, o bien HTML, o Word. El tamaño de las letras es de 11 puntos, a doble espacio, y los *links* los pone de color azul. Guarda el archivo *.tex para poder editarlo.

Para generar un documento en PDF, es necesario tener instalado L^AT_EX (un sistema de edición profesional, gratuito). Otra opción es generar un documento en Word y guardarlo como PDF (en Word, algunos detalles no funcionan bien, como la alineación de las figuras). Aunque al crear el documento Markdown se haya escogido un cierto formato, basta añadir en el encabezado la línea correspondiente para que se pueda generar otro. El documento se puede abrir luego con otro programa para seguir editándolo, pero es importante *salir de RStudio antes de editar el documento con otro programa*: dos editores con el mismo documento es la receta del desastre.

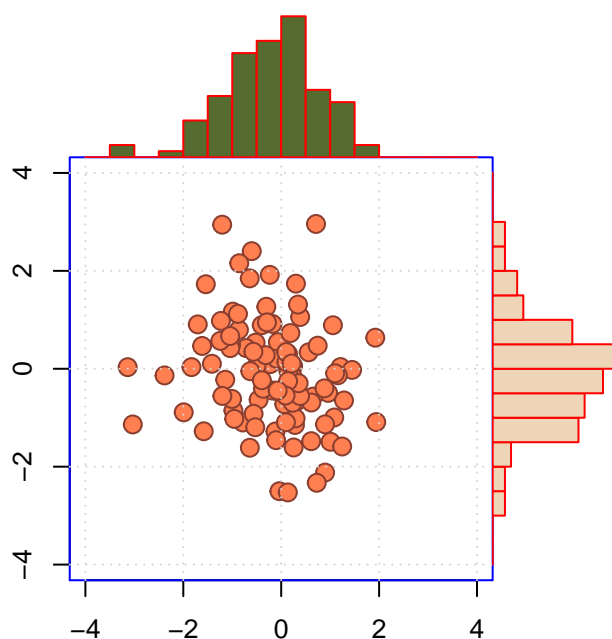
Practica por tu cuenta creando una presentación. Al crear el fichero R Markdown, escoge una presentación HTML. Se crean directamente diapositivas que funcionarán en cualquier ordenador, sin ataduras a un programa específico (como el Power Point). También se puede crear una presentación en PDF (*Beamer*). Como los archivos PDF se pueden combinar, se podrán unir a otras diapositivas creadas con un programa diferente.

Ejemplo: Diagrama de dispersión con histogramas laterales.

```

def.par <- par(no.readonly = TRUE) # para dejar luego todo como estaba
x <- rnorm(100) # 100 puntos de una distribución normal
y <- rnorm(100) # ídem
xhist <- hist(x, breaks = seq(-4,4,0.5), plot = FALSE)
yhist <- hist(y, breaks = seq(-4,4,0.5), plot = FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(-4, 4) # ojo, ¡que coja todos los puntos!
yrange <- c(-4, 4) # si alguno es mayor que 4 el histograma da un error
# layout: en [1,1] va el 2, en [1,2] ninguno, en [2,1] el 1 y en [2,2] el 3
nf <- layout(matrix(c(2,0,1,3),2,2,byrow = TRUE), c(3,1), c(1,3), TRUE)
# dimensiones: ancho de columna, alto de fila: 3 pulgadas, 1 pulgada
par(mar = c(3,3,0,0),col='blue') # márgenes en líneas: abajo, izda., arriba, dcha.
plot(x, y, xlim = xrange, ylim = yrange, xlab = "", ylab = "",
     pch=21,cex=1.5,col='coral4',bg='coral')
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted", lwd = 1)
# el gráfico 2, que va en [1,1]
par(mar = c(0,3,1,0)) # márgenes, en líneas
barplot(xhist$counts, axes = FALSE, ylim = c(0, top), space = 0,
        col='darkolivegreen',border="red")
# el gráfico 3, que va en [2,2]
par(mar = c(3,0,0,1))
barplot(yhist$counts, axes = FALSE, xlim = c(0, top), space = 0,
        horiz = TRUE,col='bisque2',border="red")

```



```

par(def.par) #- reset de los parámetros gráficos originales.

```

Capítulo 8

Inferencia

La inferencia es el proceso por el cual se describe la población a partir de una muestra. Del propio concepto ya se desprende que ese proceso debe realizarse con cuidado. Si una cantidad calculada a partir de la muestra se usa para describir una característica de la población, se llama *estimador*. Cuando el experimento está bien realizado, se espera que el estimador sea bueno, es decir, esté cerca del valor que toma esa variable en la población. Un estimador que consista en un solo valor se denomina *estimador puntual*. Sin embargo, la información que proporciona es incompleta. De un experimento, de una medición, se debe obtener el estimador puntual, un margen de error, y un grado de confianza concedida al valor del estimador obtenido. A eso se le llama un *intervalo de confianza*. El margen de error se puede proporcionar de varias maneras válidas. En cualquier caso, es importante indicar cómo se calcula.

Pensemos en el promedio, por poner un caso. La media de la muestra es un estimador puntual. Como margen de error se podría dar la desviación típica de la muestra: $\bar{x} \pm s$. Es una opción, pero no es lo mismo una desviación típica s obtenida de $n = 10$ mediciones que si se calcula a partir de $n = 10000$ mediciones. Habría que indicar también el tamaño de la muestra n .

Intervalos de confianza

Una manera de expresar el margen de error que toma en cuenta este efecto es indicar el intervalo de confianza como la probabilidad de éxito que tiene el estimador, a lo cual se le llama *nivel de confianza*. Un detalle importante es que para calcular el intervalo de confianza de esta manera se necesitan no solo mediciones de la muestra, sino también conocer la distribución de la variable en la población. Es decir, hay que asumir la hipótesis de que la variable siga en la población cierta distribución de probabilidad (por ejemplo, que esté distribuida normalmente, como una gaussiana).

Si el nivel de confianza es una probabilidad, el suceso complementario tendrá una probabilidad α que se llama *nivel de significación*. Por tanto el nivel de confianza puede escribirse como $1 - \alpha$.

La noción de intervalo de confianza en este contexto se puede resumir así: *si el experimento se repitiera muchas veces tomando muestras independientes, obteniéndose para cada muestra un intervalo de confianza, la fracción de intervalos de confianza que contendría al valor correcto correspondiente a la población sería la proporción $1 - \alpha$.*

Por ejemplo, consideremos el promedio de la muestra, \bar{x} . Por el teorema central del límite, sabemos que las medias de muchas muestras \bar{x} están distribuidas normalmente, con centro en μ y una desviación típica σ/\sqrt{n} . Si llamamos $z_{\alpha/2}^*$ al cuantil de la distribución normal correspondiente

a $\alpha/2$, entonces el intervalo de confianza para la media con un nivel de significación α es

$$\mu \in \bar{x} \pm z_{\alpha/2}^* \cdot \frac{\sigma}{\sqrt{n}}$$

Nótese que si se desconoce σ y se sustituye por la desviación típica de la muestra s , habría que tomar el estadístico t de Student con $n - 1$ grados de libertad:

$$\mu \in \bar{x} \pm t_{\alpha/2, n-1}^* \cdot \frac{s}{\sqrt{n}}$$

Como se ve, en el cálculo del intervalo de confianza interviene el cuantil de una distribución. Ese es el concepto al que se aludió más arriba.

En R, el intervalo de confianza se obtiene al ejecutar un test de hipótesis, lo cual se verá a continuación. Los tests de hipótesis para los que se asume una cierta distribución, se llaman *paramétricos*. En caso de que no se asuma tal cosa, el test se llama *no paramétrico*. Estos últimos suelen tener baja potencia comparados con los tests paramétricos.

Tests de hipótesis

Con un test de hipótesis se da un paso más: se puede obtener una cuantificación de lo lejos que se está de la hipótesis nula. El *p-valor* es la probabilidad de que al tomar una muestra se obtengan valores tan extremos o más que los observados si la hipótesis nula fuera cierta. Un p-valor muy bajo indica que la probabilidad de obtener ese valor es difícilmente compatible con la hipótesis nula, y habría motivos para rechazarla. Un p-valor alto indica que obtener la cantidad que se ha medido no es algo sorprendente si la hipótesis nula es cierta.

Test paramétrico para una media

Si los datos están distribuidos normalmente pero se desconoce la varianza, lo indicado es un test tipo t . El comando `t.test` lleva a cabo esa prueba. Para ello es necesario establecer de antemano un *nivel de confianza*, que se puede considerar como la frecuencia (o proporción) con la que el intervalo de confianza obtenido con la muestra acertaría en la estimación del parámetro. Es decir, si se tomaran infinitas muestras estadísticamente independientes, la proporción de intervalos de confianza que contendrían al verdadero valor vendría dada por el nivel de confianza. El nivel de confianza se puede expresar como $1 - \alpha$, donde α se denomina *nivel de significación*. La sintaxis básica es es:

```
t.test(x, mu = media_ref, conf.level = nivel_conf)
```

donde `x` son los datos de la muestra, `media_ref` es el valor del parámetro contra el que se realiza la prueba, y `nivel_conf` el nivel de confianza. La sintaxis completa, con las opciones, es la siguiente (*cuando se da un valor, es el que se toma por defecto si no se indica*):

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
      mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

donde `y` es un vector numérico de datos (se usa en el caso de pruebas para dos muestras), `alternative` indica si el test es de una o dos colas, `paired` si se trata de muestras emparejadas, y `var.equal` si se puede afirmar la condición de igualdad de varianzas (en caso de dos muestras; en ese caso se toma la varianza conjunta; si no, se emplea la corrección de Welch, o Satterthwaite).

Como R es un lenguaje orientado a objeto, lo que devuelve el procedimiento `t.test` es un *objeto*: en este caso, una *lista* de una determinada *clase* que contiene los siguientes componentes:

- *statistic* .- el valor del estadístico t
- *parameter* .- Los grados de libertad del estadístico
- *p.value* .- el p-valor del test
- *conf.int* .- el intervalo de confianza para la media correspondiente a la hipótesis alternativa especificada
- *estimate* .- la media estimada (o la diferencia de medias si era un test de dos muestras)
- *null.value* .- el valor hipotético de la media (o de la diferencia de medias)
- *alternative* .- una cadena de caracteres que describe la hipótesis alternativa
- *method* .- cadena de caracteres indicando el tipo de test efectuado
- *data.name* .- cadena de caracteres con el nombre de los datos

Por ejemplo, supongamos que la ingesta calórica de 11 chicas en un colegio mayor es:

```
ingesta = c(1258, 1309, 1349, 1478, 1529, 1559, 1628, 1798, 1798, 1969, 2098)
```

y queremos averiguar, con un nivel de confianza del 95 %, si esos datos son compatibles con el nivel de referencia recomendado de 1850 calorías por día. El promedio es 1615.7272727 calorías, pero con ello sólo tenemos un estimador puntual. El test tipo t para contrastar la muestra frente a $\mu_0 = 1850$ sería

```
t.test(ingesta, mu = 1850) # si no se especifica, conf.level = 0.95
```

```
##
## One Sample t-test
##
## data:  ingesta
## t = -2.8431, df = 10, p-value = 0.01746
## alternative hypothesis: true mean is not equal to 1850
## 95 percent confidence interval:
##  1432.127 1799.328
## sample estimates:
## mean of x
##  1615.727
```

Podemos ver cada uno de los elementos del *objeto* que devuelve el procedimiento `t.test` usando el operador `$`, que extrae componentes de una lista:

```
resultado_prueba = t.test(ingesta, mu = 1850) # se guarda el objeto devuelto
names(resultado_prueba) # qué hay en el objeto
```

```
## [1] "statistic"  "parameter"  "p.value"    "conf.int"   "estimate"
## [6] "null.value" "alternative" "method"     "data.name"
```

```
resultado_prueba$statistic # el valor del estadístico t
```

```
##          t
## -2.843088
```

```
resultado_prueba$parameter # grados de libertad (n-1)
```

```
## df
## 10
```

```

resultado_prueba$p.value # p-valor

## [1] 0.01745573
resultado_prueba$conf.int # intervalo de confianza

## [1] 1432.127 1799.328
## attr(,"conf.level")
## [1] 0.95
resultado_prueba$estimate # media muestral

## mean of x
## 1615.727
resultado_prueba>null.value # valor de referencia de la media

## mean
## 1850
resultado_prueba$alternative # si el test es de una o dos colas

## [1] "two.sided"
resultado_prueba$method # qué test se ha realizado

## [1] "One Sample t-test"
resultado_prueba$data.name # los datos usados

## [1] "ingesta"

```

Si solo se desea cierta información, se puede hacer directamente:

```

intervalo = t.test(ingesta, mu=1850, conf.level=0.95)$conf.int # se guarda
print(round(intervalo[1:2], digits=3)) # y se imprime redondeado

## [1] 1432.127 1799.328
t.test(ingesta, mu=1850, conf.level=0.95)$conf.int[1:2] # o se saca directamente

## [1] 1432.127 1799.328

```

Si se sospecha que las chicas comen *menos* de lo que debieran y lo que se desea es saber si la media está **por debajo**, es un test de una cola:

```

t.test(ingesta, mu=1850, conf.level=0.95, alternative="less")

##
## One Sample t-test
##
## data: ingesta
## t = -2.8431, df = 10, p-value = 0.008728
## alternative hypothesis: true mean is less than 1850
## 95 percent confidence interval:
## -Inf 1765.076
## sample estimates:
## mean of x
## 1615.727

```

El p-valor y el intervalo de confianza son mucho más restrictivos; pero no es honrado cambiar a un test de una cola *a posteriori*, después de conocer el resultado.

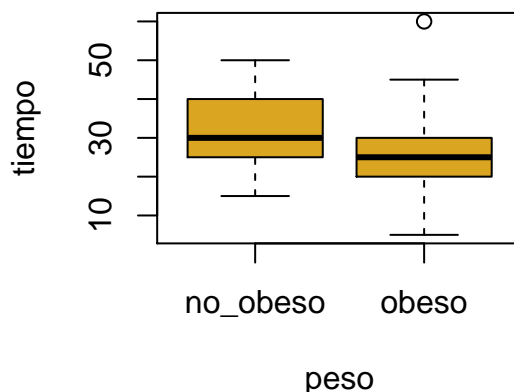
Comparación de dos medias

Varias circunstancias marcan el tipo de test: si las muestras proceden o no de una población distribuida normalmente, si las varianzas son iguales, y si las muestras están emparejadas o son independientes. En lo que sigue asumiremos normalidad (en caso contrario, se recurre a un test no paramétrico).

Muestras independientes.

El gerente de un hospital quiere estudiar las quejas de los pacientes sobre un médico, al que acusan de dedicar menos tiempo a las personas obesas. Se mide el tiempo que dedica el médico a una serie de pacientes, y la apreciación de si son obesos o no. Los datos se recogen en el fichero **tiempopeso.txt**, que contiene el *Tiempo* y el *Peso* (codificada como un factor: 1="no_obeso", 2 = "obeso").

```
tiempo_peso = read.table("tiempopeso.txt",header=TRUE)
tiempo_peso$Peso = factor(tiempo_peso$Peso, levels = 1:2,
  labels = c("no_obeso", "obeso")) # se define como factor
peso=tiempo_peso$Peso # para no estar todo el tiempo repitiendo tiempo_peso$
tiempo=tiempo_peso$Tiempo
plot(tiempo~peso, col = 'goldenrod')
```



Nótese que si x es una variable cualitativa, $\text{plot}(y \sim x)$ produce un diagrama de cajas.

Test de igualdad de varianzas.- Compara la razón de varianzas.

```
var.test(tiempo ~ peso)
```

```
##
## F test to compare two variances
##
## data: tiempo by peso
## F = 1.0443, num df = 32, denom df = 37, p-value = 0.8931
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
## 0.5333405 2.0797269
```

```
## sample estimates:
## ratio of variances
##          1.044316
```

En este caso la sintaxis del contraste de hipótesis con un test tipo t sería la siguiente (nótese que las variables se pasan como $y \sim x$; y que por defecto `var.equal = FALSE`)

```
t.test(tiempo~peso, mu=0, conf.level=0.95, var.equal=TRUE)
```

```
##
## Two Sample t-test
##
## data: tiempo by peso
## t = 2.856, df = 69, p-value = 0.005663
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.997955 11.255633
## sample estimates:
## mean in group no_obeso    mean in group obeso
##          31.36364          24.73684
```

Muestras emparejadas.

Estas muestras se suelen encontrar en situaciones donde se desea comprobar si la media ha cambiado *antes* y *después* de un tratamiento. Por tanto, los datos corresponden a los mismos individuos analizados por duplicado, en ambas situaciones (las muestras están *emparejadas*). Para el contraste de hipótesis se puede hallar la diferencia entre las dos situaciones y realizar un test tipo t, o bien directamente realizar el test con las dos mediciones, especificando que se trata de muestras emparejadas.

Por ejemplo, se registró el número de accidentes de tráfico durante un año en nueve tramos de carretera antes y después de colocar señales indicadoras de peligro; el contraste de hipótesis se hace así:

```
Antes = c(3,3,2,2,4,3,2,2,1)
Después = c(3,2,1,1,2,1,1,1,2) # el número de accidentes en los mismos tramos
Diferencia = Después - Antes
t.test(Diferencia, mu=0, conf.level=0.95) # test t con la diferencia
```

```
##
## One Sample t-test
##
## data: Diferencia
## t = -2.8737, df = 8, p-value = 0.02071
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## -1.6021826 -0.1755951
## sample estimates:
## mean of x
## -0.8888889
```

```
t.test(Antes,Después, paired=TRUE, conf.level=0.95) # con datos emparejados
```

```
##
```

```
## Paired t-test
##
## data: Antes and Después
## t = 2.8737, df = 8, p-value = 0.02071
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.1755951 1.6021826
## sample estimates:
## mean of the differences
##                0.8888889
```

Tests no paramétricos para las medias

Si la muestra no proviene de una distribución normal, no se puede llevar a cabo el test tipo t: hay que recurrir a un test no paramétrico. Téngase en cuenta que estos tests suelen tener menos potencia que los paramétricos. Se proponen los tests con los mismos ejemplos indicados más arriba.

Test no paramétrico para una media: test de Wilcoxon

```
wilcox.test(ingesta, mu = 1850)
```

```
## Warning in wilcox.test.default(ingesta, mu = 1850): cannot compute exact p-
## value with ties
##
## Wilcoxon signed rank test with continuity correction
##
## data: ingesta
## V = 8, p-value = 0.0293
## alternative hypothesis: true location is not equal to 1850
```

Test no paramétrico para muestras independientes: test de suma de rangos de Wilcoxon (equivalente al test U de Mann-Whitney)

```
wilcox.test(tiempo~peso, conf.level = 0.95)
```

```
## Warning in wilcox.test.default(x = c(15L, 15L, 45L, 40L, 45L, 20L, 40L, :
## cannot compute exact p-value with ties
##
## Wilcoxon rank sum test with continuity correction
##
## data: tiempo by peso
## W = 866, p-value = 0.003985
## alternative hypothesis: true location shift is not equal to 0
```

Test no paramétrico para muestras emparejadas: test de suma de rangos de Wilcoxon

```
wilcox.test(Antes, Después, paired = TRUE, conf.level = 0.95)
```

```
## Warning in wilcox.test.default(Antes, Después, paired = TRUE, conf.level =
## 0.95): cannot compute exact p-value with ties

## Warning in wilcox.test.default(Antes, Después, paired = TRUE, conf.level =
## 0.95): cannot compute exact p-value with zeroes

##
## Wilcoxon signed rank test with continuity correction
##
## data: Antes and Después
## V = 32.5, p-value = 0.04007
## alternative hypothesis: true location shift is not equal to 0
```

Test de proporciones

Para los datos categóricos (“factores”, en R), se pueden hacer contrastes de hipótesis con las proporciones (el número de veces que aparece una determinada variable cualitativa).

Tests para una proporción

El test exacto es el binomial (`binom.test`); pero si los números implicados son demasiado grandes, se puede usar `prop.test`, que para hacer los cálculos recurre a la aproximación normal para la binomial. Si el número de ensayos es grande (típicamente, si $n > 100$) es una buena aproximación.

Supongamos que al lanzar una moneda 1000 veces obtenemos 900 caras. Sospechamos que la moneda está trucada (hipótesis nula: $p = 1/2$; hipótesis alternativa: $p \neq 1/2$). Se puede aceptar o rechazar la hipótesis nula con un test de proporciones (por defecto, de dos colas):

```
result_ptest = prop.test(900,1000, conf.level=0.95)
names(result_ptest)
```

```
## [1] "statistic" "parameter" "p.value" "estimate" "null.value"
## [6] "conf.int" "alternative" "method" "data.name"
```

```
result_ptest
```

```
##
## 1-sample proportions test with continuity correction
##
## data: 900 out of 1000, null probability 0.5
## X-squared = 638.4, df = 1, p-value < 2.2e-16
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
## 0.8793091 0.9175476
## sample estimates:
## p
## 0.9
```

como se ve, el *objeto* que devuelve `prop.test` es de la misma *clase* y tiene los mismos componentes que el que devuelve `t.test`. Esa *clase* se llama `htest` y es similar para casi todos los tests. Sus elementos se pueden extraer con `$` y con `[]`.

El test exacto es `binom.test`:

```
binom.test(900,1000, conf.level=0.95) # compárese con el resultado anterior
```

```
##
## Exact binomial test
##
## data: 900 and 1000
## number of successes = 900, number of trials = 1000, p-value <
## 2.2e-16
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
## 0.8797121 0.9178947
## sample estimates:
## probability of success
## 0.9
```

Nótese que un número tan pequeño como 10^{-16} puede estar cerca del error numérico del microprocesador.

Los test de proporciones se pueden ejecutar directamente sobre tablas. *Ejemplo.*- De los 145 miembros electos de una corporación, 88 son hombres y 57 mujeres. ¿Cuál es la probabilidad de que esta diferencia sea debida al azar?

```
sexo = factor(c(rep(1,88),rep(2,57)),levels = 1:2,
              labels = c("hombre","mujer")) # definición del sexo
table(sexo)
```

```
## sexo
## hombre  mujer
##      88      57
```

```
prop.test(table(sexo),conf.level=0.95)
```

```
##
## 1-sample proportions test with continuity correction
##
## data: table(sexo), null probability 0.5
## X-squared = 6.2069, df = 1, p-value = 0.01273
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
## 0.5221451 0.6858942
## sample estimates:
## p
## 0.6068966
```

Test para varias proporciones

Supongamos el caso de las notas de un curso dividido en el grupo de Inglés y Español. En el primero, 14 estudiantes de 38 sacaron sobresaliente; en el otro grupo, hubo 10 sobresalientes de 40 estudiantes. Se desea contrastar la hipótesis nula de que las proporciones son idénticas. Se hace así:

```
sobresalientes = c(14,10)
n_estudiantes = c(38,40)
prop.test(sobresalientes, n_estudiantes, conf.level = 0.95)
```

```
##
## 2-sample test for equality of proportions with continuity
## correction
##
## data: sobresalientes out of n_estudiantes
## X-squared = 0.7872, df = 1, p-value = 0.3749
## alternative hypothesis: two.sided
## 95 percent confidence interval:
## -0.1110245  0.3478666
## sample estimates:
##   prop 1    prop 2
## 0.3684211 0.2500000
```

Nótese que los vectores pueden tener cualquier longitud (o sea, que se puede comparar el número de sobresalientes en muchos grupos, no solo en 2).

Tablas de contingencia

Se desea estudiar la asociación de variables cualitativas (o *factors*, tal como se denominan en R). Consideraremos al principio dos variables binarias (*Yes / No*); añadir variables y modalidades no cambia los conceptos. Se pueden considerar dos situaciones: aquellas en las que los *totales marginales* no están predefinidos, para las que se aplica un *test de independencia*; y aquellas en las que los *totales marginales* están definidos, en cuyo caso se aplica un *test de homogeneidad*. En el primer caso, de cada individuo se toman dos mediciones, una para cada factor; en el segundo caso, los individuos han sido escogidos según la modalidad de uno de los factores, de modo que sólo se les mide el otro. El estadístico de contraste es el mismo en ambos casos. A continuación se exponen sendos ejemplos.

Test de independencia

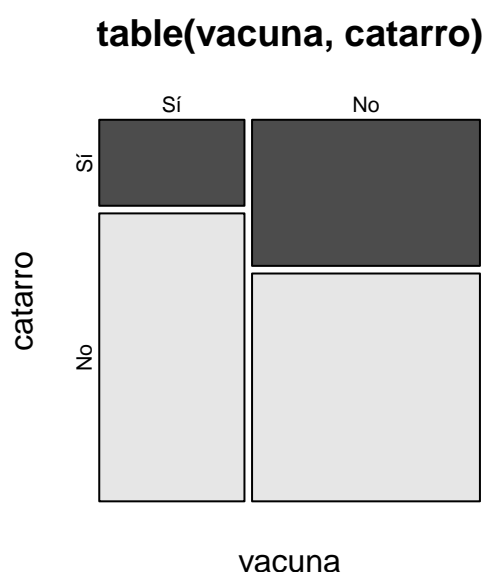
En una investigación sobre la eficiencia de una vacuna contra el resfriado, se toma una muestra aleatoria de individuos y se les pregunta (1) si han padecido catarro o no en el último año, y (2) si habían sido vacunados o no. Los resultados son los siguientes: 231 personas tuvieron catarro, de las cuales 63 habían sido vacunadas; y 473 personas no padecieron ningún catarro, de las cuales 211 habían sido vacunadas.

```
catarro = c(rep(1,231),rep(2,473)) # vector numérico (1 y 2)
catarro = factor(catarro, levels=1:2,labels=c("Sí","No")) # factor
# nótese que cada posición del vector es una persona, debe ponerse en orden:
# catarro = 1 1 ... 1 1 ... 2 2 2 ... 2 2 2 2 2
# vacuna = 1 1 ... 2 2 ... 1 1 1 ... 2 2 2 2 2
vacuna = (c(rep(1,63),rep(2,168),rep(1,211),rep(2,262)))
vacuna = factor(vacuna,levels=1:2,labels=c("Sí","No"))
table(vacuna,catarro)
```

```
##          catarro
## vacuna  Sí  No
##      Sí  63 211
##      No 168 262
```



```
par_orig = par(mai=c(0.7,0.7,0.7,0.6))
mosaicplot(table(vacuna,catarro), color = TRUE )
```



```
par(par_orig)
```

El test de independencia se lleva a cabo con `chisq.test`, cuya sintaxis completa es

```
chisq.test(x, y = NULL, correct = TRUE,
           p = rep(1/length(x), length(x)), rescale.p = FALSE,
           simulate.p.value = FALSE, B = 2000)
```

donde `x` son los datos, `correct` se refiere a la corrección de continuidad, `p` se refiere a las probabilidades de cada elemento de `x` (el contraste, por defecto, asume que la distribución es uniforme) y si se debe encontrar un `p`-valor mediante simulación de Montecarlo, con `B` réplicas.

```
restchisq = chisq.test(table(vacuna,catarro),correct = FALSE)
restchisq
```

```
##
## Pearson's Chi-squared test
##
## data: table(vacuna, catarro)
## X-squared = 19.621, df = 1, p-value = 9.44e-06
```

Es interesante notar que este test devuelve además los casos observados y los esperados (si fuera cierto que no hay asociación):

```
restchisq$observed
```

```
##      catarro
## vacuna  Sí  No
##      Sí  63 211
##      No 168 262
```

```
restchisq$expected
```

```
##          catarro
## vacuna      Sí      No
##   Sí  89.90625 184.0938
##   No 141.09375 288.9062
```

En el caso de que en alguna casilla haya pocas observaciones (menos de 5, por poner un número) lo indicado sería un test de Fisher:

```
fisher.test(table(vacuna,catarro))
```

```
##
## Fisher's Exact Test for Count Data
##
## data:  table(vacuna, catarro)
## p-value = 8.416e-06
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.3251334 0.6632077
## sample estimates:
## odds ratio
##  0.4661334
```

Test de homogeneidad

Se realizó una pregunta a la salida de un supermercado para conocer las preferencias de hombres y mujeres. Se escogieron 25 mujeres y 17 hombres a los que se preguntó si preferían los huevos (a) fritos, (b) revueltos, o (c) cocidos. Nótese que al haberse escogido de antemano el número de mujeres y hombres esos totales marginales están fijados y solo se les hace una pregunta a los sujetos.

```
mitabla=matrix(c(5,9,12,3,7,5),ncol=3)
colnames(mitabla)=c("fritos","revueltos","cocidos")
rownames(mitabla)=c("Mujer","Hombre")
mitabla
```

```
##          fritos revueltos cocidos
## Mujer      5         12         7
## Hombre     9          3         5
```

```
mosaicplot(mitabla, color=TRUE)
```

mitabla

| | Mujer | Hombre |
|-----------|-------|--------|
| fritos | 5 | 5 |
| revueltos | 5 | 5 |
| cocidos | 5 | 5 |

```
chisq.test(mitabla)
```

```
## Warning in chisq.test(mitabla): Chi-squared approximation may be incorrect
##
## Pearson's Chi-squared test
##
## data:  mitabla
## X-squared = 5.8516, df = 2, p-value = 0.05362
```

Como hay dos casillas con 5 casos o menos, se indica en una advertencia que los resultados pueden ser incorrectos. Se puede llevar a cabo el método de Montcarlo para encontrar el p-valor:

```
chisq.test(mitabla, simulate.p.value = TRUE, B = 10000)
```

```
##
## Pearson's Chi-squared test with simulated p-value (based on 10000
## replicates)
##
## data:  mitabla
## X-squared = 5.8516, df = NA, p-value = 0.05949
```

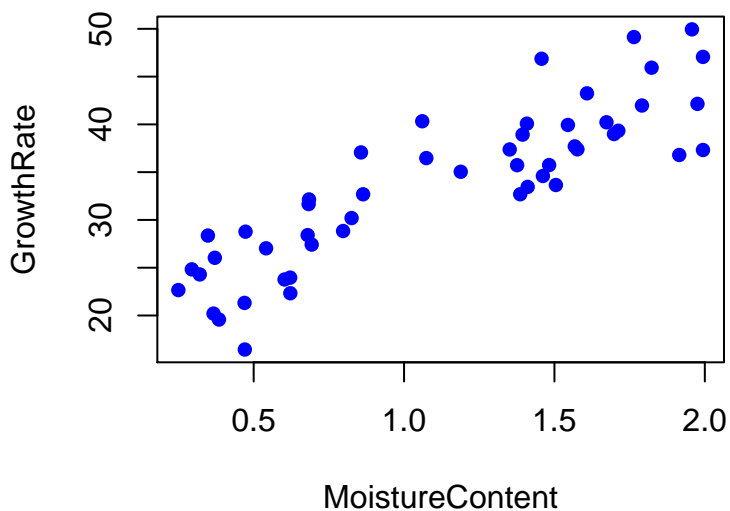
Capítulo 9

Regresión Lineal

Considérense los datos del archivo `plant_data.csv`, que contiene la humedad del suelo junto a la tasa de crecimiento de las plantas (no prestaremos atención a las unidades) en sendas columnas:

```
plant_data <- read.csv("plant_data.csv")
GrowthRate = plant_data$plant.growth.rate
MoistureContent = plant_data$soil.moisture.content
```

Se desea establecer si existe una relación lineal entre ambas variables (`MoistureContent`, que es la variable independiente, o *predictor*, y `GrowthRate`, que es la variable dependiente, o *respuesta*). El primer paso es representar los datos para hacerse una idea.



Ajuste lineal

El comando `lm` proporciona un *modelo*, un *ajuste* lineal de los datos:

```
mi_ajuste = lm(GrowthRate ~ MoistureContent)
mi_ajuste
```

```
##
## Call:
```

```
## lm(formula = GrowthRate ~ MoistureContent)
##
## Coefficients:
##      (Intercept)  MoistureContent
##           19.35           12.75
names(mi_ajuste) # este objeto contiene mucha más información
```

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

Véase la ayuda de `lm` para una explicación de cada componente; se puede obtener un resumen con

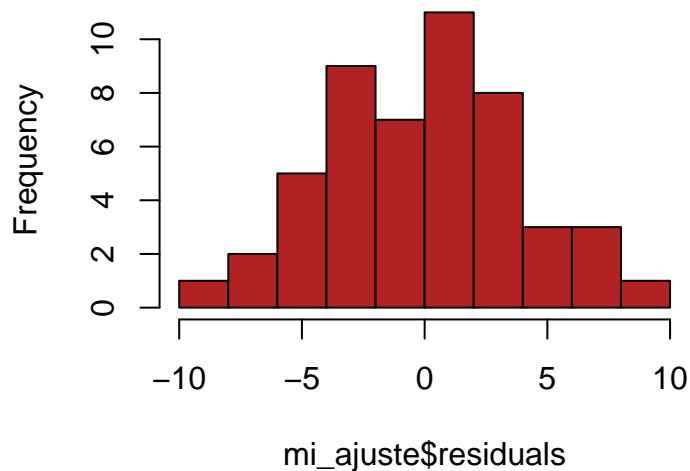
```
summary(mi_ajuste)
```

```
##
## Call:
## lm(formula = GrowthRate ~ MoistureContent)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.9089 -3.0747  0.2261  2.6567  8.9406
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    19.348     1.283   15.08 <2e-16 ***
## MoistureContent 12.750     1.021   12.49 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.019 on 48 degrees of freedom
## Multiple R-squared:  0.7648, Adjusted R-squared:  0.7599
## F-statistic: 156.1 on 1 and 48 DF,  p-value: < 2.2e-16
```

pero `mi_ajuste` contiene más información; por ejemplo, `residuals` son los residuos. Para que el ajuste por mínimos cuadrados sea válido, los residuos deberían estar distribuidos normalmente:

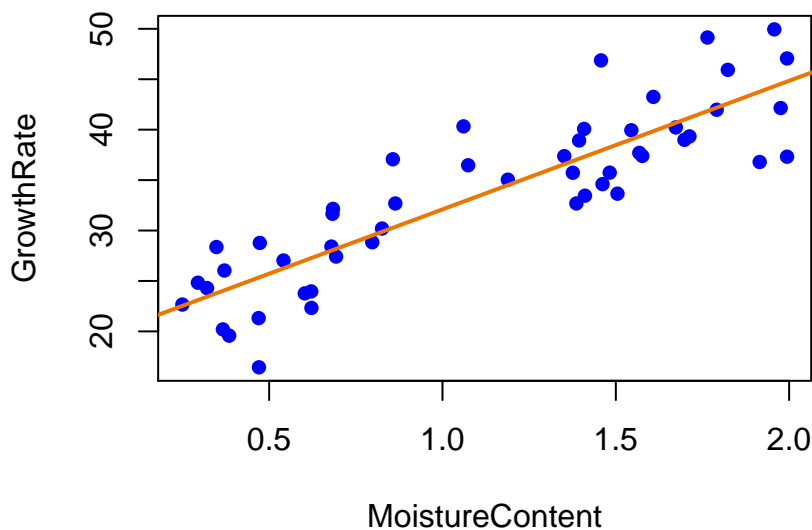
```
hist(mi_ajuste$residuals, col='firebrick', main = "distribución de residuos")
```

distribución de residuos



Una de las particularidades de un lenguaje de programación orientado a objeto es que los métodos admiten entradas muy flexibles; así, `abline` puede gestionar directamente el objeto `mi_ajuste` y dibuja la línea correspondiente:

```
plot( GrowthRate ~ MoistureContent, pch=16, col='blue')
abline(mi_ajuste,col='darkorange2',lwd=2)
```



```
coeficientes=mi_ajuste$coefficients
coeficientes
```

```
##      (Intercept) MoistureContent
##      19.34846      12.74954
```

Por tanto, el ajuste lineal es $\mu_{Y|x} = 19,35 + 12,75 \cdot x$ (los números están redondeados), donde x es el predictor (`MoistureContent`) e y la respuesta (`GrowthRate`) que se desea predecir.

Fórmulas

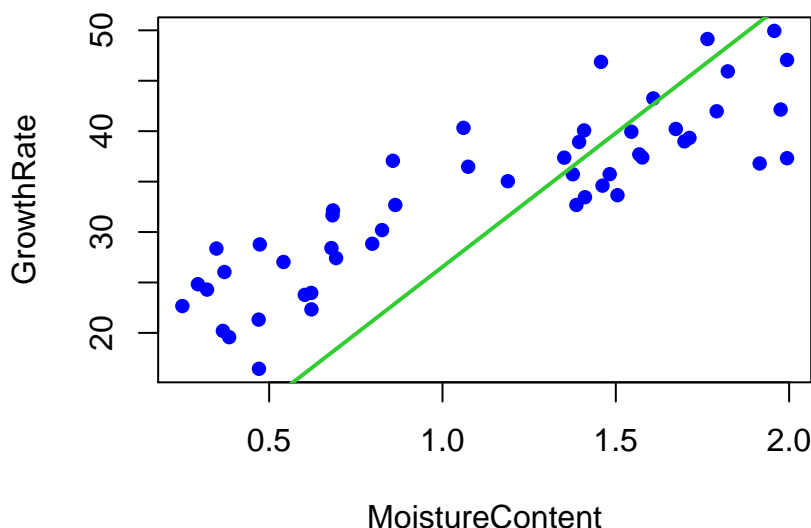
En el argumento de `lm` se puede incluir la fórmula para el ajuste lineal, que de manera general se puede escribir $y = a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_n \cdot x_n + b$; el ajuste halla los coeficientes $\{a_i\}$ y b que minimizan la suma de las distancias al cuadrado desde la recta y a los datos. La manera de especificar el ajuste a través de una fórmula es un poco enrevesada en R. Para empezar, el ajuste de una recta $y = a \cdot x + b$ se calcula como dos números a y b que multiplican respectivamente a x y a 1. En R eso se interpreta como que el ajuste es $y \sim x + 1$. La sintaxis $y \sim x$ se traduce internamente a $y \sim x + 1$.

En las fórmulas, los operadores aritméticos tienen un sentido diferente. Por ejemplo, el `-` significa *excluir* un término del ajuste. Si se desea ajustar una recta que pase por el origen (es decir: $y = a \cdot x$) hay que escribir explícitamente $y \sim x - 1$, que se leería “ajustar el coeficiente de x pero excluir la ordenada en el origen” (si se pusiera solo $y \sim x$, por defecto se incluye la ordenada en el origen).

```
ajuste0 = lm(GrowthRate ~ MoistureContent - 1)
ajuste0
```

```
##
## Call:
## lm(formula = GrowthRate ~ MoistureContent - 1)
##
## Coefficients:
## MoistureContent
##           26.55
```

```
plot( GrowthRate ~ MoistureContent, pch=16, col='blue')
abline(ajuste0,col='limegreen',lwd=2)
```



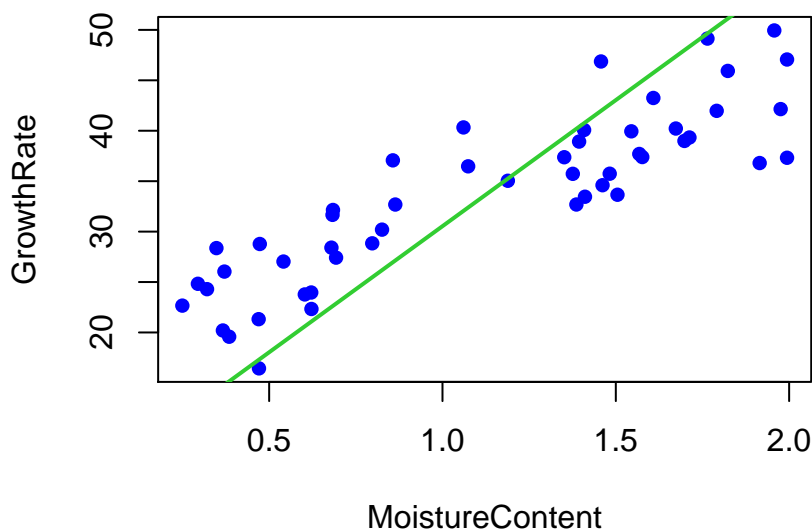
Si se deseara por el contrario ajustar únicamente la ordenada en el origen fijando la pendiente a un determinado valor fijo C , es decir, $y = C \cdot x + b$, entonces la sintaxis sería $y \sim C*x - 1$, que se lee “ajustar $y - C * x$ para hallar la ordenada en el origen” :

```
C = 25;
ajuste1 = lm(GrowthRate - C*MoistureContent ~ 1) # y - C*x = constante
ajuste1
```



```
##
## Call:
## lm(formula = GrowthRate ~ C * MoistureContent ~ 1)
##
## Coefficients:
## (Intercept)
##      5.538

plot( GrowthRate ~ MoistureContent, pch=16, col='blue')
abline(a = ajuste1$coefficients[[1]],b = C,col='limegreen',lwd=2)
```

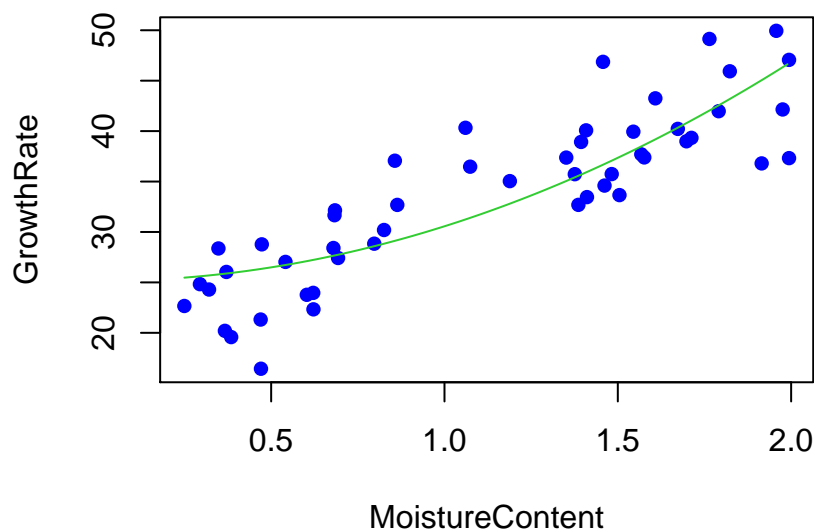


Con `lm` también se puede ajustar $y = a * f(x) + b$, donde $f(x)$ es una función de x . Pero para que la expresión de f no se confunda con los signos de la fórmula, hay que “aislarla”. Si se escribiera x^2 en la fórmula, el significado es otro (relacionado con las interacciones entre las componentes que pudiera tener x). Por tanto, habría que escribir $I(x^2)$, donde I es el operador *insulate*. Ajustemos $y = a_1 + a_2 \cdot x^2$:

```
ajuste2 = lm(GrowthRate ~ 1 + I(MoistureContent^2))
ajuste2
```

```
##
## Call:
## lm(formula = GrowthRate ~ 1 + I(MoistureContent^2))
##
## Coefficients:
## (Intercept)  I(MoistureContent^2)
##      25.134          5.431

plot( GrowthRate ~ MoistureContent, pch=16, col='blue')
cc = ajuste2$coefficients
x = seq(from=min(MoistureContent),to=max(MoistureContent),by=0.01)
y = cc[[1]] + cc[[2]] * (x^2)
lines(x,y,col="limegreen")
```

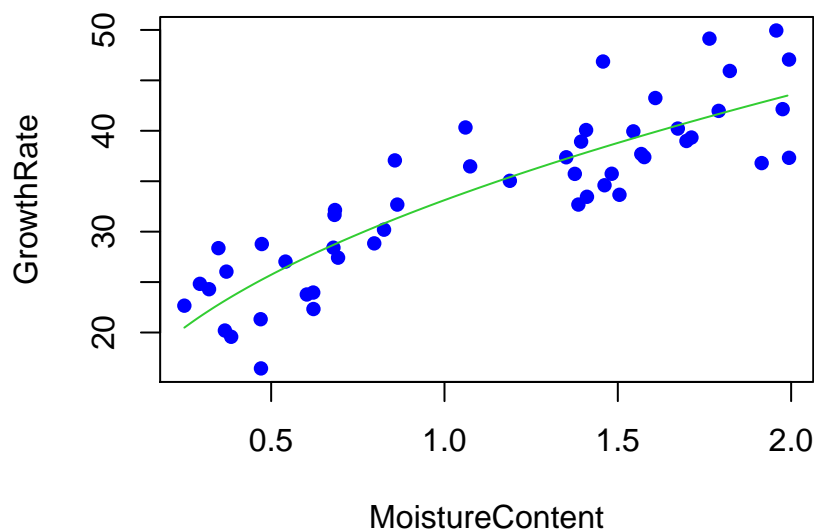


En vez de x^2 , se puede poner cualquier fórmula; por ejemplo, $y = a\sqrt{x} + b$:

```
ajuste3 = lm(GrowthRate ~ I(sqrt(MoistureContent)) + 1)
ajuste3
```

```
##
## Call:
## lm(formula = GrowthRate ~ I(sqrt(MoistureContent)) + 1)
##
## Coefficients:
##           (Intercept)  I(sqrt(MoistureContent))
##                7.868                25.255
```

```
plot( GrowthRate ~ MoistureContent, pch=16, col='blue')
coe = ajuste3$coefficients
x = seq(from=min(MoistureContent),to=max(MoistureContent),by=0.01)
y = coe[[2]] * sqrt(x) + coe[[1]]
lines(x,y,col="limegreen")
```



Las *interacciones* se señalan en las fórmulas con el signo de multiplicar: *. Así, $y \sim x * z$

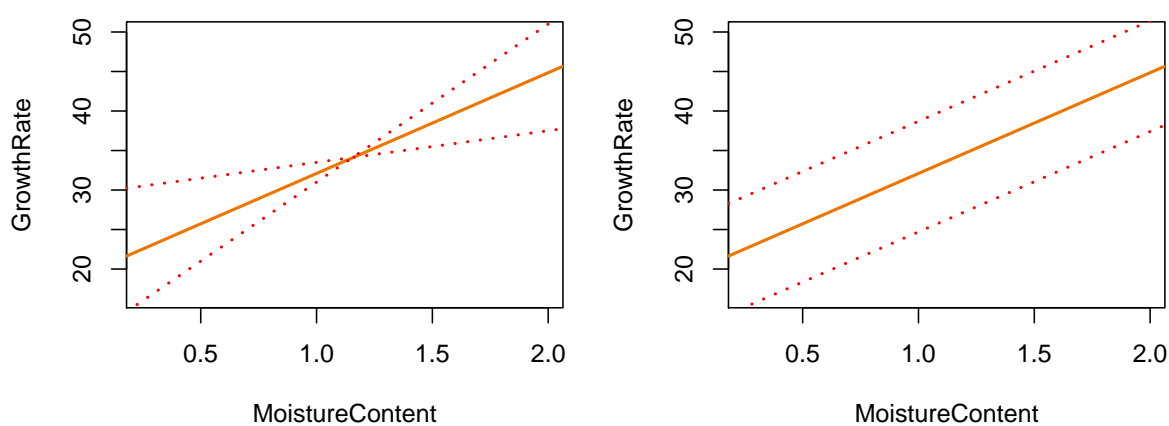
ajustaría $y = a_1 \cdot x + a_2 \cdot z + a_3 \cdot x \cdot z + b$ (nótese el término no lineal $a_3 \cdot x \cdot z$).

Nótese, de pasada, que con un poco de habilidad se puede ajustar cualquier cosa. En palabras de E. Fermi, “*dadme cinco parámetros y ajusto un elefante*”. Dicho de otra manera, un ajuste no explica nada; el ajuste hay que explicarlo previamente.

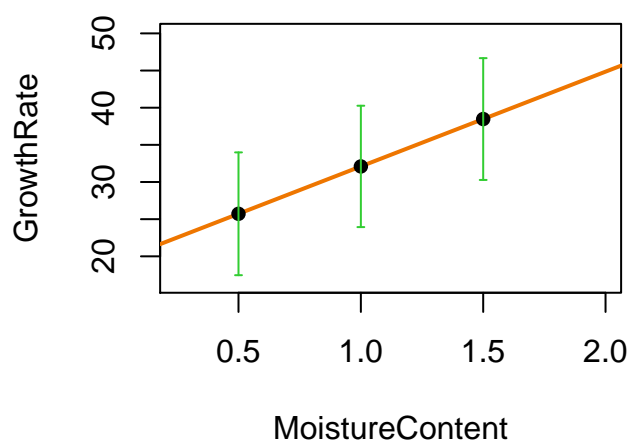
Bandas de confianza

La bondad del ajuste puede visualizarse mediante los intervalos de confianza, que pueden darse para los coeficientes y para las predicciones.

Las bandas de confianza para los coeficientes indican el error admisible en la pendiente y en la ordenada en el origen:



mientras que las bandas de confianza para las predicciones indican el intervalo de confianza para una nueva predicción



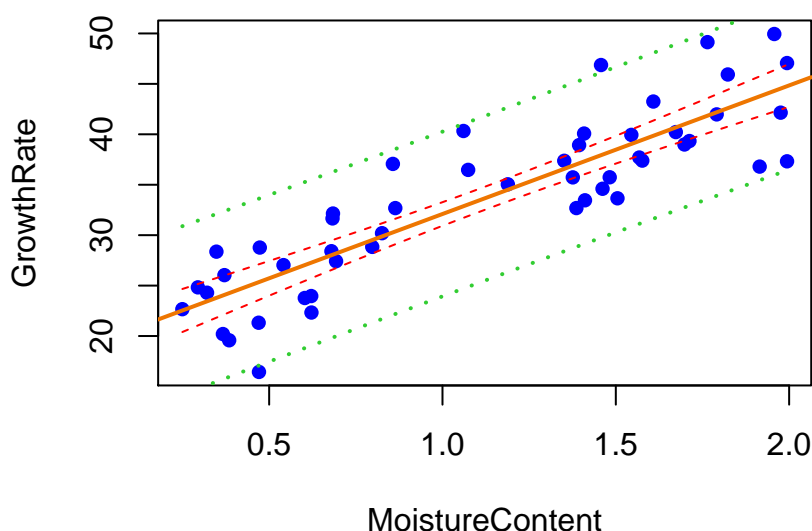
Pues bien, las bandas de confianza para los coeficientes son la envolvente de las líneas punteadas y las bandas de confianza para las predicciones son las envolventes de las barras de error. Para trazar esas líneas se necesitan unas series de puntos $\{x,y\}$, que se obtienen con el comando `predict` (en el primer caso con la opción `int = "c"` y en el segundo con `int = "p"`). La sintaxis es compleja y no se explica detenidamente.

```
plot(GrowthRate ~ MoistureContent, pch=16, col='blue')
abline(mi_ajuste,col='darkorange2', lwd=2)
```

```

# definición de una nueva serie de abscisas:
x = seq(from=min(MoistureContent),to=max(MoistureContent),by=0.01)
# intervalos de confianza para los coeficientes: nótese int="c"
y_conf_band = predict(mi_ajuste,
  newdata = data.frame(MoistureContent = x), int="c")
# las nuevas abscisas se pasan así: newdata = data.frame(MoistureContent = x)
lines(x,y_conf_band[,2],col="red",lty=2) # la de arriba
lines(x,y_conf_band[,3],col="red",lty=2) # la de abajo
# intervalos de confianza para la predicción: nótese int="p"
y_pred_int = predict(mi_ajuste,
  newdata = data.frame(MoistureContent = x), int="p")
lines(x,y_pred_int[,2],col="limegreen",lty=3,lwd=2)
lines(x,y_pred_int[,3],col="limegreen",lty=3,lwd=2)

```

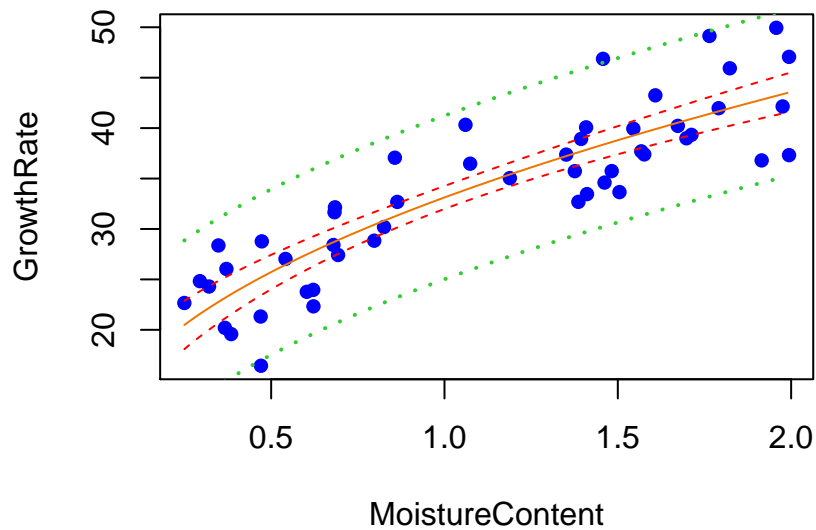


Las bandas de confianza se pueden dibujar para cualquier ajuste obtenido con `lm`:

```

plot(GrowthRate ~ MoistureContent, pch=16, col='blue')
ajuste3 = lm(GrowthRate ~ I(sqrt(MoistureContent)) + 1)
# ajuste3 es: y = a * sqrt(x) + b
xx = seq(from=min(MoistureContent),to=max(MoistureContent),by=0.01)
yy = coe[[2]] * sqrt(xx) + coe[[1]]
lines(xx,yy,col="darkorange2")
yy_conf_band = predict(ajuste3,
  newdata = data.frame(MoistureContent=xx), int="c")
lines(xx,yy_conf_band[,2],col="red",lty=2) # la de arriba
lines(xx,yy_conf_band[,3],col="red",lty=2) # la de abajo
yy_pred_int = predict(ajuste3,
  newdata = data.frame(MoistureContent=xx), int="p")
lines(xx,yy_pred_int[,2],col="limegreen",lty=3,lwd=2)

```



Correlación

El coeficiente de correlación de Pearson también se puede obtener así:

```
cor(GrowthRate, MoistureContent)
```

```
## [1] 0.8745262
```

Se hace notar que este número al cuadrado se proporciona en el resultado del modelo lineal como Multiple R-squared (*no* el llamado Adjusted R-squared).

Capítulo 10

Anova

Anova de una vía.

Vamos a usar unos datos incluidos en R. Se trata de un estudio sobre 6 insecticidas. Se dividió un área geográfica homogénea en 72 zonas del mismo tamaño. Se seleccionaron 12 zonas al azar para el insecticida A, de las restantes 12 para el insecticida B, de las restantes 12 para el insecticida C, etc., hasta tener una asignación aleatoria de 12 zonas para cada insecticida. Los datos están guardados en `InsectSprays`.

```
data(InsectSprays)
str(InsectSprays)
```

```
## 'data.frame':   72 obs. of  2 variables:
## $ count: num  10 7 20 14 14 12 10 23 17 20 ...
## $ spray: Factor w/ 6 levels "A","B","C","D",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(InsectSprays)
```

```
##   count spray
## 1    10    A
## 2     7    A
## 3    20    A
## 4    14    A
## 5    14    A
## 6    12    A
```

Vamos a realizar algunos estadísticos descriptivos por insecticida utilizando `dplyr`.

```
library(dplyr)
```

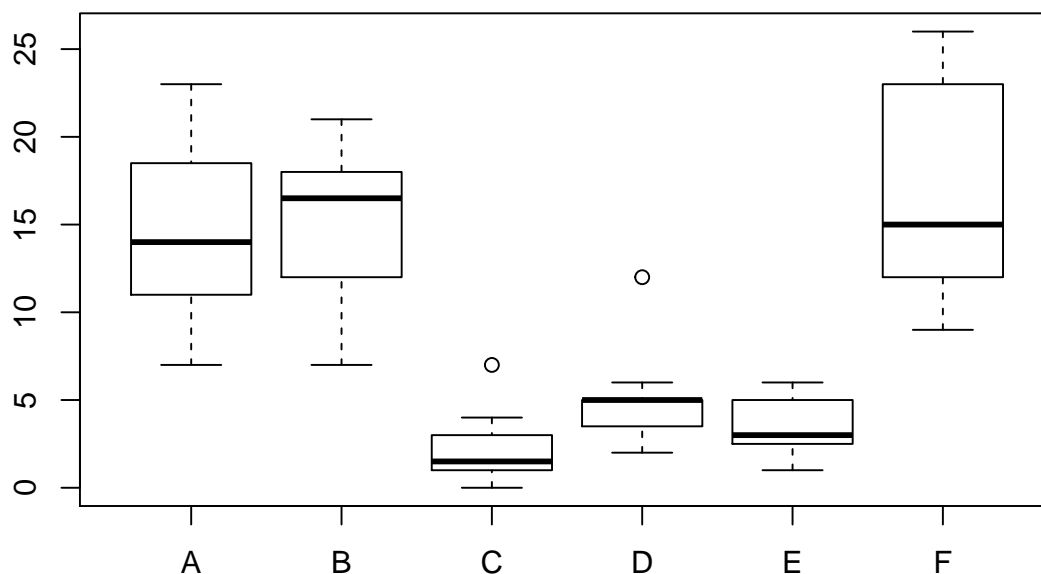
```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##   filter, lag
##
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
InsectSprays %>%
  group_by(spray) %>%
  summarise(media=mean(count), desvest=sd(count), n=length(count))
```

```
## # A tibble: 6 x 4
##   spray media desvest   n
##   <fct> <dbl> <dbl> <int>
## 1 A      14.5    4.72    12
## 2 B      15.3    4.27    12
## 3 C       2.08    1.98    12
## 4 D       4.92    2.50    12
## 5 E       3.5     1.73    12
## 6 F      16.7    6.21    12
```

Ahora dibujamos los correspondientes diagramas de caja. Vemos como boxplot nos permite una alternativa a `with(...)` usando `boxplot(...,data=InsectSprays)`. Hay muchas funciones en R que permiten especificar el *dataframe* como opción con `data=nombredataframe`. El resultado es el mismo que el que obtendríamos usando `with(InsectSprays,boxplot(count~spray))`

```
boxplot(count ~ spray, data=InsectSprays)
```



En este gráfico se aprecia que no se cumplen los requisitos para una ANOVA. Algunas distribuciones no son simétricas, hay bastantes *outlayers*, y además las varianzas parecen bastante distintas.

Aún así realizaremos una ANOVA para ver cuál sería el procedimiento. Hay varias maneras de realizar una ANOVA con R.

```
salida.aov = aov(count ~ spray, data=InsectSprays)
summary(salida.aov)
```



```
##           Df Sum Sq Mean Sq F value Pr(>F)
## spray      5  2669   533.8   34.7 <2e-16 ***
## Residuals 66  1015    15.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Otra manera es utilizar la orden `oneway.test()` que hace la corrección de Welch para varianzas distintas (pero requiere normalidad)

```
oneway.test(count~spray, data=InsectSprays)
```

```
##
## One-way analysis of means (not assuming equal variances)
##
## data:  count and spray
## F = 36.065, num df = 5.000, denom df = 30.043, p-value = 7.999e-12
```

El resultado difiere del anterior por la corrección de Welch. Si ponemos la opción `var.equal=TRUE` obtendremos los mismos resultados que con `aov`

```
oneway.test(count~spray, data=InsectSprays, var.equal=TRUE)
```

```
##
## One-way analysis of means
##
## data:  count and spray
## F = 34.702, num df = 5, denom df = 66, p-value < 2.2e-16
```

Una tercera manera es utilizar un modelo lineal con `lm` y después utilizar la orden `anova()` sobre la salida del modelo. En realidad esto es lo que hace la orden `aov` internamente. En una *anova* de una vía con una variable *tratamiento* y una variable *respuesta* haremos `lm(respuesta~tratamiento)`. Esto equivale a una regresión donde la variable respuesta es cuantitativa y la variable tratamiento es cuantitativa.

```
modelo=lm(count~spray, data=InsectSprays)
anova(modelo)
```

```
## Analysis of Variance Table
##
## Response: count
##           Df Sum Sq Mean Sq F value    Pr(>F)
## spray      5 2668.8   533.77  34.702 < 2.2e-16 ***
## Residuals 66 1015.2    15.38
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Test Post-Hoc

Si queremos ahora ver qué grupos son significativamente distintos, podemos hacer un test Post Hoc. Se pueden hacer estos tests con el comando `pairwise.t.test(variable, grupo)`. Este método nos permite hacer distintas correcciones con `p.adjust=` Por ejemplo para realizar la corrección de Bonferroni se haría así:

```
with(InsectSprays,
     pairwise.t.test(count,spray,p.adj="bonferroni"))
```

```
##
## Pairwise comparisons using t tests with pooled SD
##
## data:  count and spray
##
##   A         B         C         D         E
## B 1         -         -         -         -
## C 1.1e-09 1.3e-10 -         -         -
## D 1.5e-06 1.8e-07 1         -         -
## E 4.1e-08 4.9e-09 1         1         -
## F 1         1         4.2e-12 6.1e-09 1.6e-10
##
## P value adjustment method: bonferroni
```

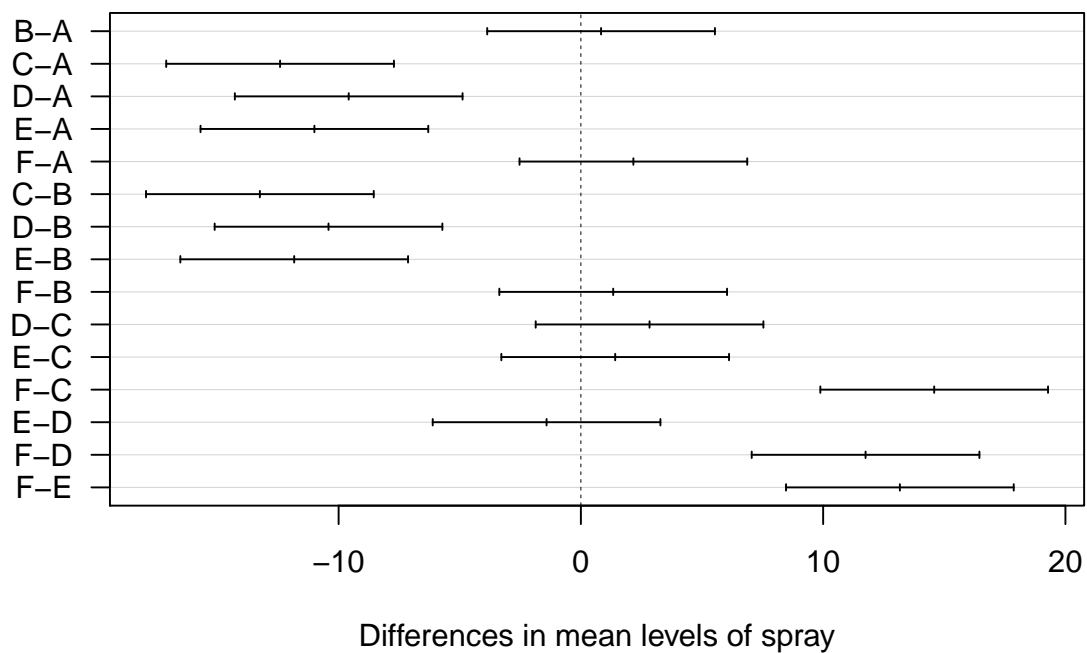
En R el test HSD de Tuckey (*Honest Significant Difference*) nos da intervalos de confianza corregidos para las diferencias de medias correspondientes a cada par de grupos.

```
TukeyHSD(salida.aov)
```

```
## Tukey multiple comparisons of means
## 95% family-wise confidence level
##
## Fit: aov(formula = count ~ spray, data = InsectSprays)
##
## $spray
##           diff           lwr           upr           p adj
## B-A  0.8333333 -3.866075  5.532742 0.9951810
## C-A -12.4166667 -17.116075 -7.717258 0.0000000
## D-A  -9.5833333 -14.282742 -4.883925 0.0000014
## E-A -11.0000000 -15.699409 -6.300591 0.0000000
## F-A   2.1666667 -2.532742  6.866075 0.7542147
## C-B -13.2500000 -17.949409 -8.550591 0.0000000
## D-B -10.4166667 -15.116075 -5.717258 0.0000002
## E-B -11.8333333 -16.532742 -7.133925 0.0000000
## F-B   1.3333333 -3.366075  6.032742 0.9603075
## D-C   2.8333333 -1.866075  7.532742 0.4920707
## E-C   1.4166667 -3.282742  6.116075 0.9488669
## F-C  14.5833333   9.883925 19.282742 0.0000000
## E-D  -1.4166667 -6.116075  3.282742 0.9488669
## F-D  11.7500000   7.050591 16.449409 0.0000000
## F-E  13.1666667   8.467258 17.866075 0.0000000
```

```
plot(TukeyHSD(salida.aov),las=1)
```

95% family-wise confidence level



las indica la orientación de las etiquetas con respecto a los ejes; puede tomar los valores: 0=paralelas, 1=todas horizontales, 2=todas perpendiculares a los ejes, 3=todas verticales.

Podemos también obtener información examinando el modelo lineal

```
summary(modelo)
```

```
##
## Call:
## lm(formula = count ~ spray, data = InsectSprays)
##
## Residuals:
##   Min     1Q  Median     3Q    Max
## -8.333 -1.958 -0.500  1.667  9.333
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  14.5000     1.1322  12.807 < 2e-16 ***
## sprayB       0.8333     1.6011   0.520  0.604
## sprayC     -12.4167     1.6011 -7.755 7.27e-11 ***
## sprayD      -9.5833     1.6011 -5.985 9.82e-08 ***
## sprayE     -11.0000     1.6011 -6.870 2.75e-09 ***
## sprayF       2.1667     1.6011  1.353  0.181
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.922 on 66 degrees of freedom
## Multiple R-squared:  0.7244, Adjusted R-squared:  0.7036
## F-statistic: 34.7 on 5 and 66 DF, p-value: < 2.2e-16
```

Otros métodos para comparaciones múltiples se pueden encontrar en el paquete `multcomp`.

Verificación de las condiciones

Una manera formal de probar si las varianzas son iguales es el test de Bartlett.

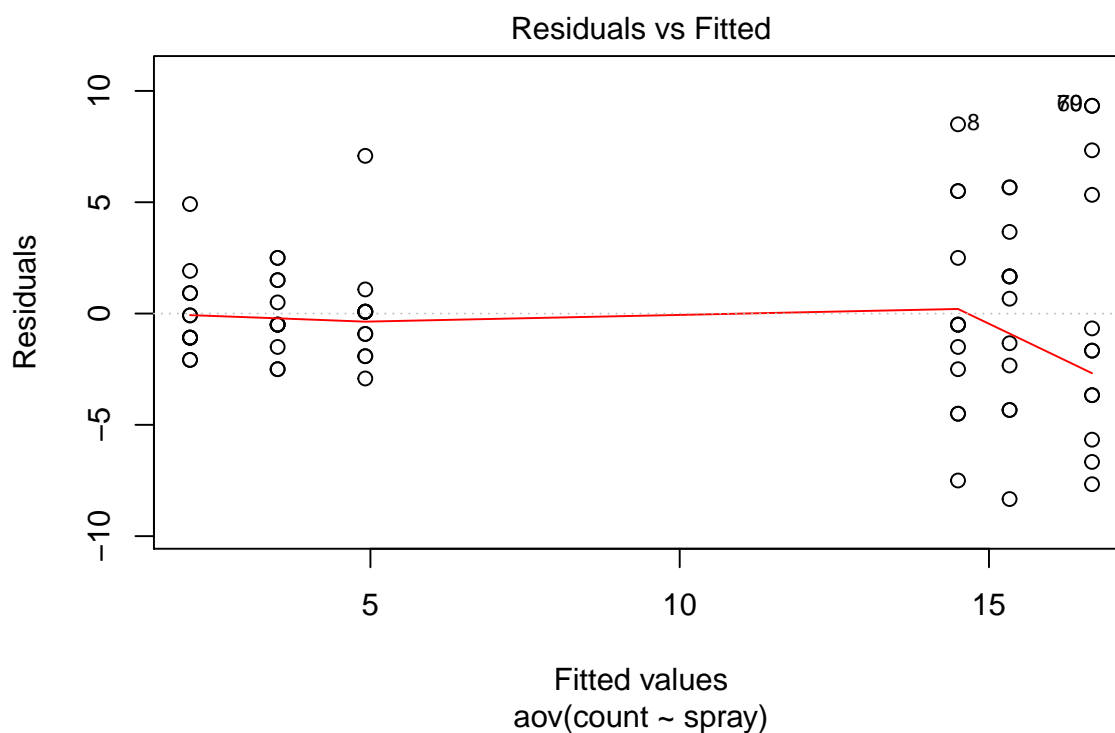
```
bartlett.test(count ~ spray, data=InsectSprays)
```

```
##
## Bartlett test of homogeneity of variances
##
## data: count by spray
## Bartlett's K-squared = 25.96, df = 5, p-value = 9.085e-05
```

No podemos asegurar que las varianzas son iguales.

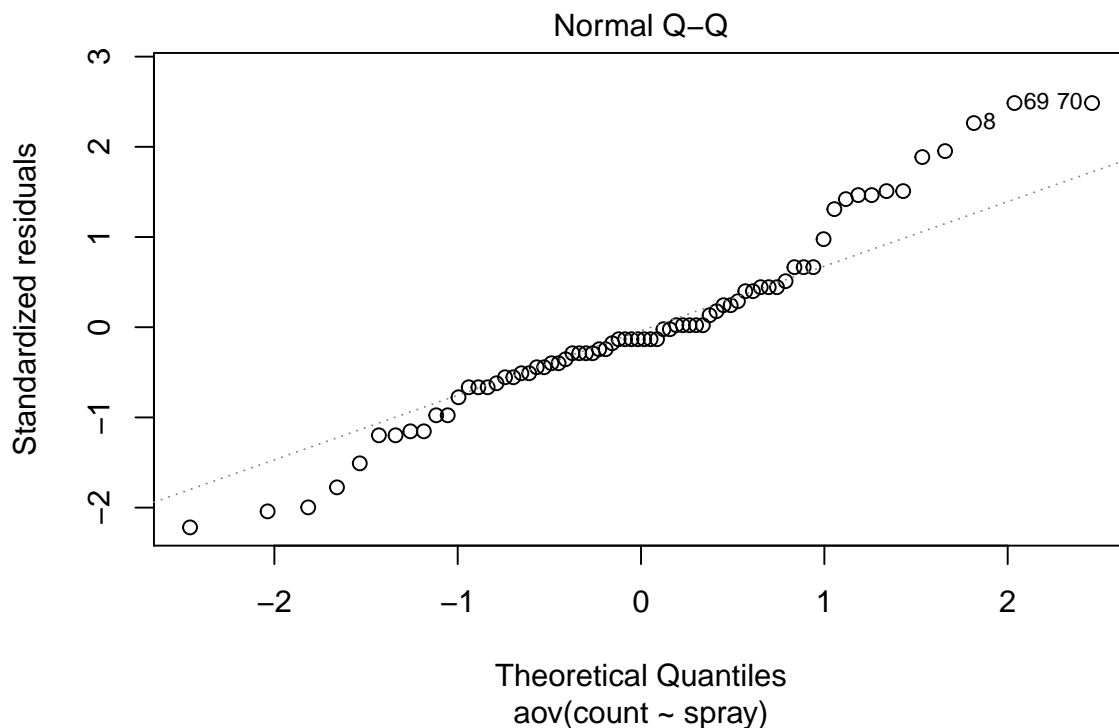
Podemos obtener una serie de diagnósticos de un test de anova con `plot(salida.aov)`

```
plot(salida.aov, 1)
```



Nos da un gráfico de la distribución de los residuos. Idealmente ha de mostrar la misma dispersión en cada valor. Aquí se aprecia un preocupante aumento en la dispersión de los residuos para valores grandes.

```
plot(salida.aov, 2)
```



Nos da un gráfico cuantil-cuantil que compara la distribución de los residuos con una normal. Vemos que en las colas se aleja de una normal.

Alternativa no paramétrica. Test de Kruskal-Wallis.

La alternativa no paramétrica es el test de Kruskal-Wallis. Este test necesita que todas las distribuciones sean similares y que haya homogeneidad de varianzas.

```
kruskal.test(count ~ spray, data=InsectSprays)
```

```
##
## Kruskal-Wallis rank sum test
##
## data: count by spray
## Kruskal-Wallis chi-squared = 54.691, df = 5, p-value = 1.511e-10
```

Transformaciones de los datos

Cuando nuestros datos no cumplen las condiciones para un test (normalidad, igualdad de varianzas), cabe aplicar transformaciones a nuestros datos $Y' = f(Y)$ para que sí las cumplan y realizar los tests sobre los datos transformados. Habrá que tener especial cuidado a la hora de interpretar los resultados, pues nos interesan resultados sobre los datos sin transformar.

Las transformaciones más habituales son:

- $Y' = \log(Y)$ Transformación válida para valores positivos o $Y' = \log(Y + 1)$ si los datos incluyen al 0. La media aritmética queda transformada en media geométrica. Algunos casos en los que esta transformación es útil comprenden cuando las medidas son cocientes

o productos de variables o cuando la distribución de los datos tiene una cola a la derecha o cuando el grupo que tiene la media mayor también tiene la desviación estándar mayor o cuando los datos comprenden varios ordenes de magnitud.

- $Y' = \text{logit}(Y)$ Usada fundamentalmente para proporciones.
- $Y' = \sqrt{Y + a}$ donde a puede tomar los valores 0, 1/2 o 1 según los datos. Usada cuando los datos son conteos.

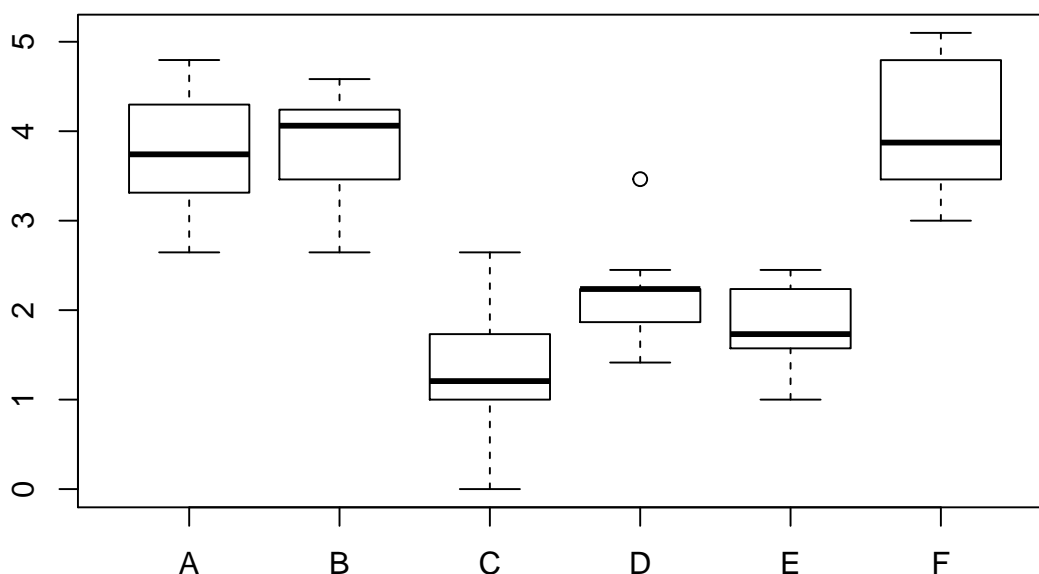
Hay muchas otras transformaciones posibles, pero hay que tener cuidado a la hora de probar distintas transformaciones porque podríamos incurrir en los mismos problemas que al hacer comparaciones múltiples (Incremento del error de tipo I)

En los datos de InsectSpray si hacemos:

```
bartlett.test(sqrt(count) ~ spray, data=InsectSprays)
```

```
##
## Bartlett test of homogeneity of variances
##
## data: sqrt(count) by spray
## Bartlett's K-squared = 3.7525, df = 5, p-value = 0.5856
```

```
with(InsectSprays,boxplot(sqrt(count) ~ spray))
```

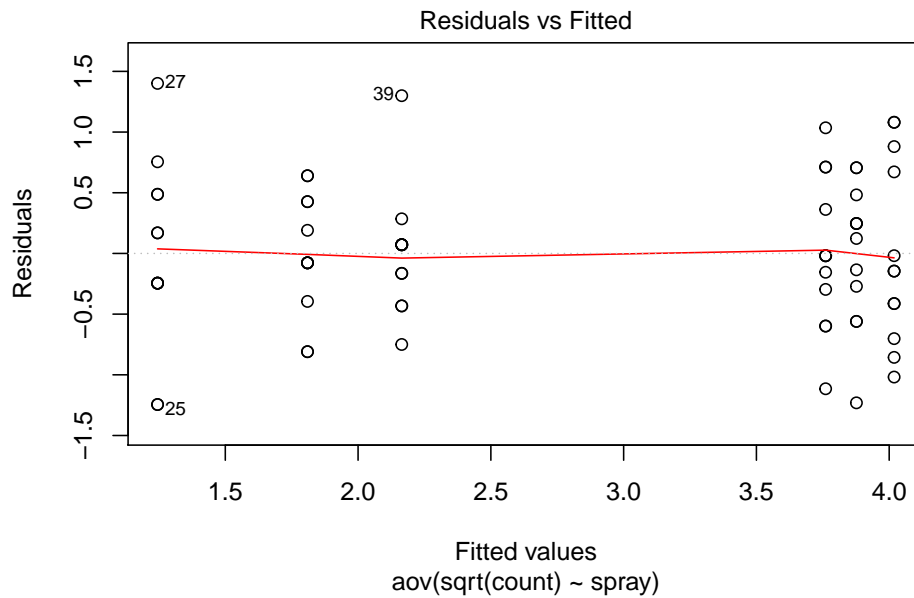


```
salida.aov2 <- aov(sqrt(count) ~ spray, data = InsectSprays)
summary(salida.aov2)
```

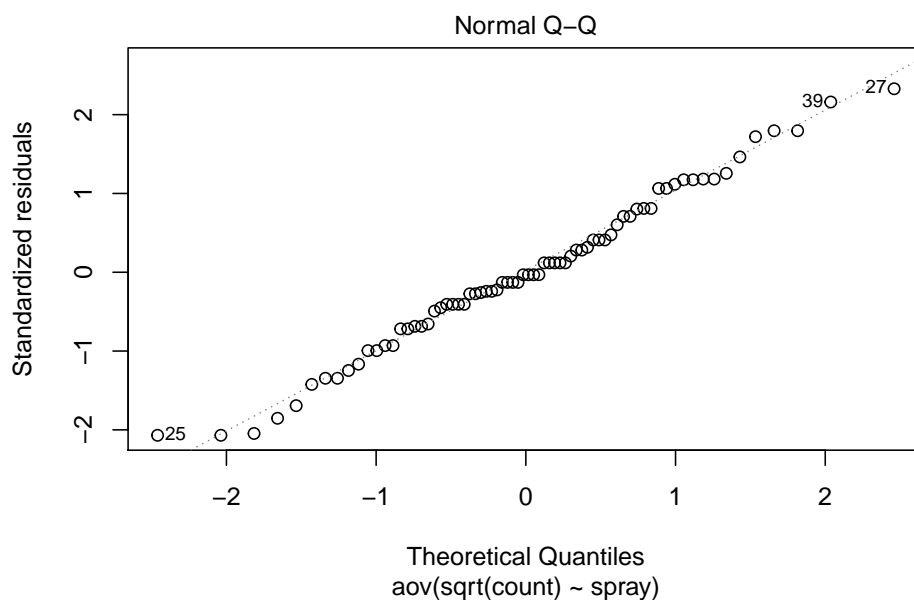
```
##           Df Sum Sq Mean Sq F value Pr(>F)
## spray      5  88.44  17.688    44.8 <2e-16 ***
## Residuals 66  26.06   0.395
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(salida.aov2,1)
```



```
plot(salida.aov2,2)
```



Anova de dos vías

En el fichero *quinn.csv* encontramos datos sobre los efectos que tienen la estación del año y la densidad de ejemplares adultos en la producción de huevos de la especie *Sinphonaria Diemenensis*

```
quinn=read.csv("./DATA/quinn.csv")
str(quinn)
```

```
## 'data.frame': 24 obs. of 3 variables:
## $ DENSITY: int 8 8 8 8 8 8 15 15 15 15 ...
## $ SEASON : Factor w/ 2 levels "spring","summer": 1 1 1 2 2 2 1 1 1 2 ...
## $ EGGS : num 2.88 2.62 1.75 2.12 1.5 ...
```

Este conjunto de datos es típico de un ANOVA con dos factores (y efectos fijos). Tenemos una variable (respuesta) cuantitativa, que es la producción de huevos (la tercera columna de los datos). Y queremos estudiar la relación de esa variable con dos variables (explicativas), que son la la densidad de adultos y la estación (primera y segunda columnas, respectivamente).

Se trata de dos variables cualitativas o factores. Esto es especialmente evidente en el caso de la variable SEASON, que tiene dos niveles (*spring* y *summer*). Vemos que la variable DENSITY no es un factor, así que la transformamos en factor.

```
quinn.aov = aov(EGGS ~ SEASON + DENSITY, data = quinn)
anova(quinn.aov)
```

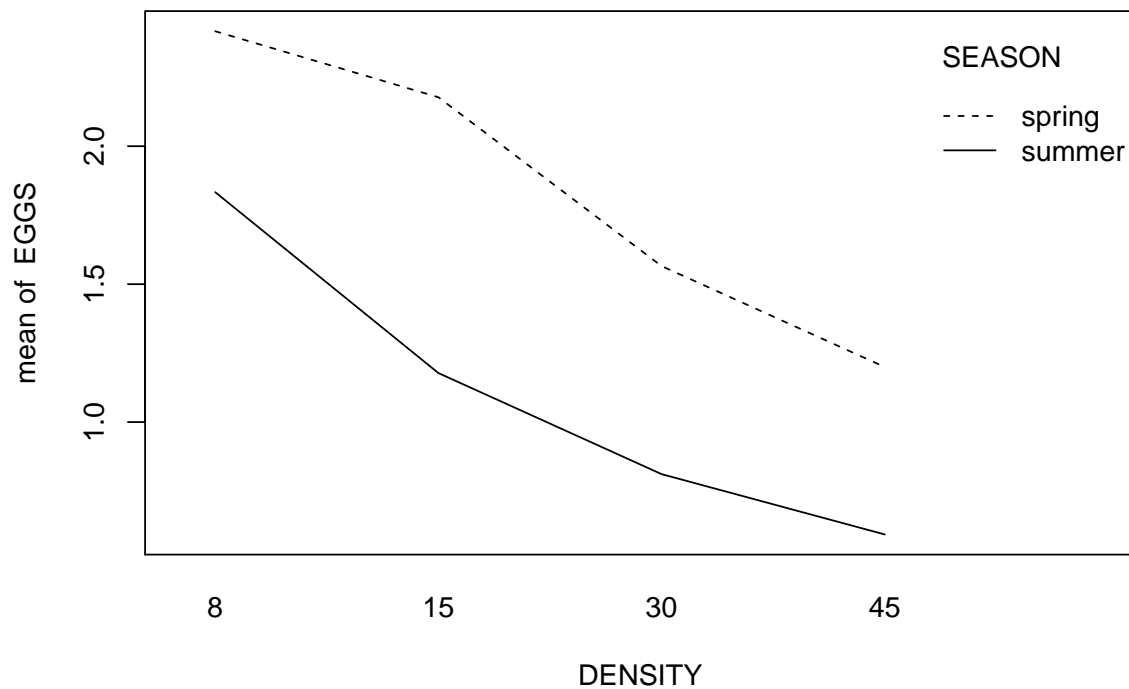
```
## Analysis of Variance Table
##
## Response: EGGS
##          Df Sum Sq Mean Sq F value    Pr(>F)
## SEASON     1  3.2502   3.2502  20.439 0.000187 ***
## DENSITY     1  5.0241   5.0241  31.595 1.406e-05 ***
## Residuals 21  3.3394   0.1590
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Podemos hacer un estudio factorial (viendo las posibles interacciones)

```
quinn.aov = aov(EGGS ~ SEASON * DENSITY, data = quinn)
anova(quinn.aov)
```

```
## Analysis of Variance Table
##
## Response: EGGS
##          Df Sum Sq Mean Sq F value    Pr(>F)
## SEASON     1  3.2502   3.2502  19.5350 0.0002637 ***
## DENSITY     1  5.0241   5.0241  30.1971 2.226e-05 ***
## SEASON:DENSITY 1  0.0118   0.0118   0.0711 0.7925333
## Residuals  20  3.3275   0.1664
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
with(quinn,interaction.plot(DENSITY, SEASON, EGGS))
```

El factor DENSITY se muestra en el eje horizontal, la respuesta EGGS en el eje vertical, y cada uno de los dos factores de SEASON se muestra como una línea que conecta los correspondientes valores.

En un caso ideal, la ausencia completa de interacción correspondería a dos líneas perfectamente paralelas en este gráfico (y la existencia de interacción es evidente cuando las líneas se cruzan).

En la práctica, el paralelismo en general está lejos de ser perfecto. No obstante, la gráfica de este ejemplo sí parece indicar que no existe interacción entre ambos factores. Y en cualquier caso, tenemos los resultados de la tabla ANOVA para corroborarlo.

Otros diseños.

Para un estudio más detallado de la Anova y otros diseños recomendamos el excelente libro online gratuito de Luka Meier

[Anova: A short intro using R](#)

Capítulo 11

Bibliografía y recursos

Libros de referencia

1. P. Dalgaard, “*Introductory Statistics with R*”, Springer (2008).
2. W. N. Venables, D. M. Smith and the R Core Team, “*Introduction to R*”, disponible en pdf en Internet.
3. G. Grolemund and H. Wickham, “*R for Data Science*”, disponible en Internet: <https://r4ds.had.co.nz/>

Tutorials

Una página web realizada por W. B. King, de la Coastal Carolina University, con muchos ejemplos prácticos: <http://ww2.coastal.edu/kingw/statistics/R-tutorials/>

Recursos en Internet

- **Quick R** <https://www.statmethods.net/> (resumen, tutorial, lo básico). Muy bueno para empezar.
- **A compendium of clean graphs in R** <http://shinyapps.org/apps/RGraphCompendium/index.php> :una gran cantidad de gráficos bien hechos, que adjunta el código con el que se han elaborado listo para copiar y pegar.
- Un buscador internet restringido a temas de R: <https://rseek.org/>
- Cuestiones y respuestas de usuarios y programadores, para varios lenguajes: <https://stackoverflow.com/questions>