Visualizacion de datos con R

FSC

Contents

La importancia de la visualización de datos

Cuando trabajamos con grandes cantidades es esencial utilizar diferentes tipo de visualizaciones para explorar los datos. Sólo así podremos darnos cuenta de diferentes sesgos que deben ser corregidos o incluso de errores.

Además la visualización de los datos es esencial para extraer patrones y conclusiones cuando vemos muchos datos. Esto es algo que la mente humana en la población general no puede hacer a partir de números. Por ejemplo, este artículo del Wall Street Journal muestra como ha sido la evolución de ciertas enfermedades infecciosas desde su aparición hasta la aplicación de las vacunas: *http://graphics.wsj.com/infectious-diseases-and-vaccines/?mc_cid=711ddeb86e*

Vamos a comenzar por ver partes de la charla TED de Hans Rosling de la fundación gapminder de la que os hablé en la clase anterior: *https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen*

R Basic plots: paquete *graphics()*

Una de las grandes fortalezas de R consiste en la facilidad con la que podemos representar datos de diferentes formas y con formato de alta calidad. Visualizar los datos es esencial para entenderlos y proponer modelos.

Para seguir esta sesión necesitarás tener instaladas las siguientes librerias:

```
library(dslabs)
library(ggplot2)
library(dplyr)
library(tidyverse)
library(readr)
```

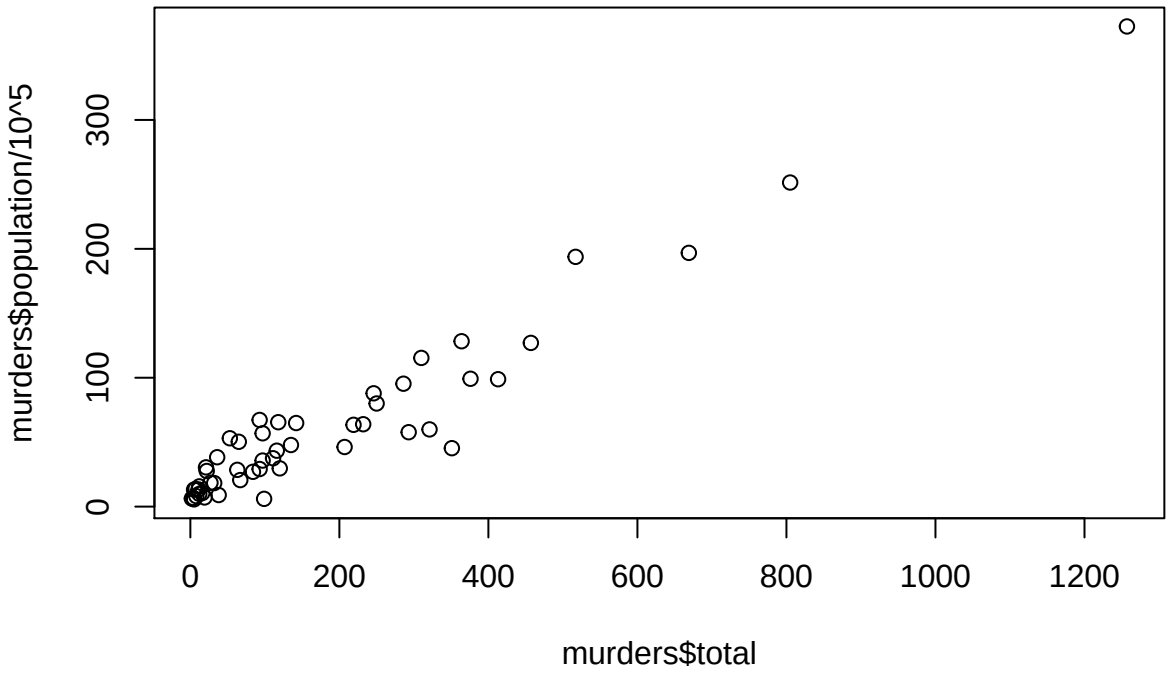
Scatterplots (nubes de puntos)

El plot más básico en el que podemos pensar es una nube de puntos. Solemos utilizarlo con frecuencia si queremos entender la relación que existe entre dos variables. Por ejemplo, si queremos ver la relación entre el número de asesinatos en un estado y su población (por 100.000 habitantes) utilizamos la función *plot()*

```
data(murders)
?plot
```

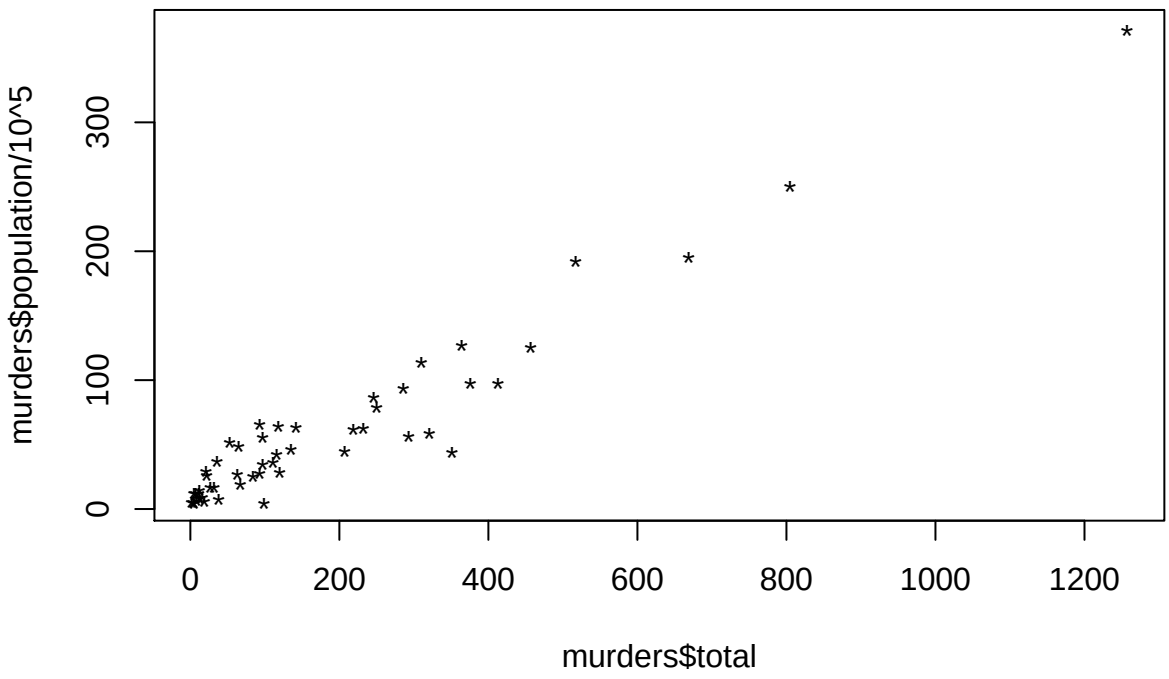
```
## starting httpd help server ... done
```

```
plot(murders$total,murders$population/10^5)
```



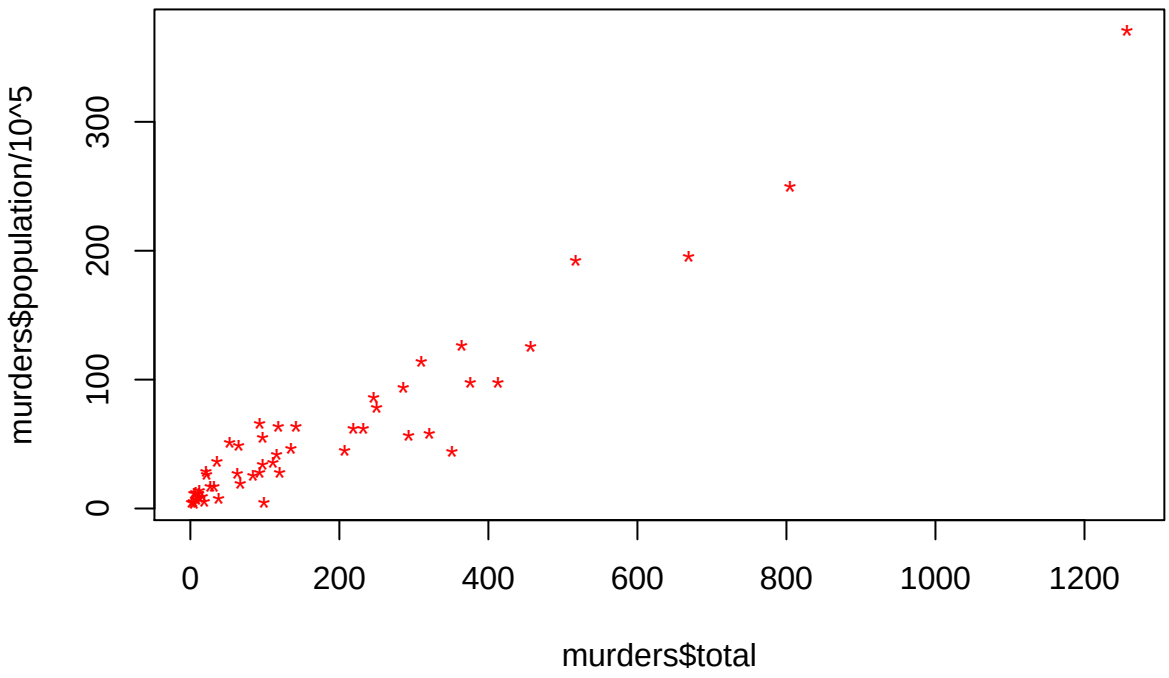
Para esta función podemos ajustar el tipo de punto con el parámetro *pch*:

```
plot(murders$total,murders$population/10^5,pch="*")
```



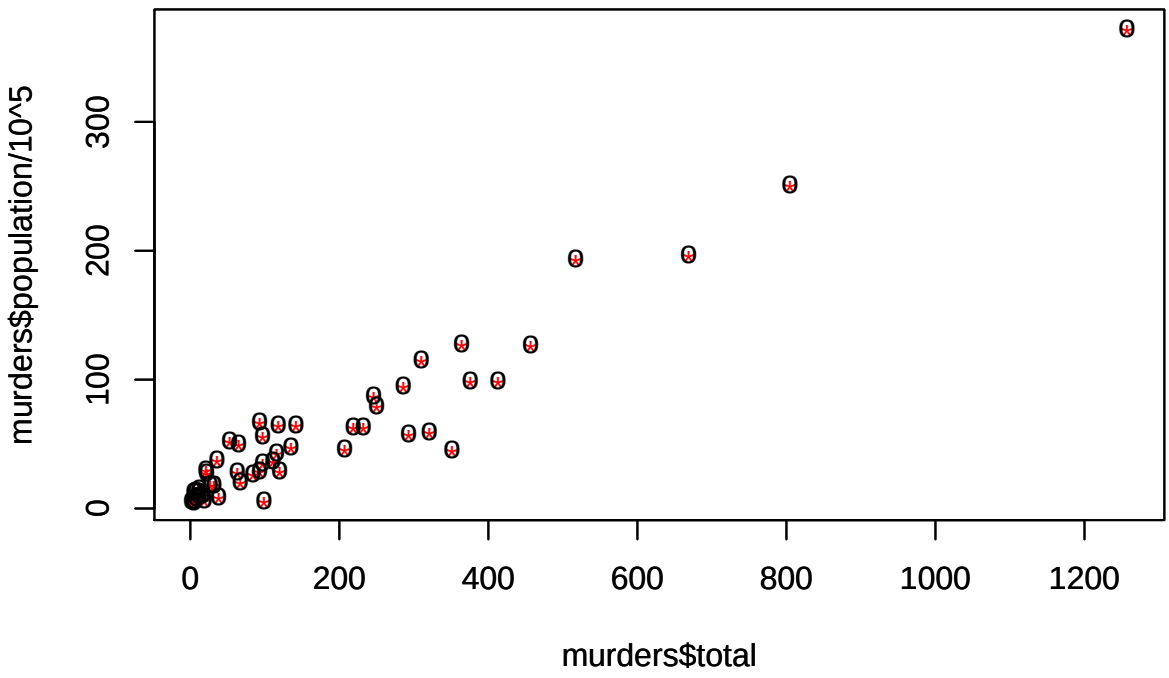
O el color:

```
plot(murders$total,murders$population/10^5,pch="*",col="red")
```



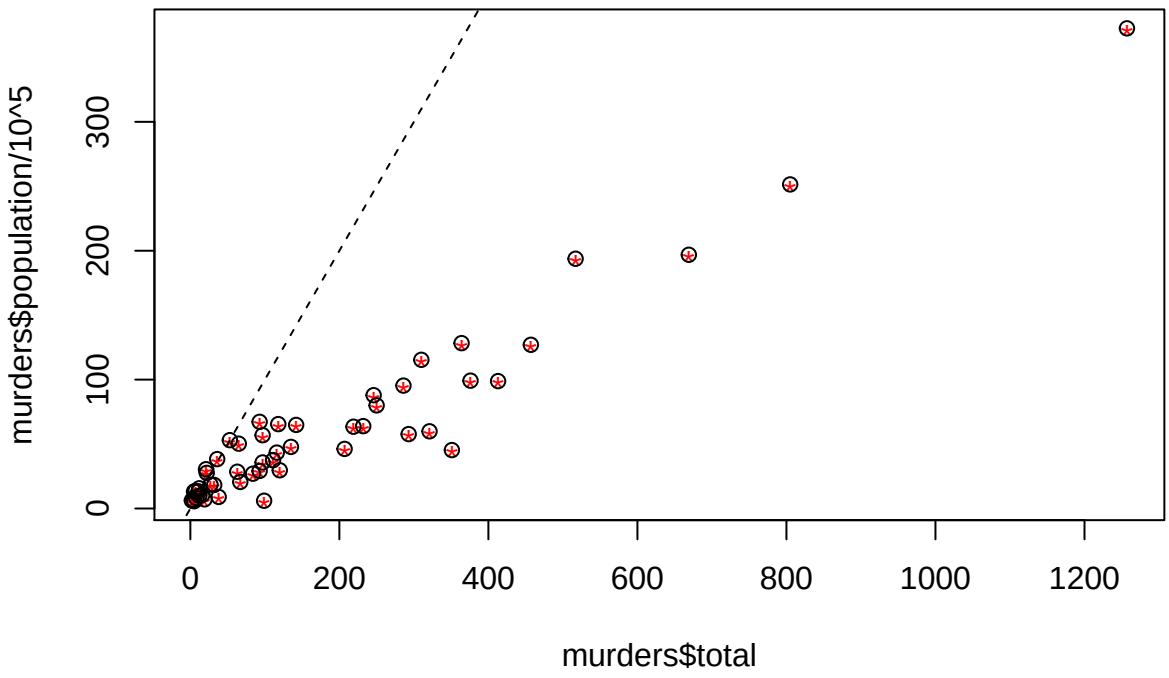
Cada vez que llamamos a la función *plot()* se abre una nueva ventana. Si queremos pintar en un gráfico ya existente tenemos dos opciones:

```
plot(murders$total,murders$population/10^5,pch="*",col="red")
par(new=T)
plot(murders$total,murders$population/10^5,pch="o")
```



o bien usar *points()* o *lines()* que pintan puntos o unen puntos por medio de líneas si los puntos ya están dibujados. La función abline pinta una linea de pendiente b y ordenada en el origen a.

```
plot(murders$total,murders$population/10^5,pch="*",col="red")
points(murders$total,murders$population/10^5,lty=2)
abline(a=0,b=1,lty=2)
```



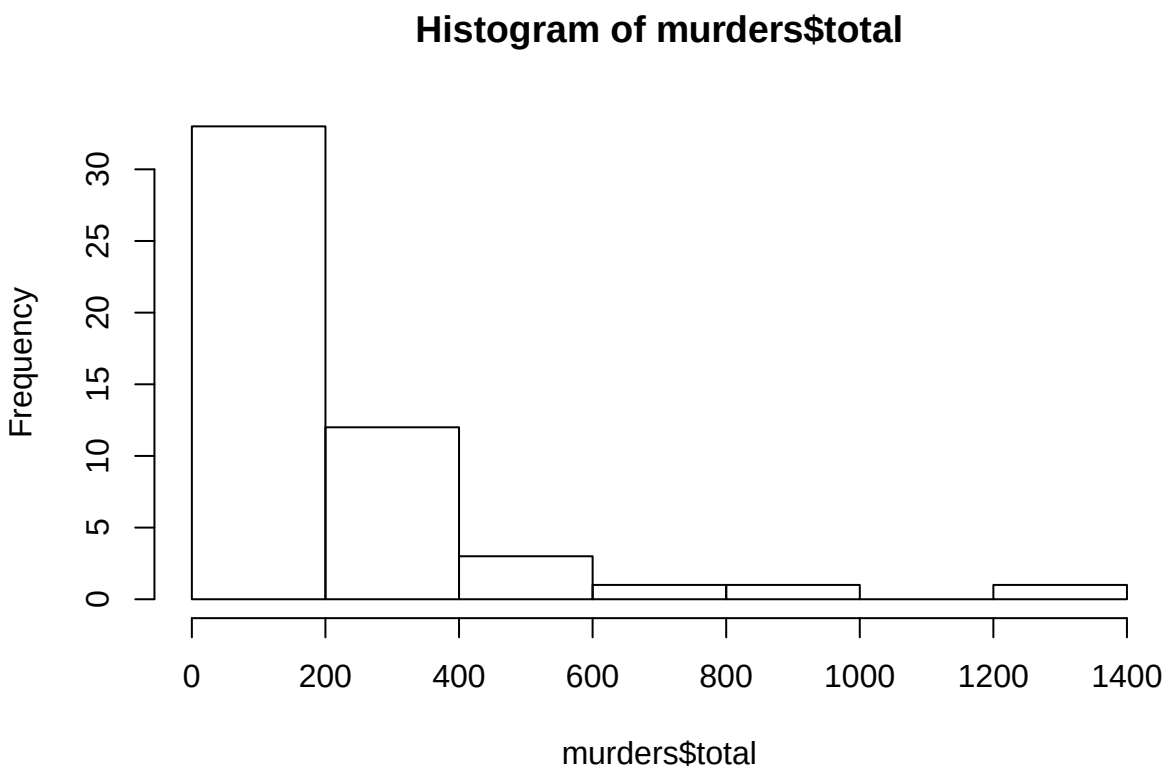
El parámetro lty controla el tipo de línea (sólida, discontinua, etc)

Estos simples gráficos ya nos muestran que hay una relación de tipo lineal entre el numero de asesinatos y la población total de un estado

Histograms

Un histograma nos muestra la distribución de los elementos de una muestra. Es decir, nos dice cuántos elementos de cada tipo hay.

```
hist(murders$total)
```



Es decir, hay unos 32-33 estados con menos de 200 asesinatos; hay unos 10 estados con entre 200 y 400 asesinatos y el resto (~10 estados) sufrieron más de 400 asesinatos. EN particular parece que hay 1 estado con entre 1200 y 1400 asesinatos.

Vamos a comprobarlo con una tabla. Primero binarizamos nuestros resultados en bins de 200:

```
murders$total.bin=murders$total
murders$total.bin[which(murders$total<=200)]=200
murders$total.bin[which(murders$total>200 & murders$total<=400)]=400
murders$total.bin[which(murders$total>400 & murders$total<=600)]=600
murders$total.bin[which(murders$total>600 & murders$total<=800)]=800
murders$total.bin[which(murders$total>800 & murders$total<=1000)]=1000
murders$total.bin[which(murders$total>1000 & murders$total<=1200)]=1200
murders$total.bin[which(murders$total>1200 & murders$total<=1400)]=1400
table(murders$total.bin)
```

```
##
##  200  400  600  800 1000 1400
##   33   12    3    1    1    1
```

Arpovechamos para introducir la sintaxis de un loop en R. Podríamos haber utilizado un bucle para programar lo anterior:

```
murders$total.bin=murders$total
for (i in 1:7){
  murders$total.bin[which(murders$total>200*(i-1) & murders$total<=200*i)]=200*i
}
table(murders$total.bin)
```

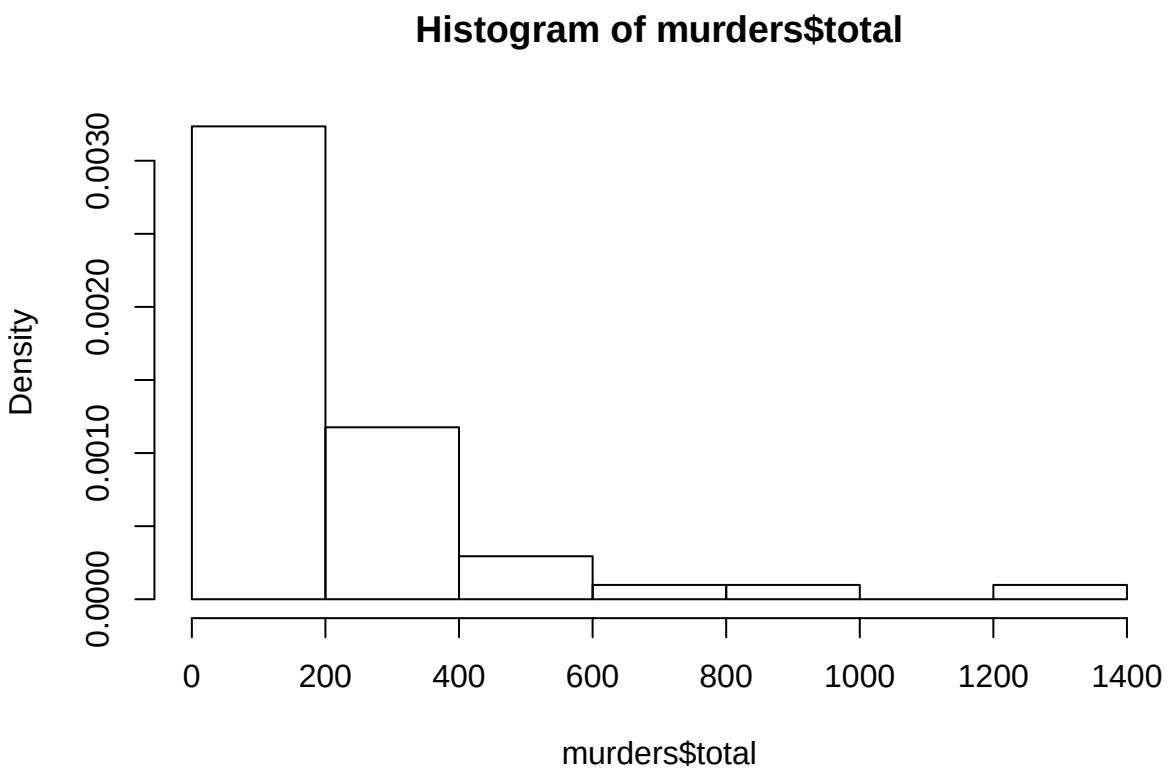
```
##
##  200  400  600  800 1000 1400
##   33   12    3    1    1    1
```

el número de bins puede cambiarse fácilmente con el comando *breaks()*

```
hist(murders$total,breaks=10)
```

Y también podemos elegir ver la frecuencia (número de elementos en cada bin) o la probabilidad de tener un elemento en cada bin (# elementos en bin/total numero de elementos)

```
hist(murders$total,freq = F)
```

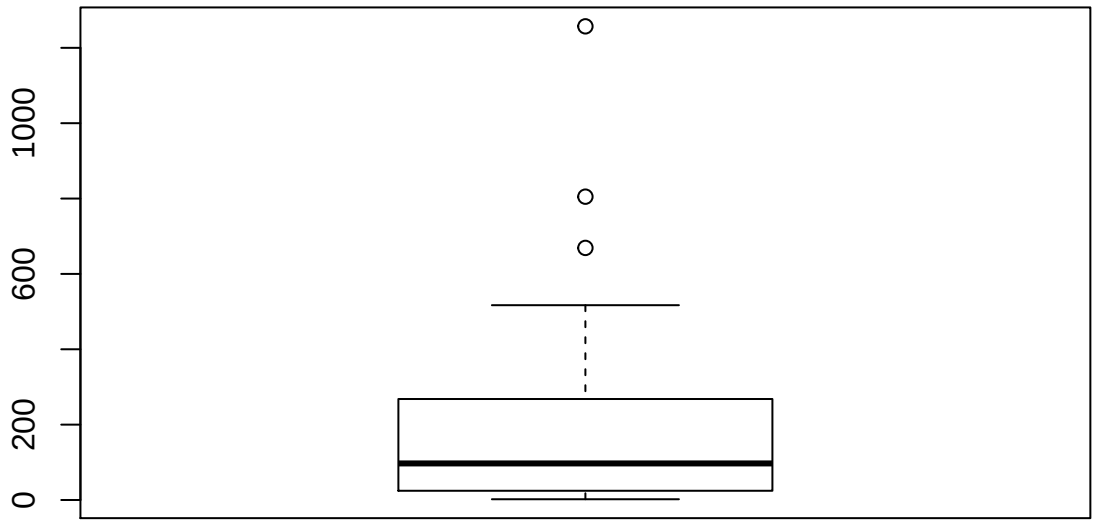


Con este tipo de plot obtenemos una idea de cómo es la distribución de los datos: no simétrica, con el valor mas probable entre 0 y 200 y con algunos valores muy distintos dl resto. Estas características serán las que miraremos más adelante cuando queramos caracterizar distribuciones de datos.

Boxplot

Un boxplot también nos da pistas acerca de la distribución de un conjunto de datos

```
boxplot(murders$total)
```



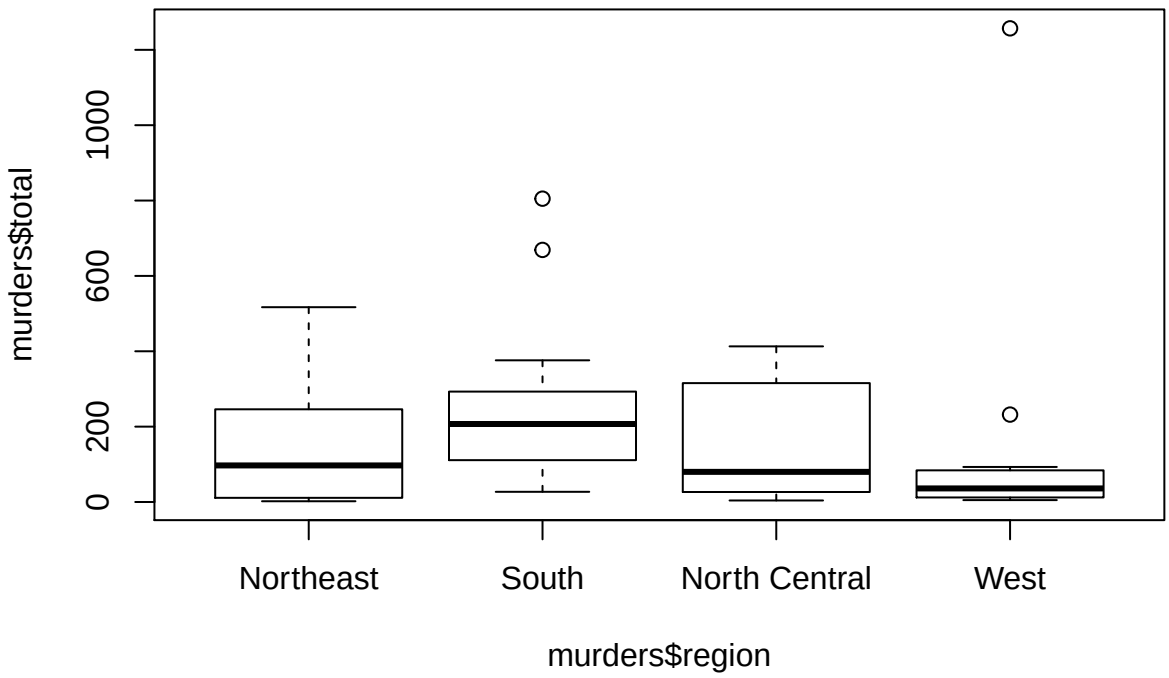
Vemos que la caja tiene una linea central (la mediana) que en este caso no está en el medio, lo cual significa que el 50% de los valores más pequeños están más cerca entre si que los valores del 50% superior. Además vemos tres puntos por encima de la linea (whishart). Estos son outliers. Tenemos 3 entre los datos más altos. Los outliers o valores extremos están más allá del valor que deja a su izquierda el 75% de la distribución multiplicado por 1.5. Todos estos datos se pueden observar usando la función *summary()* sobre un vector numérico:

```
summary(murders$total)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0    24.5    97.0   184.4   268.0  1257.0
```

Los boxplots son particularmente interesantes para comparar distribuciones de diferentes grupos de datos. Por ejemplo, como tenemos la información del número de asesinatos por region podemos hacer un boxplot del número de asesinatos para cada region:

```
boxplot(murders$total~murders$region)
```



La región con el menor número de asesinatos en general es "West", aunque hay dos estados que son outliers. Podemos buscarlo:

```
murders.west<-filter(murders,region=="West")
murders.west[which.max(murders.west$total),]
```

```
##        state abb region population total total.bin
## 3 California  CA   West   37253956  1257      1400
```

La región con un mayor número de asesinatos en general es "South"

Ejercicio #1: dataset *babies*

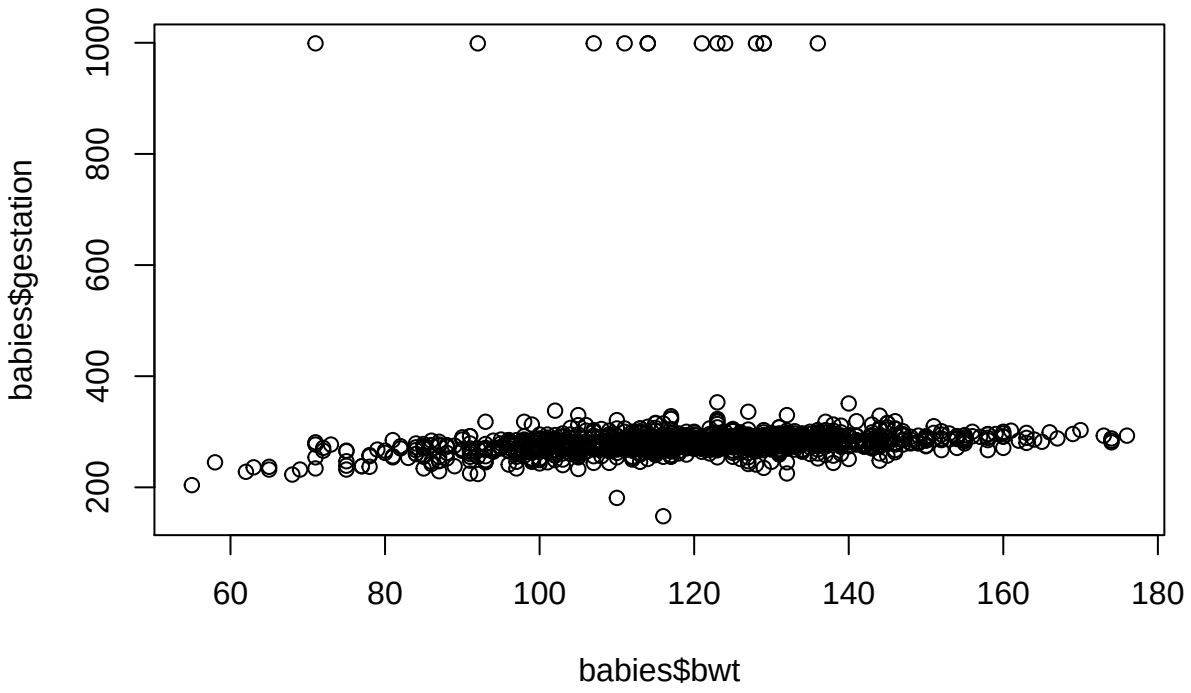
Utilizando el dataset "babies.txt"

1. Utilizando un scatterplot *plot()* encontrar si existe una relación de algún tipo entre el peso al nacimiento de los bebés y la edad gestacional (en semanas)
2. Compara utilizando un boxplot la distribución de los pesos al nacer de los niños con madres fumadoras frente a aquellos con madres no fumadoras
3. Explora usando un histograma la distribución general de los pesos de los bebes.

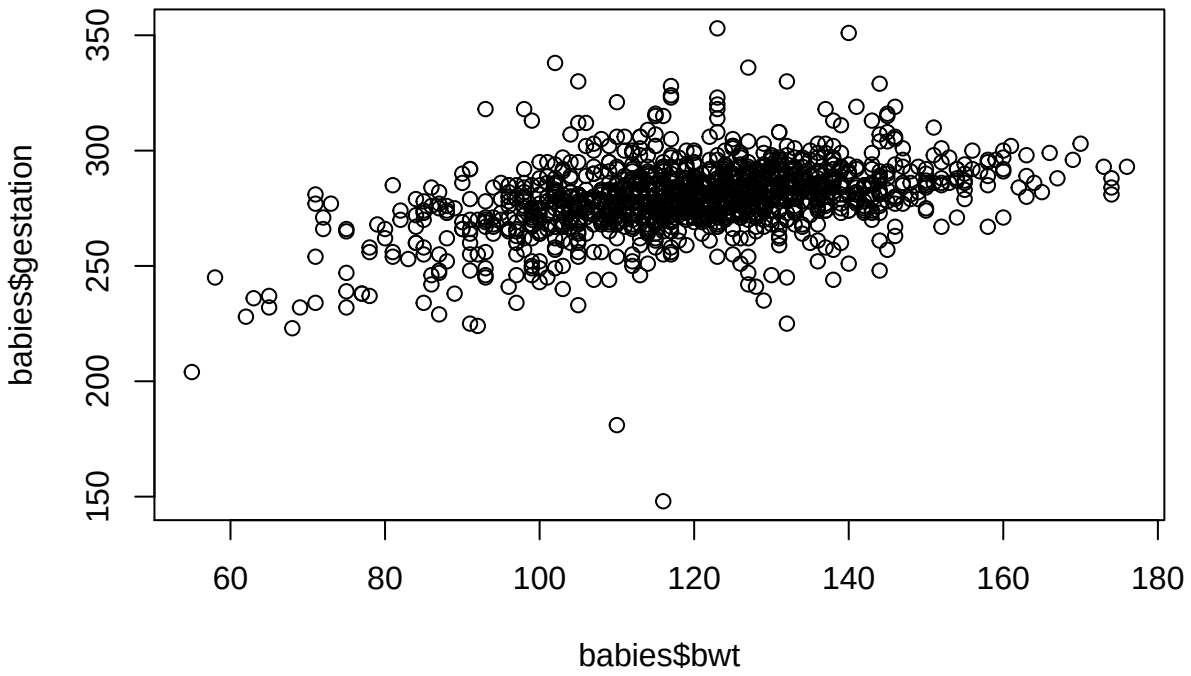
---

1. Utilizando un scatterplot *plot()* encontrar si existe una relación de algún tipo entre el peso al nacimiento de los bebés y la edad gestacional (en semanas)

```
setwd("C:/Users/fscabo/Desktop/MasterDataScience_KSchool/Ejercicios")
babies=read.delim("babies.txt",header=T,sep="\t",stringsAsFactors = F)
plot(babies$bwt,babies$gestation)
```

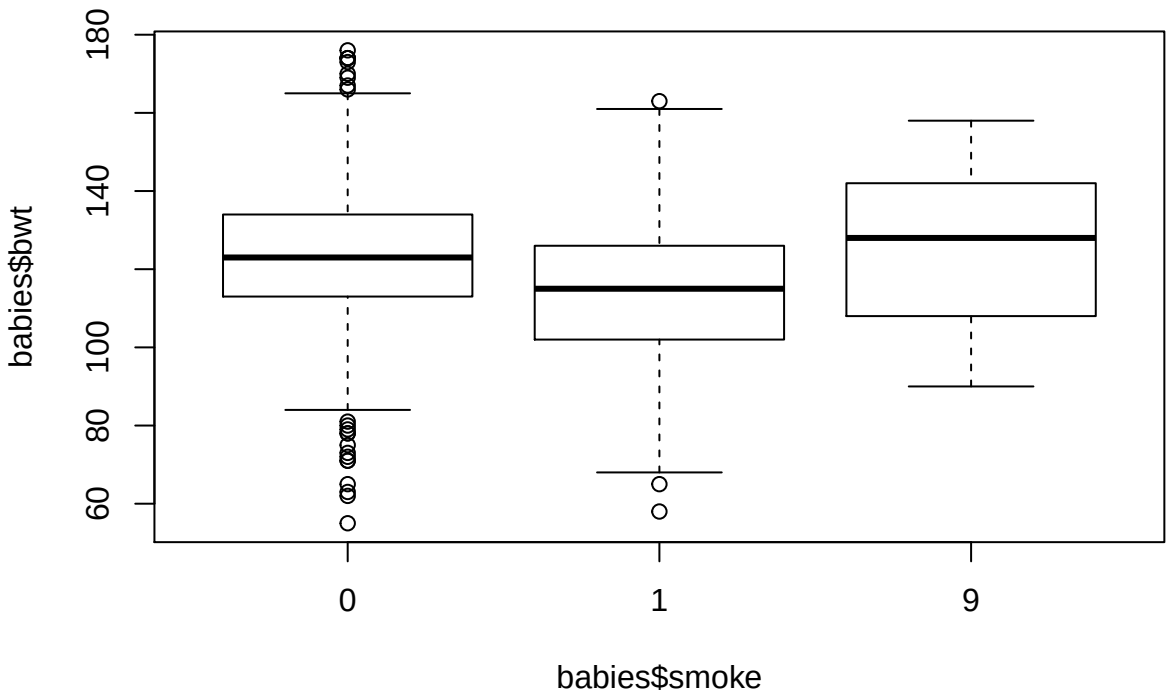



```
#necesitamos poner a NA los missing (999)
babies$gestation[which(babies$gestation=="999")]=NA
plot(babies$bwt,babies$gestation)
```



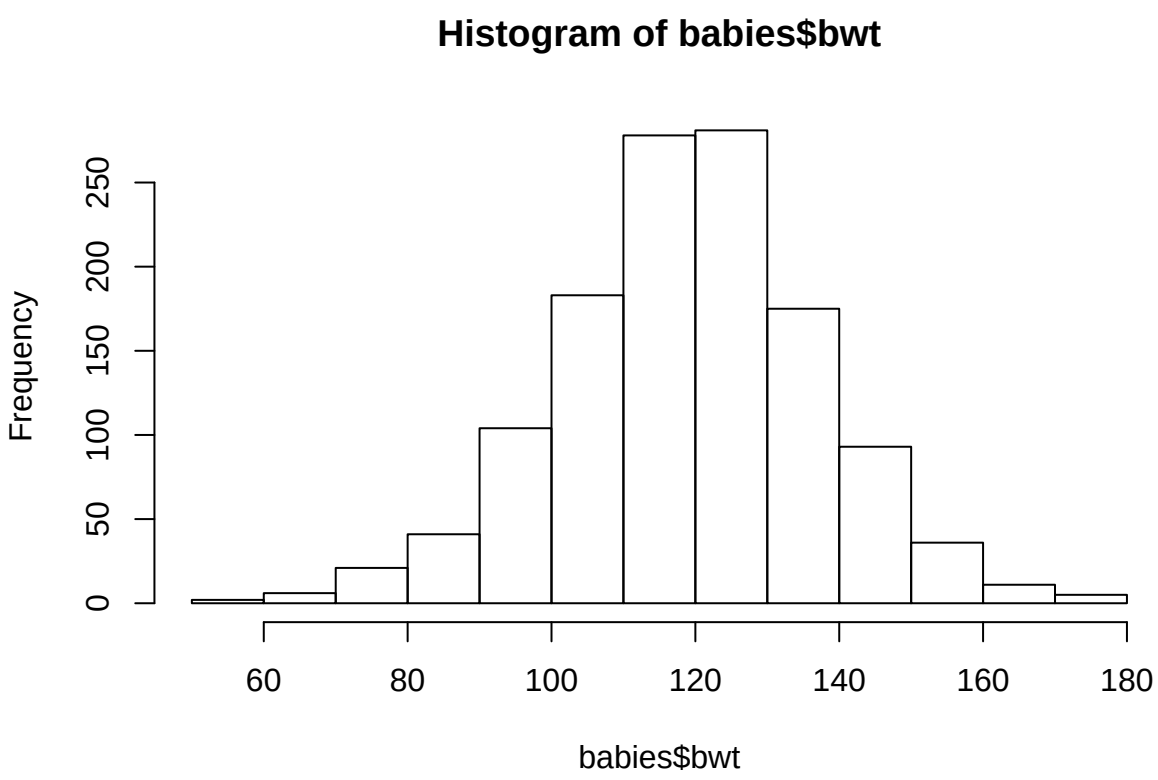
2. Compara utilizando un boxplot la distribución de los pesos al nacer de los niños con madres fumadoras frente a aquellos con madres no fumadoras

```
boxplot(babies$bwt~babies$smoke)
```



3. Explora usando un histograma la distribución general de los pesos de los bebes.

```
hist(babies$bwt)
```



Sumarizando datos y detectando outliers

Cuando tenemos muchos datos lo primero que solemos querer hacer es intentar resumir la información en un sólo número. En principio esto debería de darnos una idea acerca de algunas de las características importantes de nuestros datos. Queé medidas de sumarización se nos ocurren? La media, la mediana, la desviación estandar... ahora vamos a ver cuando usar cada una de ellas, cuando son y cuando no son informativas.

Medidas de resumen paramétricas: Media y desviación standard

Utilizando el ejemplo "babies.txt", calcular la media y la desviación estandar para los peso de los bebes:

```
mean(babies$bwt)
```

```
## [1] 119.5769
```

```
sqrt(var(babies$bwt))
```

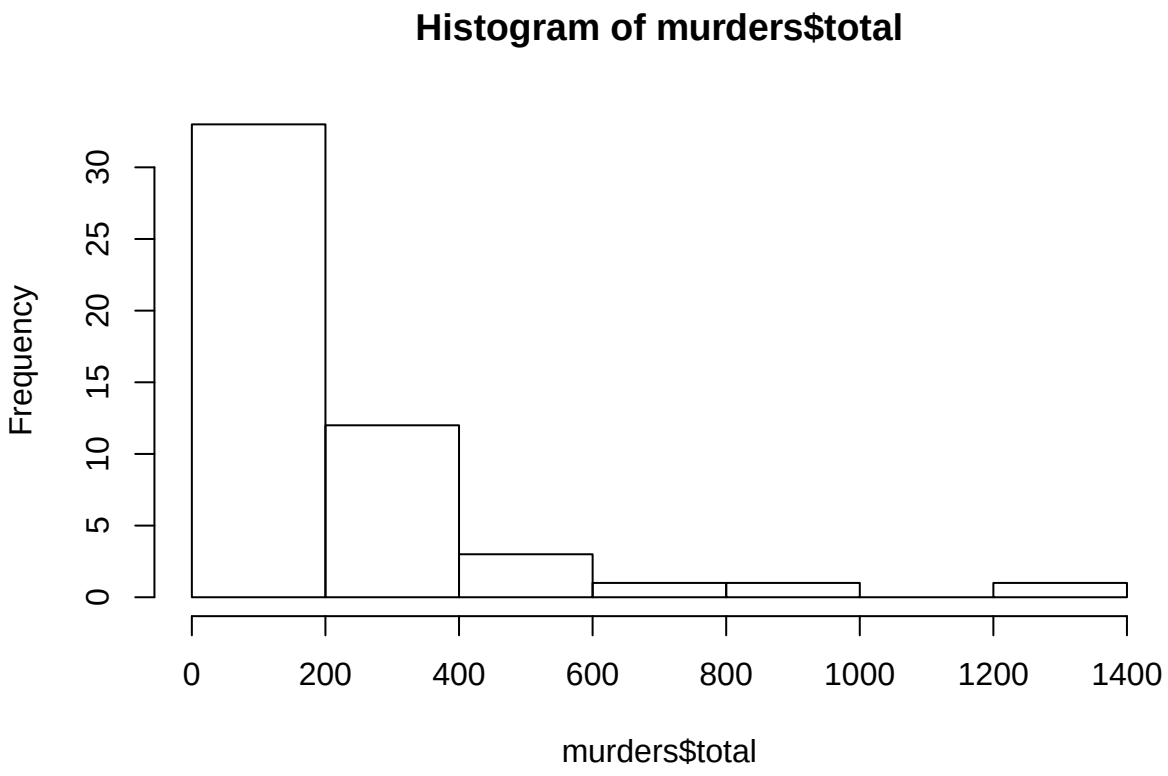
```
## [1] 18.23645
```

Recordando cómo era su distribución (simétrica, un sólo valor más probable...) parece que estos valores son bastantes informativos acerca de estos datos

Medidas de resumen no paramétricas: Mediana, IQR

Sin embargo, si miramos el histograma del total de asesinatos del ejemplo murder y calculamos los mismos parámetros

```
hist(murders$total)
```




```
mean(murders$total)
```

```
## [1] 184.3725
```

```
sqrt(var(murders$total))
```

```
## [1] 236.1261
```

Como veis estos dos valores nos harian pensar que hay numeros de asesinatos negativos en algunos estados, no nos da pistas acerca del numero mas habitual de asesinatos que encontramos y no podriamos saber que el numero de asesinatos es tan alto como 1400 en otros. Hay otros parámetros para sumarizar datos que no siguen una distribución "normal":

```
median(murders$total)
```

```
## [1] 97
```

```
IQR(murders$total)
```

```
## [1] 243.5
```

La mediana nos dice cual es el valor que deja el 50% de los datos a la izquierda de el y el 50% a su derecha. El IQR (Interquantile Range) nos da una idea de en que intervalo cae el 50% de los datos. Podemos calcular los quantiles de una distribucion usando:

```
summary(murders$total)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0    24.5    97.0   184.4   268.0  1257.0
```

```
summary(babies$bwt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    55.0   108.8   120.0   119.6   131.0   176.0
```

Si un conjunto de datos es aproximadamente normal (simétrico, con un sólo valor más probable, etc) en ese caso la media y la mediana son muy parecidas, como en el caso del peso de los bebes. Sin embargo, cuando una distribución no es parecida a una normal (murders$total) la media y la mediana son muy distintas y la media no suele ser muy informativa. Tampoco la desviación estandard. El IQR se calcula como Q_3-Q_1. Todos los valores que esten alejados 1.5 veces el IQR del q1 o de q3 se consideran outliers:

```
summary(murders$total)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0    24.5    97.0   184.4   268.0  1257.0
```

```
q1=quantile(murders$total, p=0.25)
q1
```

```
##  25%
## 24.5
```

```
q3=quantile(murders$total, p=0.75)
q3
```

```
## 75%
## 268
```

```
iqr=(q3-q1)
iqr
```

```
##   75%
## 243.5
```

```
r <- c(q1 - 1.5*iqr, q3 + 1.5*iqr)
r
```

```
##     25%     75%
## -340.75  633.25
```

Buscamos los outliers:

```
which(murders$total<=r[1])
```

```
## integer(0)
```

```
which(murders$total>=r[2])
```

```
## [1]  5 10 44
```

```
murders[which(murders$total>=r[2]),]
```

```
##         state abb region population total total.bin
## 5  California  CA   West   37253956  1257      1400
## 10    Florida  FL  South   19687653   669       800
## 44      Texas  TX  South   25145561   805      1000
```

Hay otro tipo de outliers aún más lejanos que son los *far_out* outliers

```
r2 <- c(q1 - 3*iqr, q3 + 3*iqr)
r2
```

```
##    25%    75%
## -706.0  998.5
```

```
which(murders$total>=r2[2])
```

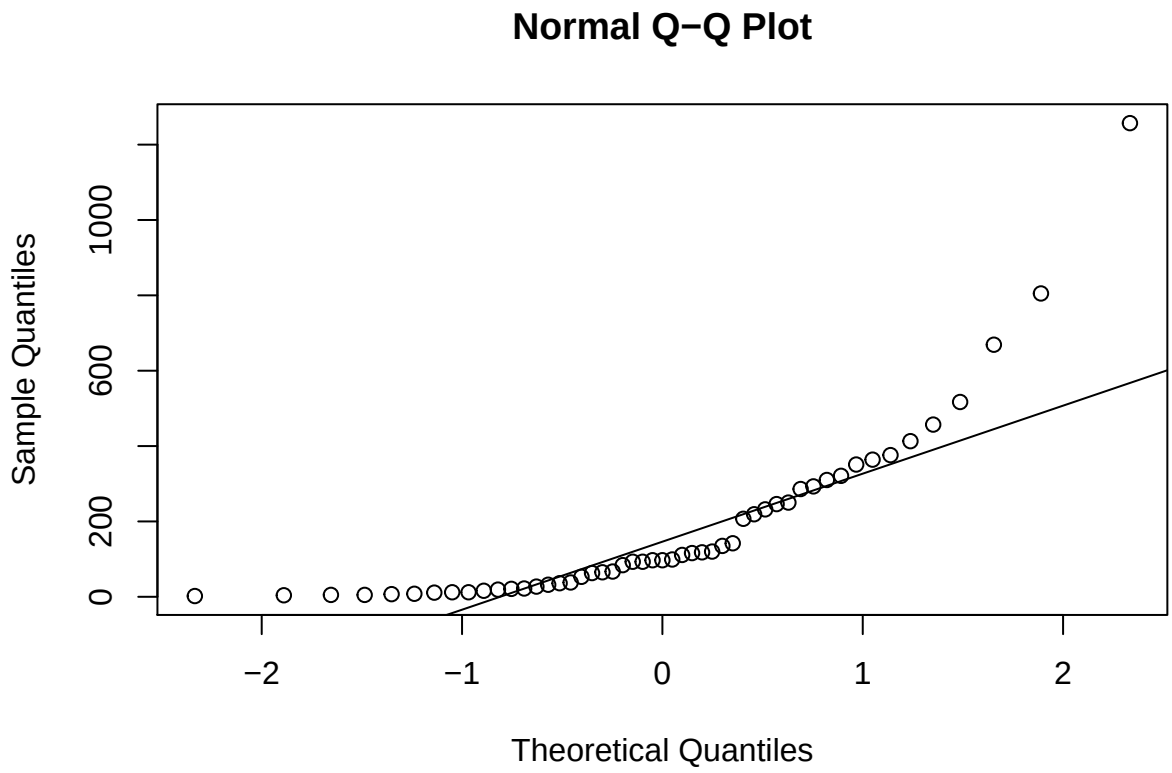
```
## [1] 5
```

```
murders[which(murders$total>=r2[2]),]
```

```
##        state abb region population total total.bin
## 5 California  CA   West   37253956  1257      1400
```

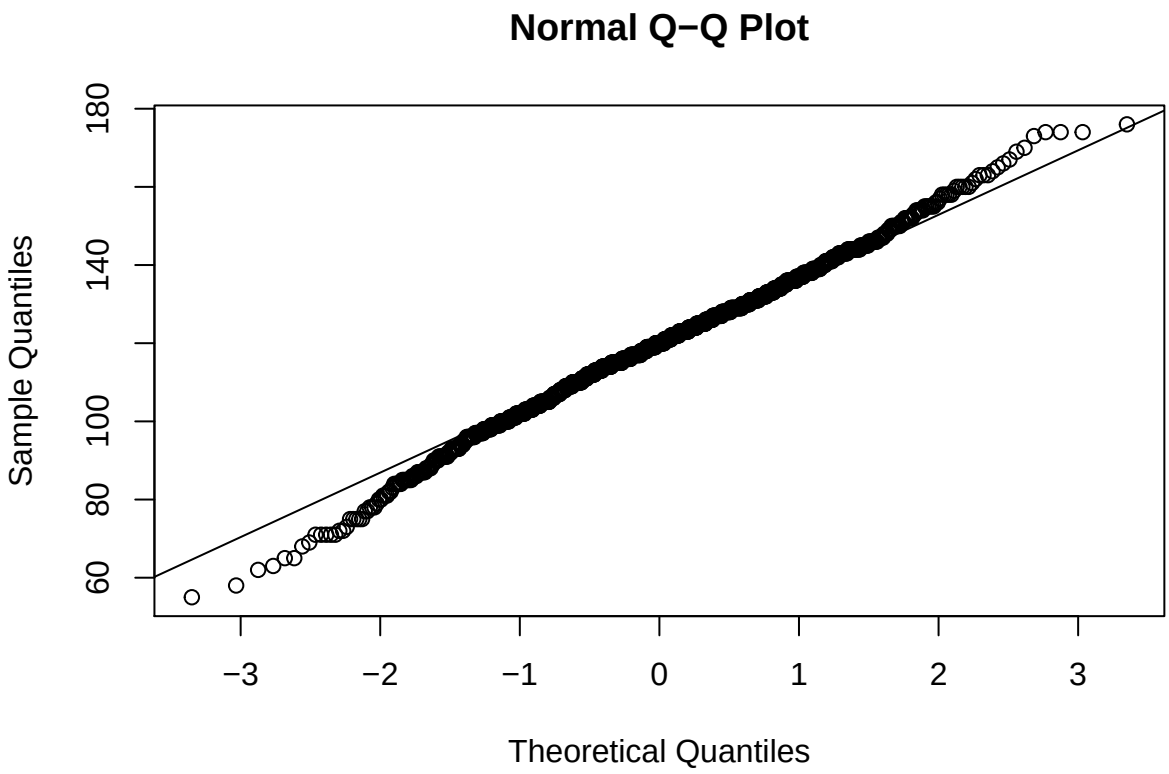
Un tipo de plot que no habíamos visto y que también es muy informativo es el que compara los quantiles de nuestros datos con los de una distribucion normal:

```
qqnorm(murders$total)
qqline(murders$total)
```



Frente a esto, la distribución de los pesos de los bebes tienen una distribución mas o menos normal, con la excepción de que las colas son mas pesadas.

```
qqnorm(babies$bwt)
qqline(babies$bwt)
```



Finalmente, para datos no normales hay otra medida que nos da una idea de la dispersión, la *mad*: median absolute deviation. Se trata de la suma de la distancia absoluta entre cada valor y la mediana.

```
mad(babies$bwt)
```

```
## [1] 16.3086
```

```
mad(murders$total)
```

```
## [1] 126.021
```

Sumarizando data con *dplyr()*

summarize()

Vamos a utilizar los datos de alturas del paquete dslabs

```
library(dslabs)
data(heights)
head(heights)
```

```
##      sex height
## 1   Male     75
## 2   Male     70
## 3   Male     68
## 4   Male     74
## 5   Male     61
## 6 Female     65
```

```
str(heights)
```

```
## 'data.frame':    1050 obs. of  2 variables:
##  $ sex   : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 1 1 1 1 2 ...
##  $ height: num  75 70 68 74 61 65 66 62 66 67 ...
```

La función *summarize()* del paquete *dplyr* nos calcula cualquier agregado que le pidamos de un vector de un data.frame o de un tibble. Como el input era un data.frame() el output también lo es.

```
s <- heights %>%
  filter(sex == "Male") %>%
  summarize(average = mean(height), standard_deviation = sd(height))
s
```

```
##    average standard_deviation
## 1 69.31475           3.611024
```

```
str(s)
```

```
## 'data.frame':    1 obs. of  2 variables:
##  $ average           : num 69.3
##  $ standard_deviation: num 3.61
```

Como ya comentamos anteriormente, para datos que no siguen una distribución normal o gausiana es mejor utilizar la mediana, mad o IQR. Podemos usar otra vez la función *summarize* para el data de total murders:

```
s <- murders %>%
  summarize(median = median(total), mad=mad(total),min=min(total),max=max(total))
s
```

```
##   median     mad min  max
## 1     97 126.021   2 1257
```

```
str(s)
```

```
## 'data.frame':    1 obs. of  4 variables:
##  $ median: num 97
##  $ mad   : num 126
##  $ min   : num 2
##  $ max   : num 1257
```

NOTA: con la función *summarize* solo podemos llamar funciones que devuelvan un solo valor.

dot

Recordemos en el último ejercicio de la sesión II necesitábamos sumarizar el rate por estado del dataset murders para poder unirlo a la tabla con el rate de todos los países del mundo.

```
s <- murders %>%
  mutate(rate=total/population*100000) %>%
  summarize(mean(rate))
s
```

```
##   mean(rate)
## 1   2.779125
```

```
str(s)
```

```
## 'data.frame':    1 obs. of  1 variable:
##  $ mean(rate): num 2.78
```

Como las funciones de dplyr devuelven el mismo tipo de objeto que su input en este caso queremos acceder sólo al valor que tienen almacenado. Podemos hacerlo asi:

```
s
```

```
##   mean(rate)
## 1   2.779125
```

```
s %>% .$rate
```

```
## NULL
```

"." simplemente reemplaza al objeto que pasamos por el pipe, en este caso s que es n data frame. Por eso accedemos su información con $

```
s <- murders %>%
  summarize(rate=mean(total)/mean(population)*100000) %>%
  .$rate
s
```

```
## [1] 3.034555
```

group_by()

```
babies.new<-babies %>%
  select(bwt, smoke) %>%
       group_by(smoke)
str(babies.new)
```

```
## Classes 'grouped_df', 'tbl_df', 'tbl' and 'data.frame':  1236 obs. of  2 variables:
##  $ bwt  : int  120 113 128 123 108 136 138 132 120 143 ...
##  $ smoke: int  0 0 1 0 1 0 0 0 0 1 ...
##  - attr(*, "groups")=Classes 'tbl_df', 'tbl' and 'data.frame':   3 obs. of  2 variables:
##   ..$ smoke: int  0 1 9
##   ..$ .rows:List of 3
##   .. ..$ : int  1 2 4 6 7 8 9 11 15 16 ...
##   .. ..$ : int  3 5 10 12 13 14 17 18 20 22 ...
##   .. ..$ : int  170 219 255 256 433 601 642 666 672 921
##   ..- attr(*, ".drop")= logi TRUE
```

```
babies.new
```

```
## # A tibble: 1,236 x 2
## # Groups:   smoke [3]
##      bwt smoke
##    <int> <int>
##  1   120     0
##  2   113     0
##  3   128     1
##  4   123     0
##  5   108     1
##  6   136     0
##  7   138     0
##  8   132     0
##  9   120     0
## 10   143     1
## # ... with 1,226 more rows
```

```
babies %>%
  select(bwt, smoke) %>%
       group_by(smoke)  %>%
  summarize(mean(bwt))
```

```
## # A tibble: 3 x 2
##   smoke `mean(bwt)`
##   <int>       <dbl>
## 1     0        123.
## 2     1        114.
## 3     9        127.
```

Ordenar data.frames: *arrange() top_n()*

La función *arrange()* ordena tablas enteras por una variable

```
murders %>% arrange(population) %>% head()
```

```
##                  state abb        region population total total.bin
## 1              Wyoming  WY          West     563626     5       200
## 2 District of Columbia  DC         South     601723    99       200
## 3              Vermont  VT     Northeast     625741     2       200
## 4         North Dakota  ND North Central     672591     4       200
## 5               Alaska  AK          West     710231    19       200
## 6         South Dakota  SD North Central     814180     8       200
```

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(rate) %>%
  head()
```

```
##           state abb        region population total total.bin      rate
## 1       Vermont  VT     Northeast     625741     2       200 0.3196211
## 2 New Hampshire  NH     Northeast    1316470     5       200 0.3798036
## 3        Hawaii  HI          West    1360301     7       200 0.5145920
## 4  North Dakota  ND North Central     672591     4       200 0.5947151
## 5          Iowa  IA North Central    3046355    21       200 0.6893484
## 6         Idaho  ID          West    1567582    12       200 0.7655102
```

Si tenemos empates podemos usar una segunda columna para deshacer dicho empate:

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(total,rate) %>%
  head()
```

```
##           state abb        region population total total.bin      rate
## 1       Vermont  VT     Northeast     625741     2       200 0.3196211
## 2  North Dakota  ND North Central     672591     4       200 0.5947151
## 3 New Hampshire  NH     Northeast    1316470     5       200 0.3798036
## 4       Wyoming  WY          West     563626     5       200 0.8871131
## 5        Hawaii  HI          West    1360301     7       200 0.5145920
## 6  South Dakota  SD North Central     814180     8       200 0.9825837
```

Por último podemos seleccionar las primeras filas de un data.frame o de un tibble usando la función *top_n()*. Nota que la función *desc()* indica que se ordena de manera descendente el data.frame.

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(desc(rate)) %>%
  top_n(10)
```

```
## Selecting by rate
```

```
##                   state abb        region population total total.bin
## 1  District of Columbia  DC         South     601723    99       200
## 2             Louisiana  LA         South    4533372   351       400
## 3              Missouri  MO North Central    5988927   321       400
## 4              Maryland  MD         South    5773552   293       400
## 5        South Carolina  SC         South    4625364   207       400
## 6              Delaware  DE         South     897934    38       200
## 7              Michigan  MI North Central    9883640   413       600
## 8           Mississippi  MS         South    2967297   120       200
## 9               Georgia  GA         South    9920000   376       400
## 10              Arizona  AZ          West    6392017   232       400
##         rate
## 1  16.452753
## 2   7.742581
## 3   5.359892
## 4   5.074866
## 5   4.475323
## 6   4.231937
## 7   4.178622
## 8   4.044085
## 9   3.790323
## 10  3.629527
```

Ejercicio #2: Distribución de las alturas de los estudiantes

Recogemos la altura, el género y la edad de todos los miembros de la clase.

- Escribir los datos en un fichero y guardalo en tu directorio de trabajo con el nombre: "alturas.txt"
- Leelo en R en el objeto "altura"
- Haz un scatterplot que relacione la altura con la edad
- Que distribucion tiene la altura? Y la edad? Cual seria la mejor forma de sumarizar la altura? Y la edad?
- Haz un boxplot que muestre la distribución de alturas en hombres frente a mujeres
- Hay algun outlier?
- Utilizando *dplyr* sumariza la altura de los hombres y de las mujeres por separado
- quienes son los tres hombres mas altos de la clase? Y las tres mujeres más jóvenes?

Visualizando datos con R: *ggplot2()*

Hay otras librerias para visualizar datos en R, como por ejemplo las funciones de la instalacion base que ya hemos visto, *grid* o *lattice*. Sin embargo, ggplot2 se basa en la llamada *grammar of graphics*: al igual que bloques gramaticales básicos permiten crear cientos de frases en *ggplot2* un pequeño número de comandos permite crear gráficos muy distintos.

En particular, *ggplot2* está articulado sobre tres conceptos básicos que tienen que definirse cada vez que vamos a plotear algo: **data**, **geometry** y **aesthetics**

- Data:

Los datos en *ggplot2()* tienen que estar en formato tidy. Ya hemos visto que esto significa que cada elemento a dibujar está indexado por un sólo índice: el número de la fila. Eso si, para cada elemento a representar podemos tener múltiples atributos. El data.frame *murders* es tidy porque cada estado aparece en una sola fila aunque hay varios atributos (region, total, population) para cada estado. Si en lugar de una foto fija

tuvieramos una serie temporal cada combinacion (estado, año) sería la “key” de la tabla y apareceria tambien una sola vez en la tabla en una fila.

- Geometry:

Queremos representar un scatterplot? Un histograma? Un boxplot?

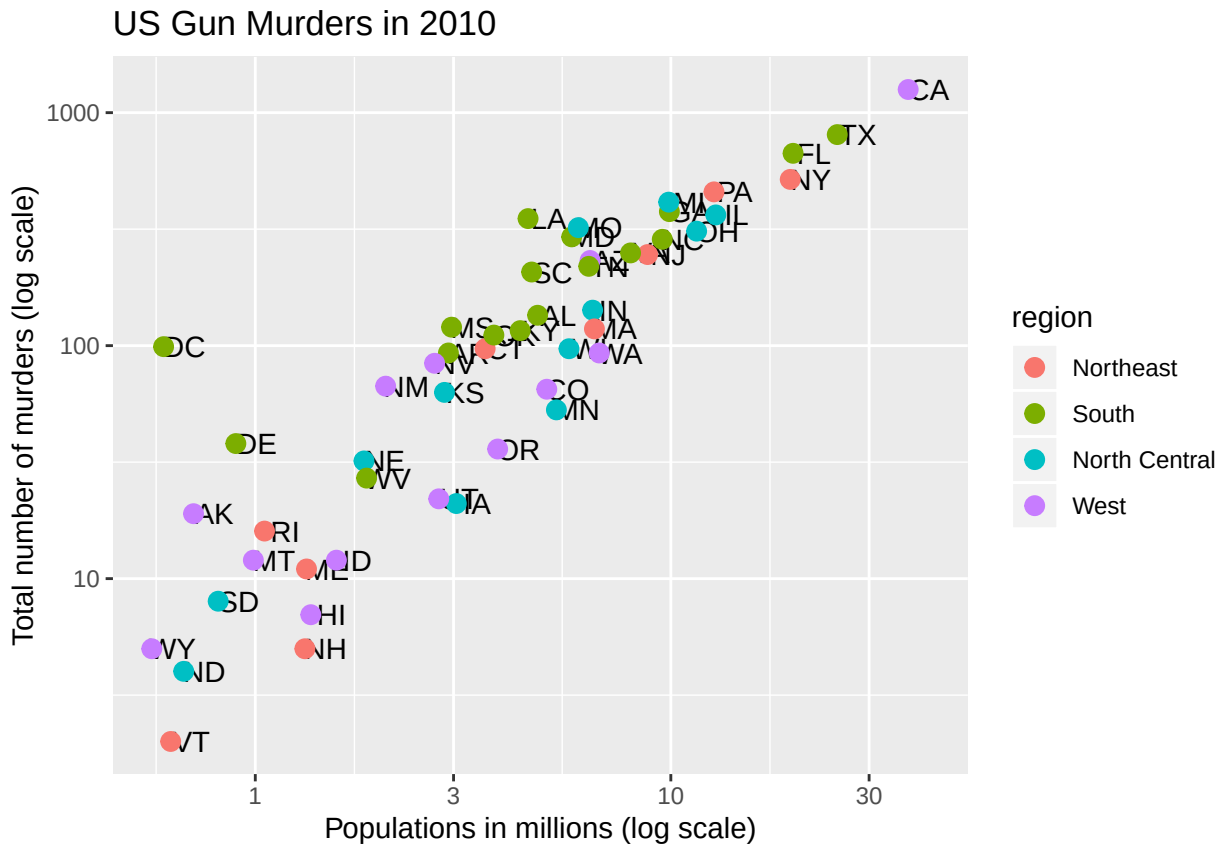
- Aesthetic Mapping:

Que represento en el eje de las x? Que represento en el eje de las y? Que color uso? Que letra uso? Parte de estos parámetros dependerán de la geometría del plot

Ejercicio 1:

Para el siguiente plot, describe los datos, la geometria y el aesthetic mapping.

```
##
## Attaching package: 'data.table'
## The following object is masked from 'package:purrr':
##
##     transpose
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```



- Data: murders
- Geometria: Scatterplot

- Aesthetics Mapping:

  ++ x-axis: population size

  ++ y-axis: total number of murders

  ++ text: states

  ++ colors: the four different regions

Creando un plot con *ggplot2()*

El primer paso es "inicializar" el plot diciéndole que datos queremos usar y algunas características básicas de el.

```
ggplot(data = murders)
```



```
#alternativamente con pipe:
murders %>% ggplot()
```

```
p <- ggplot(data = murders)
class(p)
```

```
## [1] "gg"     "ggplot"
```

```
print(p)
```

```
p
```

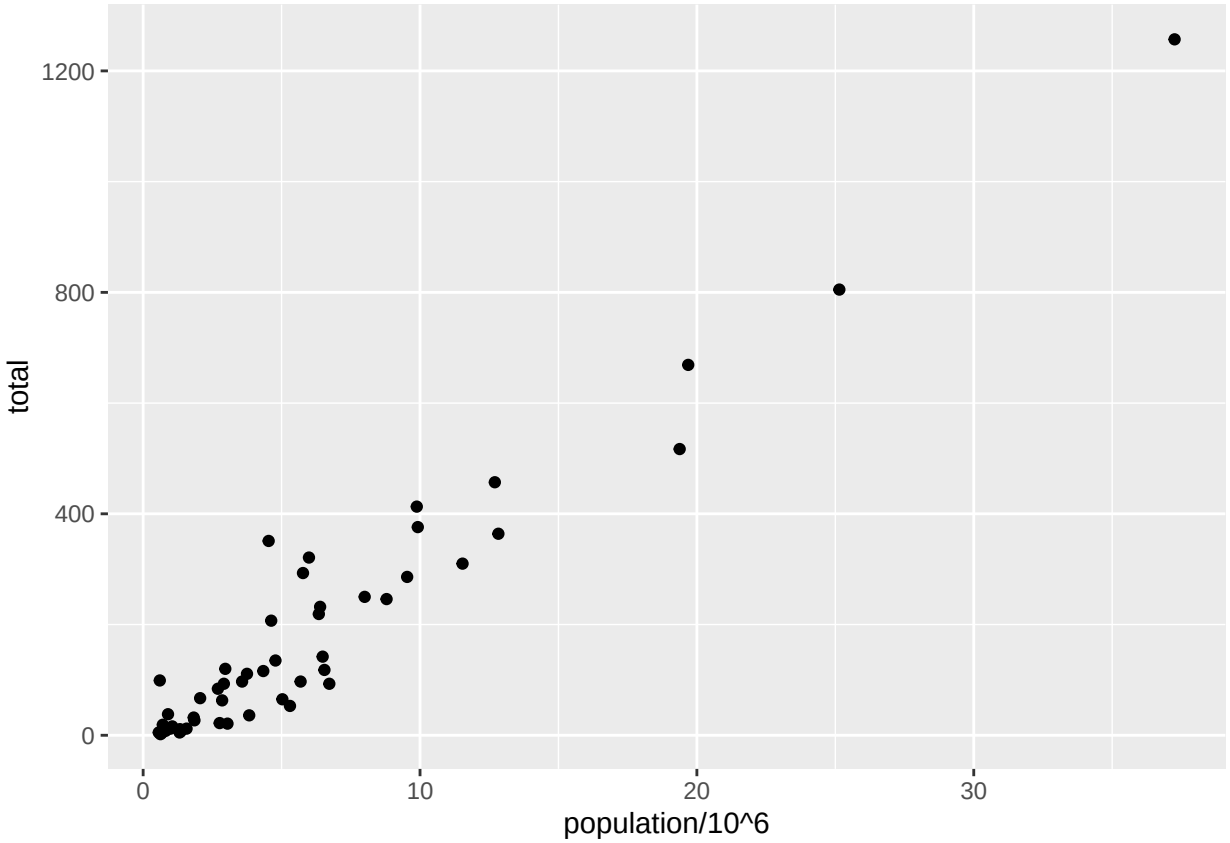
Geometria

La geometría define el tipo de representación de nuestros datos que estamos usando: cómo se situan los valores en el plano o en el espacio.

Cada geometría necesita una serie de parámetros fijos para poder pintar y otros opcionales

Por ejemplo, un scatterplot con la función *geom_point* requiere obligatoriamente los valores en x y en y. Además se le puede dar color, tamaño, etc. Queremos crear un scatterplot:

```
? geom_point
murders %>% ggplot() +
  geom_point(aes(x = population/10^6, y = total))
```



Habéis visto que hemos añadido la geometría al plot inicial usando +. Así es como se concatenan las distintas capas de un plot

Aesthetics

Cuando hacemos un gráfico estamos transformando nuestro datos en valores que componen el gráfico final. La iniciativa *grammar of graphics (gg)* lo que intenta es hacer esto de manera sistemática sea cual sea el tipo de geometría que vamos a utilizar.

Con la función *aes()* estamos tratando de asignar a valores de nuestros datos (de nuestro data frame *murders* en este caso) acaracterísticas cuantificables del gráfico. Estas características son las aesthetics. Da igual si se trata de un pie chart, de un scatterplot... al final ggplot2 lo que necesita es saber qué pone en cada dimensión (x,y) para gráficos de dos dimensiones, (x,y,z) para gráficos 3D, el color, la fuente de la letra... Necesita saber que valor pone en cada pixel de la pantalla y cuales son sus características. Esto es lo que tratamos de hacer con aes(): mapear valores en características de un gráfico

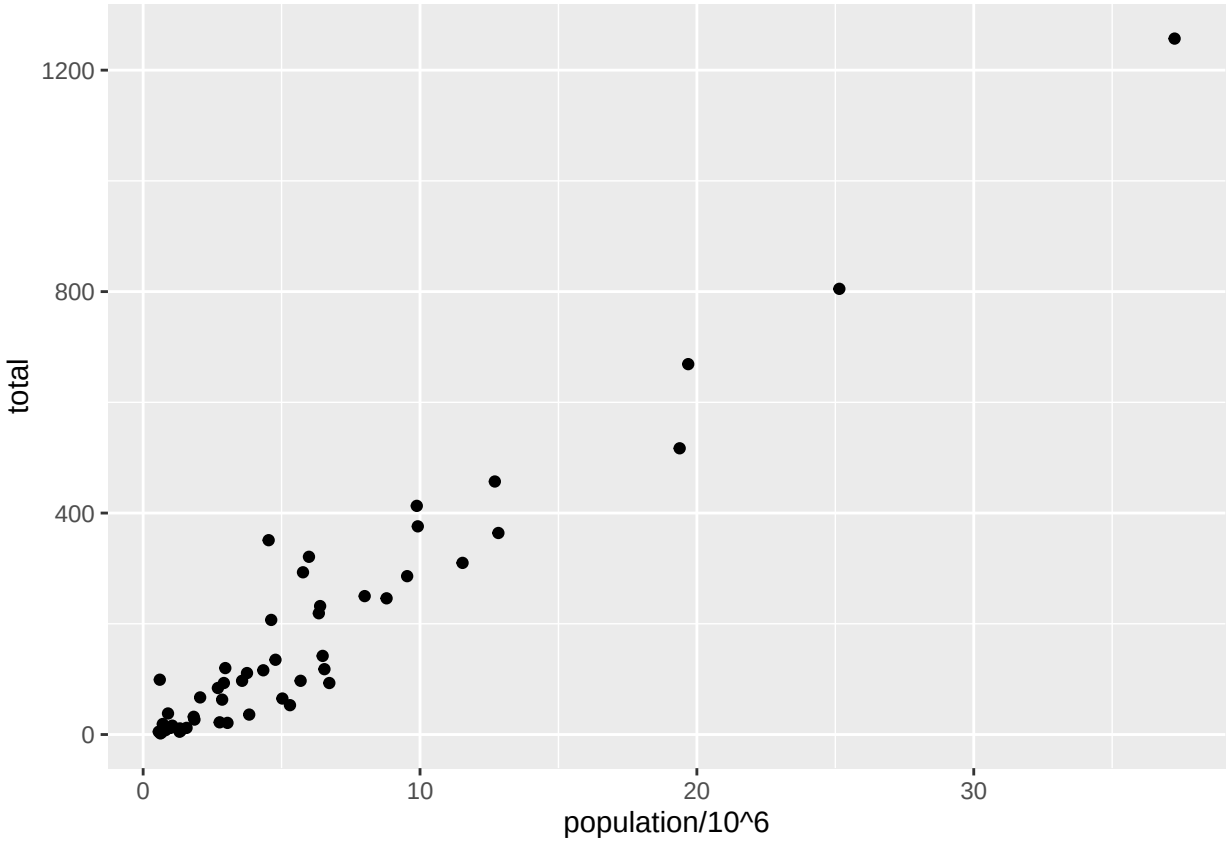
La función *aes()* es la que le indica a la geometría que necesita pintar en el plot y cómo. Además se le pueden dar muchas propiedades como color, tamaño, etc, en función de los datos. Cada geometría requiere un cierto tipo de mapeo datos/visualización.

Global vs local aesthetics

Las características estéticas de un gráfico pueden definirse de manera global al inicializar el plot. De esta forma cada nueva capa heredará estas características globales.

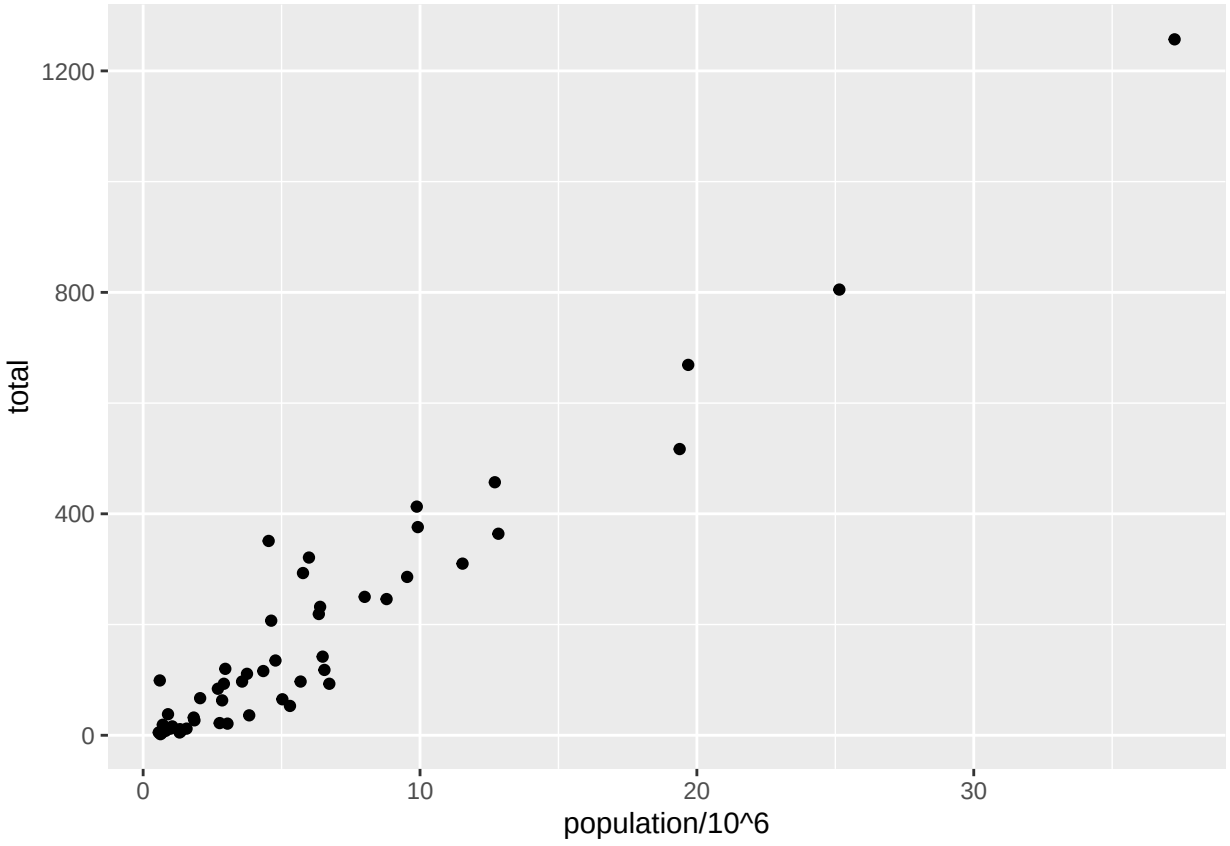
Si le hubieramos pasado las estéticas al plot anterior no haria falta dárselas a geom_point utilizaria las del objeto ggplot2

```
murders %>% ggplot(aes(x = population/10^6, y = total)) +
  geom_point()
```



También podríamos añadir la capa con la geometría al objeto p

```
p<-murders %>% ggplot(aes(x = population/10^6, y = total))

p+geom_point(aes(x = population/10^6, y = total))
```



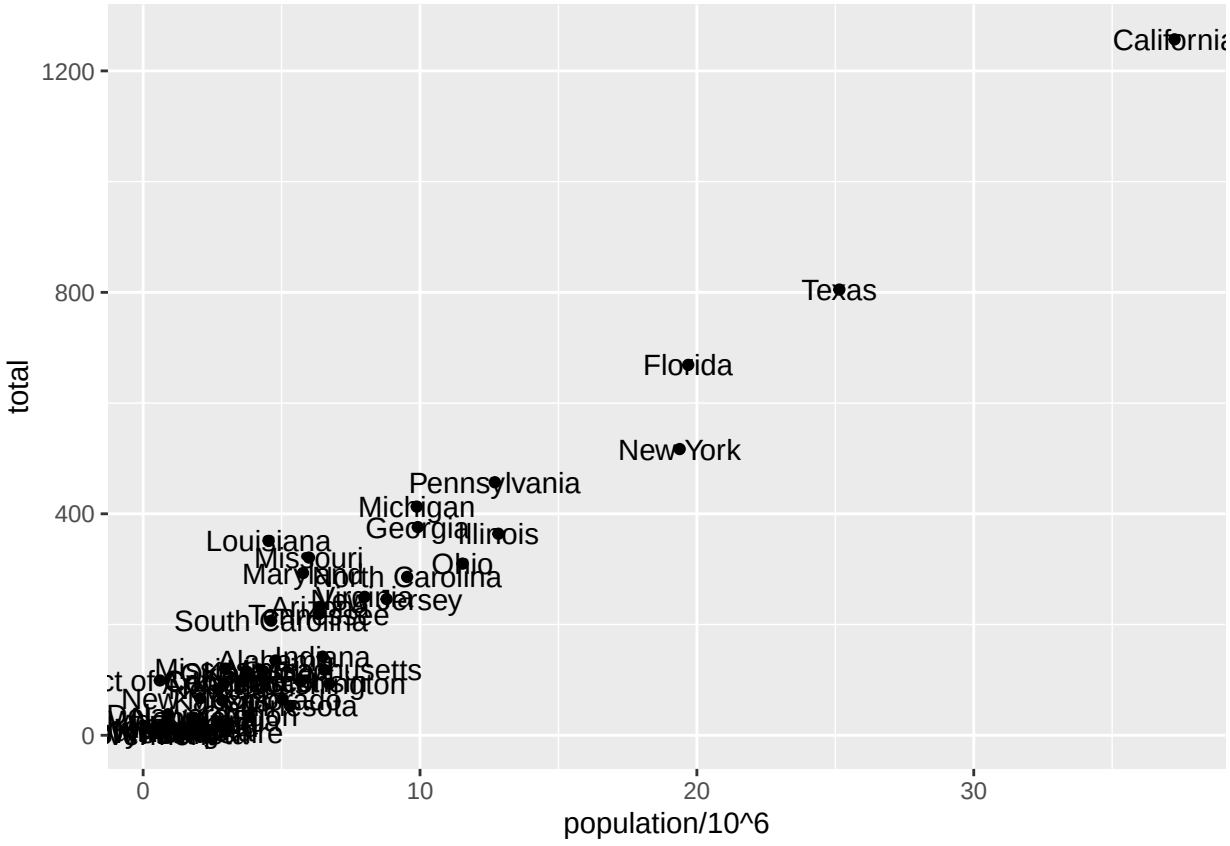
##Capas

Los gráficos en *ggplot2* se definen usando diferentes capas que se unen unas a otras usando *+*

Una vez que hemos creado un objeto *ggplot()* como antes le vamos añadiendo capas, la primera de ellas siempre es la geometria. Después podemos seguir añadiendo características.

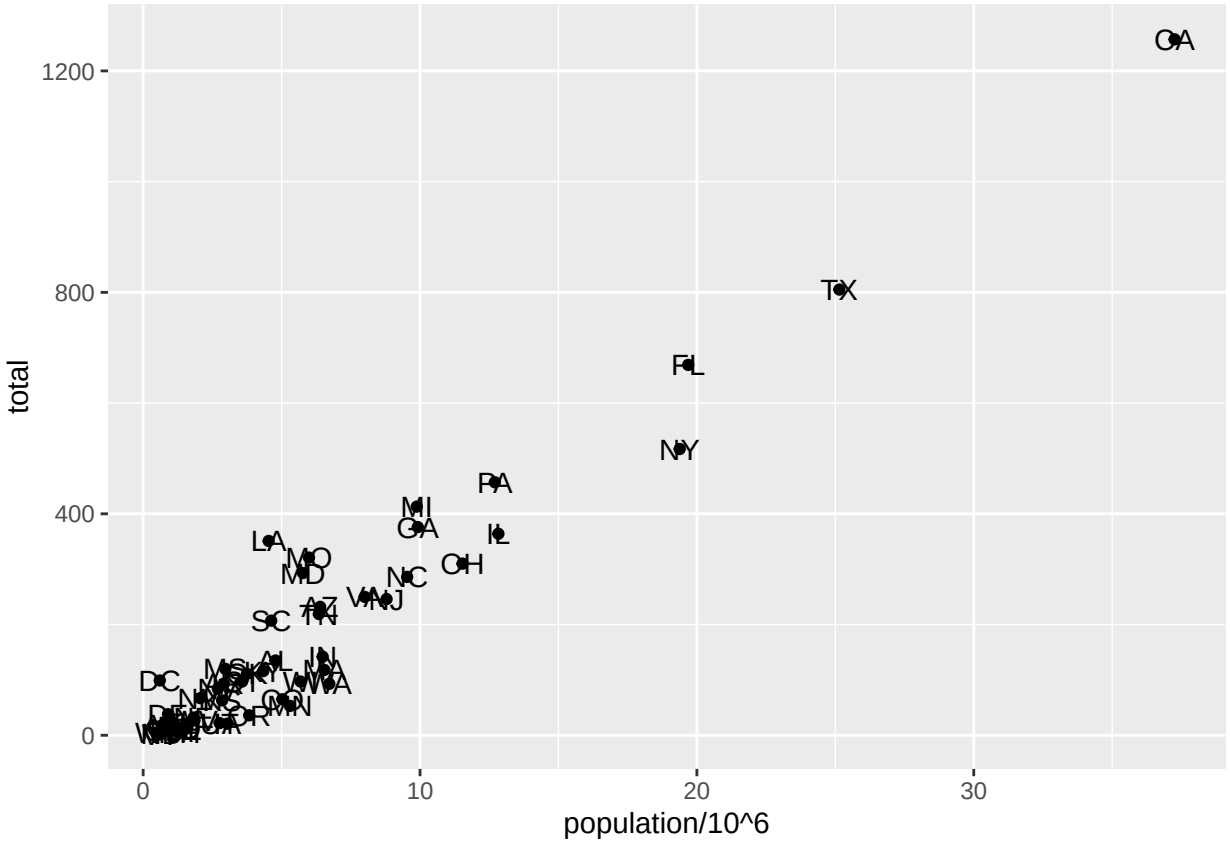
Otra capa que queremos añadir es texto para cada punto. Se utiliza la función *geom_text()*

```
p+geom_point(aes(x = population/10^6, y = total))+
  geom_text(aes(x = population/10^6, y = total,label=state))
```



Si queremos visualizar la abreviación

```
p+geom_point(aes(x = population/10^6, y = total))+
  geom_text(aes(x = population/10^6, y = total,label=abb))
```



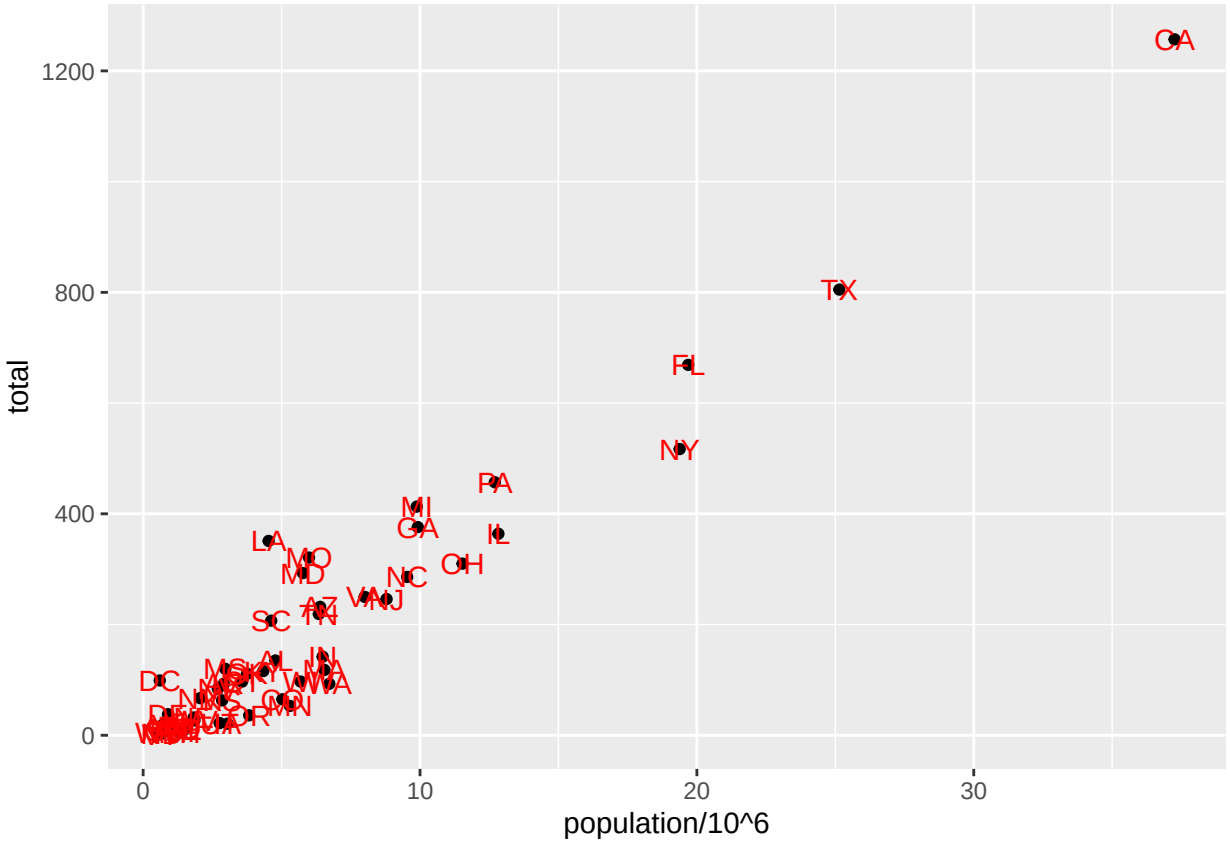
Fuera de *aes()* no se reconocen las variables de los objetos para las que queremos mapear características estéticas

```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

```
## Error in layer(data = data, mapping = mapping, stat = stat, geom = GeomText, : object 'abb' not found
```

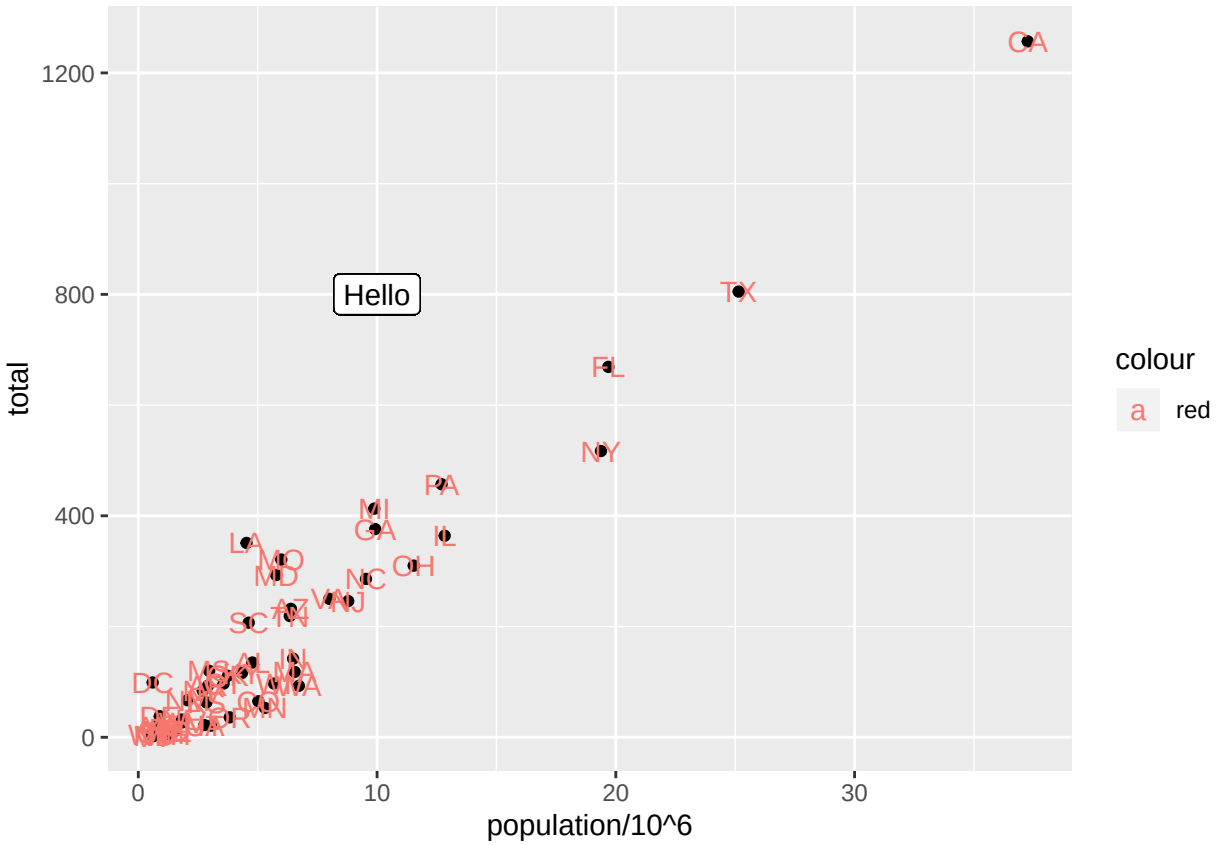
Cuando en una capa no se requiere ese mapeo dato-característica del plot podemos no usar la función aes(), pero para ello tenemos que haber definido globalmente las características básicas que necesita la función:

```
murders %>%
  ggplot(aes(x = population/10^6, y = total,label=abb))+
  geom_point()+
  geom_text(col="red")
```



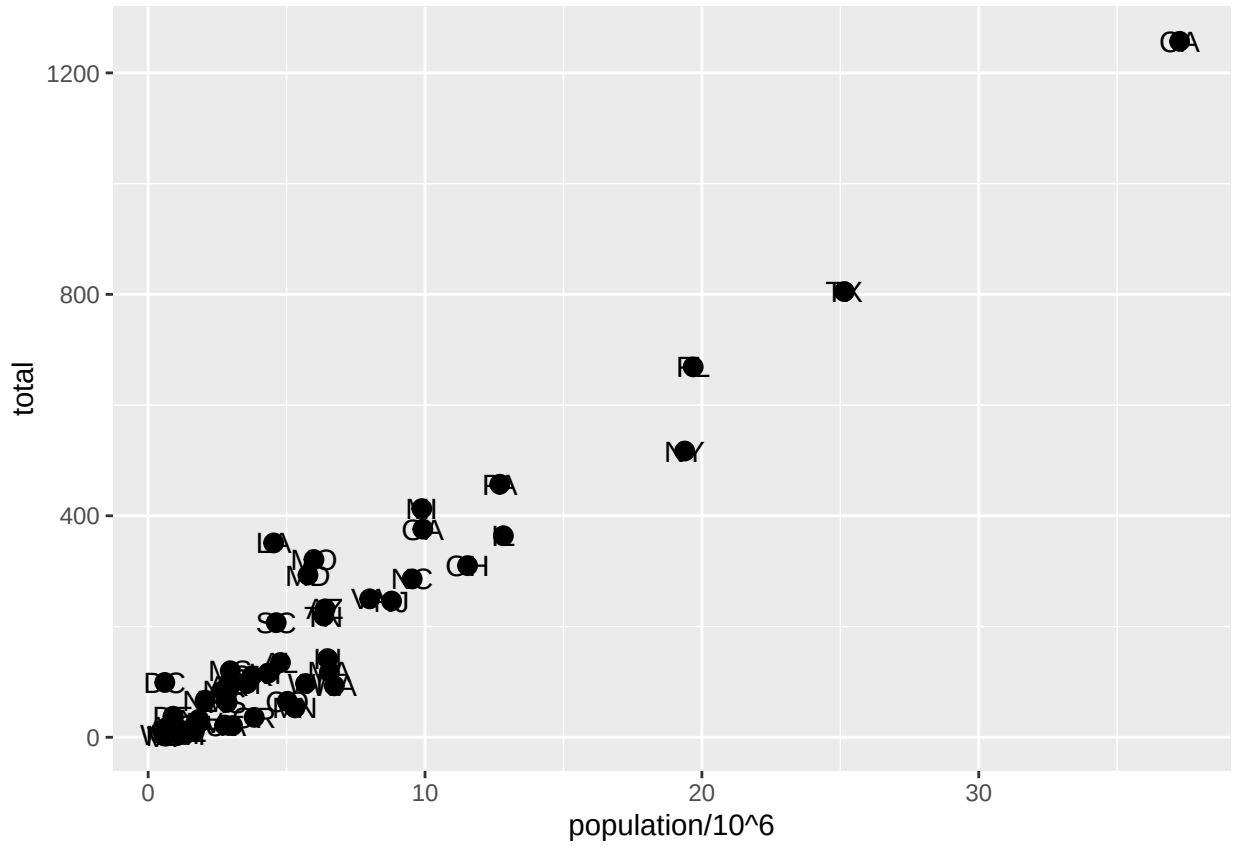
Las capas van tomando ese mapeado global hasta que lo sobre-escribimos:

```
murders %>% ggplot(aes(x = population/10^6, y = total,label=abb))+
  geom_point()+
  geom_text(aes(col="red"))+
  geom_label(aes(x=10,y=800,label="Hello"))
```

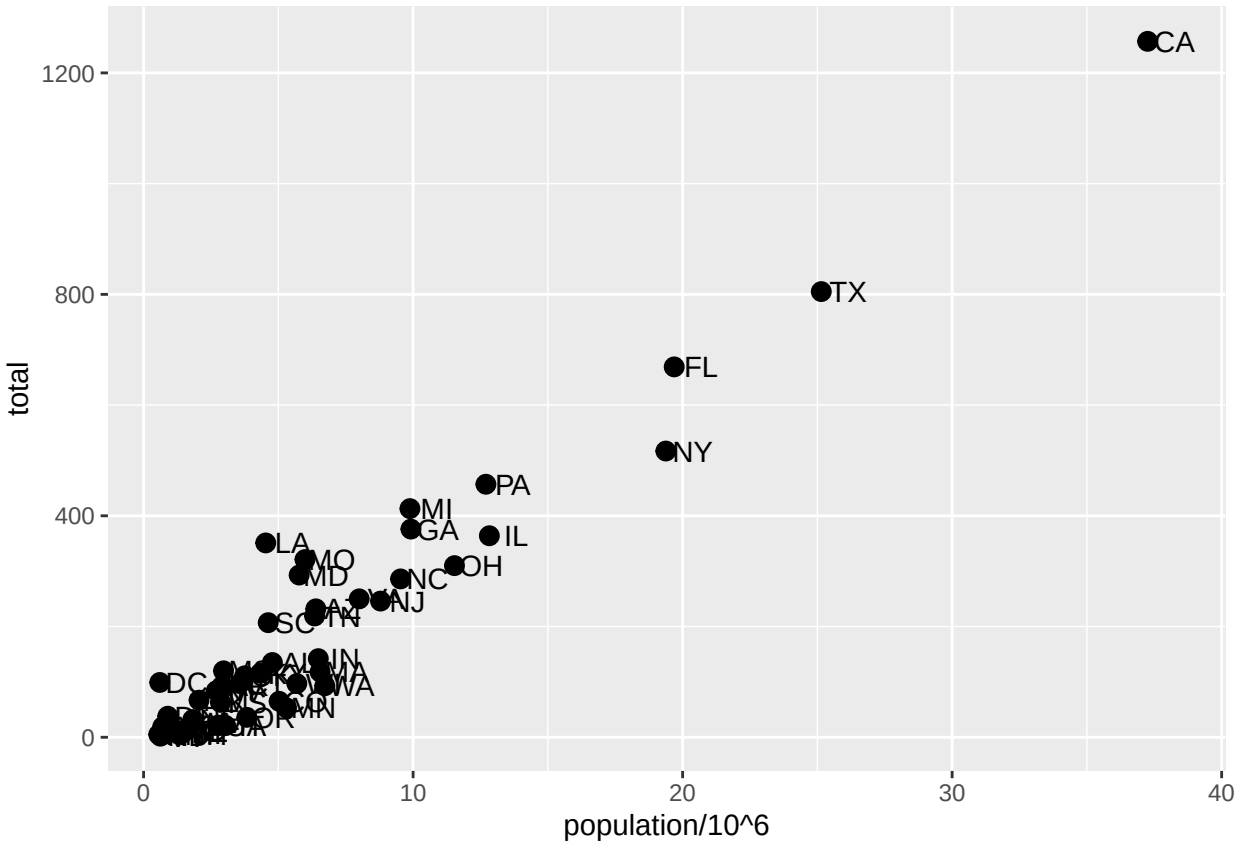


Cada función requiere diferentes parámetros unos que son *aesthetics* y otros fijos. La diferencia es que aesthetics mapea datos a propiedades estéticas (para uno o para todos los datos) mientras que lo que está fuera de *aes()* afecta a todo el plot, no hay un mapeo dato->estética.

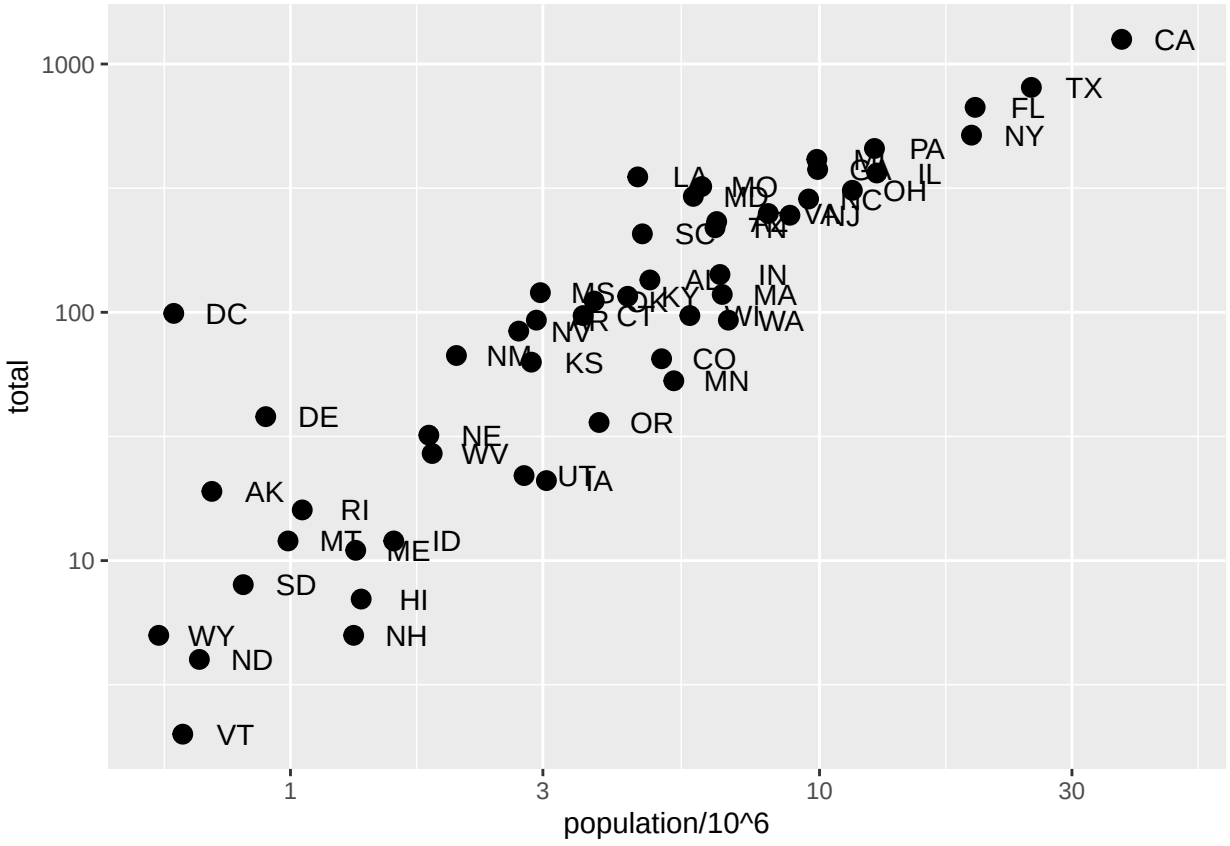
```
p<-murders %>% ggplot()
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```




```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb),nudge_x=1)
```



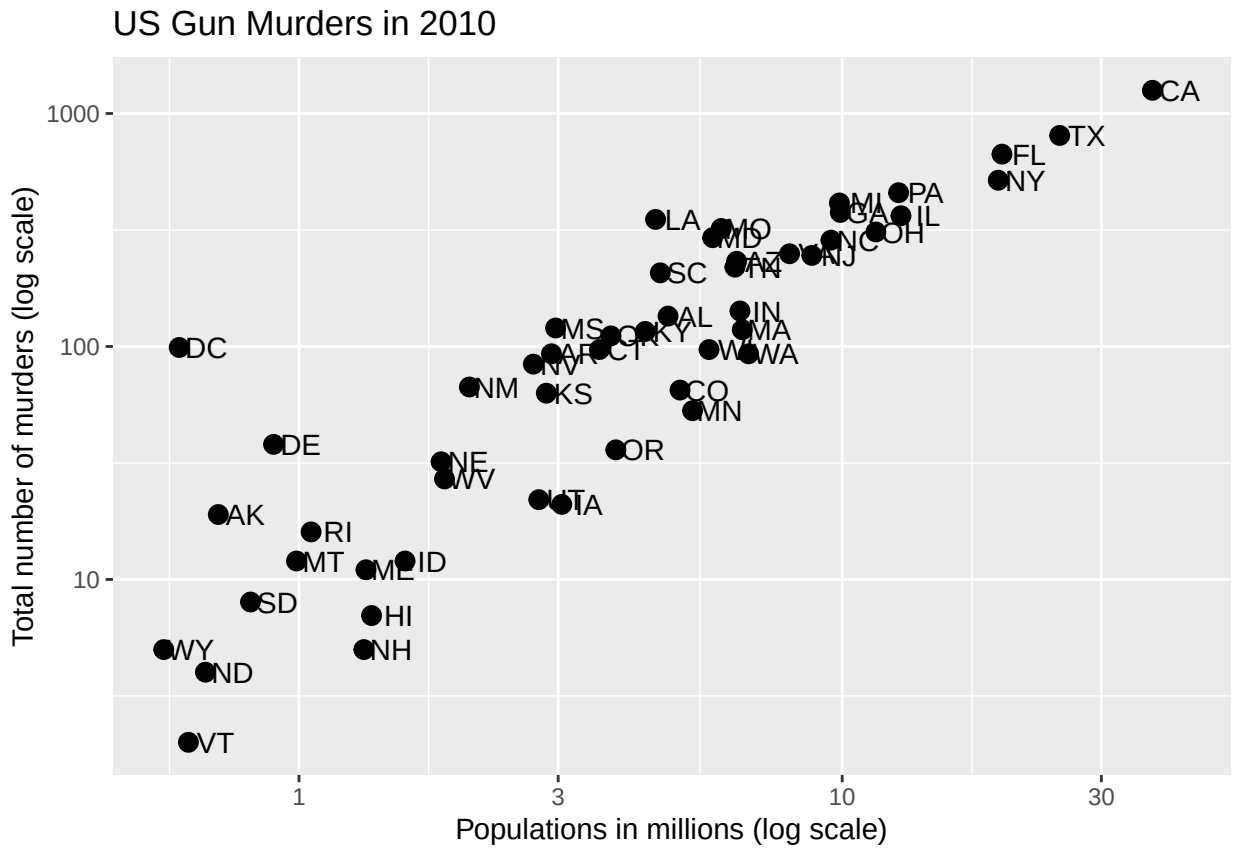

```
p<- murders %>% ggplot(aes(x=population/10^6,y=total,label=abb))
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.1) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Labels and titles

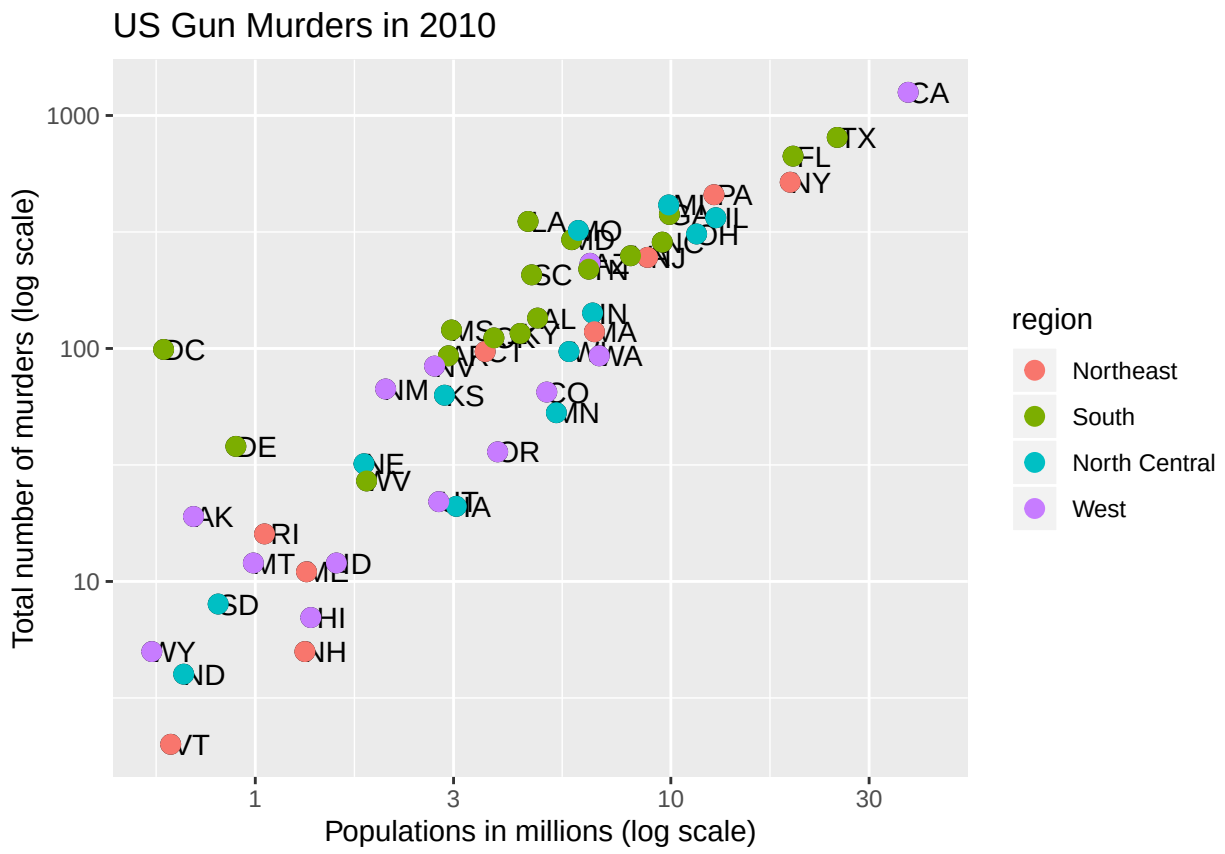
Mirando la cheatsheet vemos que necesitamos los siguientes comandos para poner titulo y labels a los ejes:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```



Colores dinámicos que dependen de una variable

```
p<- murders %>%
  ggplot(aes(x=population/10^6,y=total,label=abb))+
  geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")

p+geom_point(aes(col=region),size=3)
```

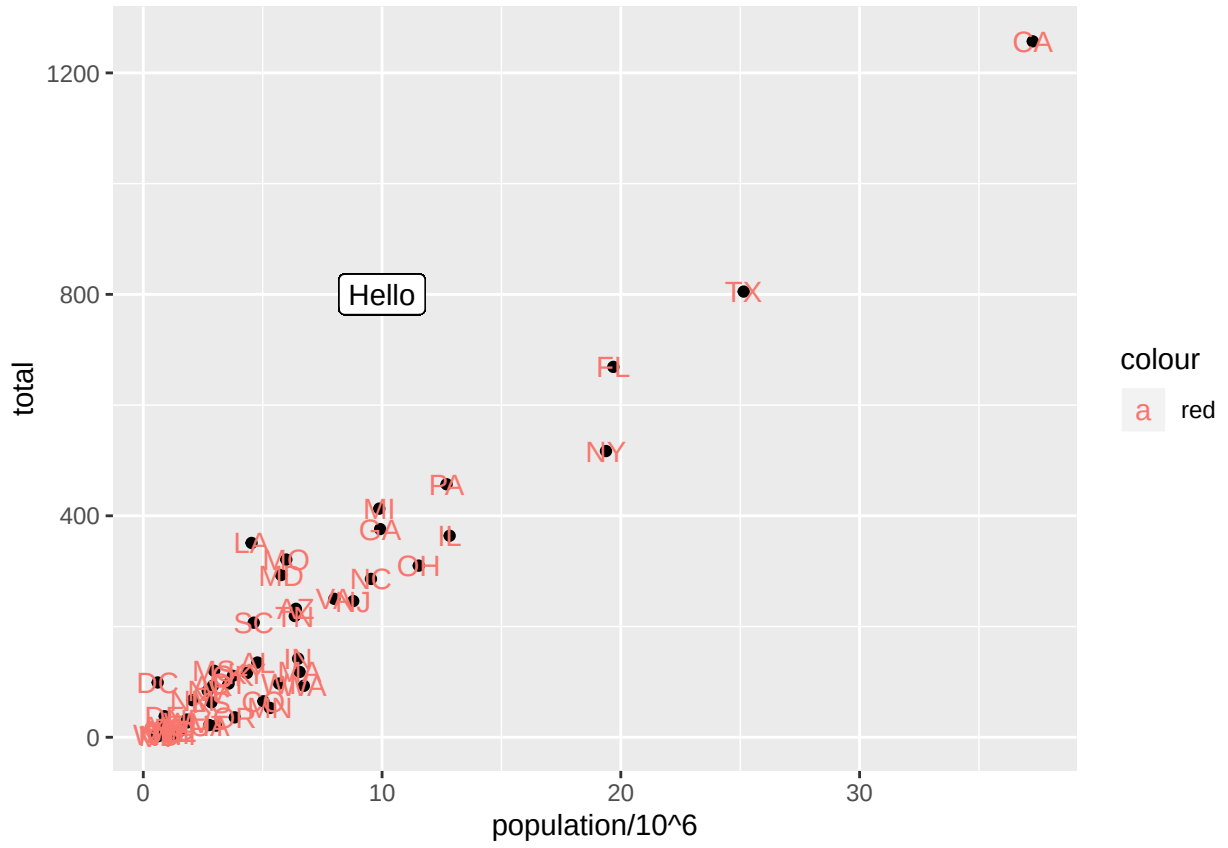


Los valores de x,y (las coordenadas) que precisa esta función geom_point las hereda de lo que hemos ya definido en p. El mapping está en primera posición porque es lo que geom_point espera.

Añadiendo anotaciones

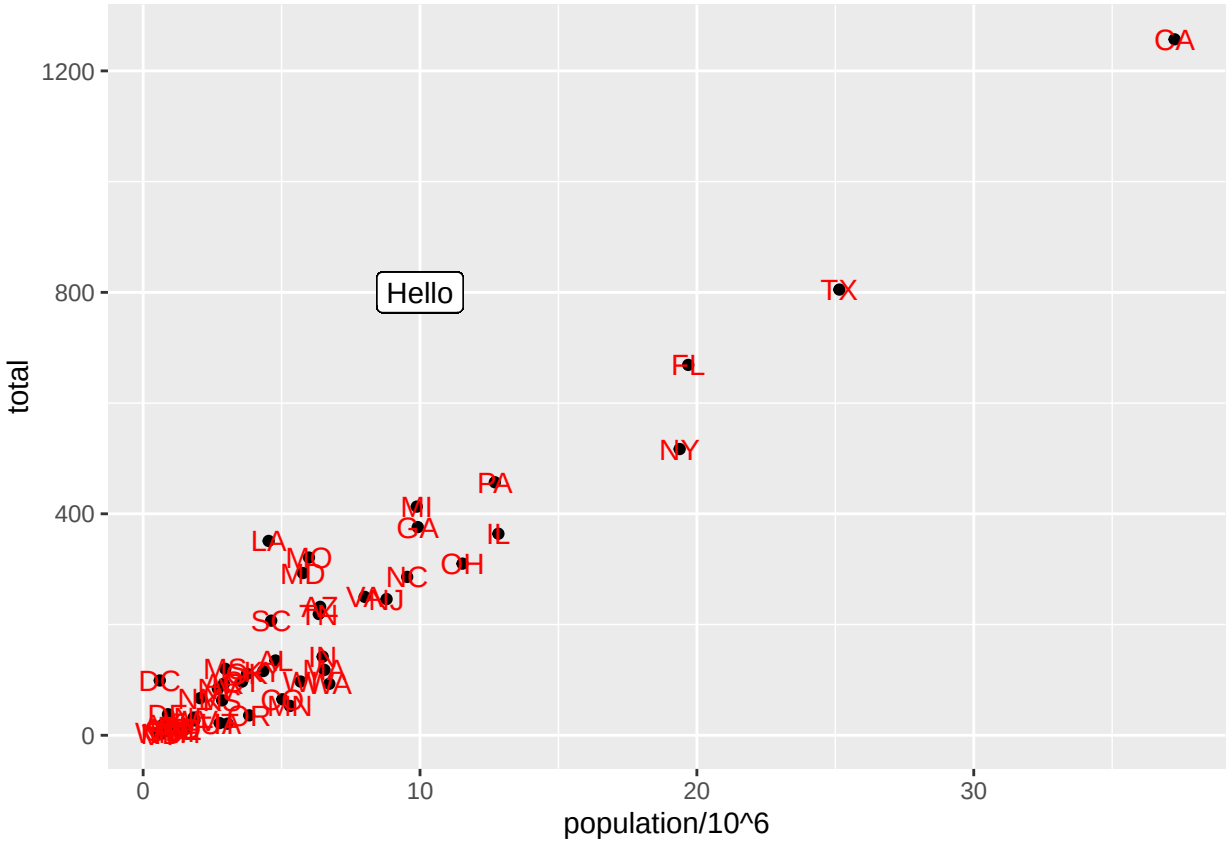
Si queremos añadir algo al plot que no está directamente asociado con el mapeo dato->estética no necesitamos la función *aes()*. Lo hacíamos antes añadiendo un texto en unas coordenadas fijas:

```
murders %>%
  ggplot(aes(x = population/10^6, y = total,label=abb))+
  geom_point()+
  geom_text(aes(col="red"))+
  geom_label(aes(x=10,y=800,label="Hello"))
```



funciona también haciendo:

```
murders %>%
  ggplot(aes(x = population/10^6, y = total,label=abb))+
  geom_point()+
  geom_text(col="red")+
  geom_label(x=10,y=800,label="Hello")
```



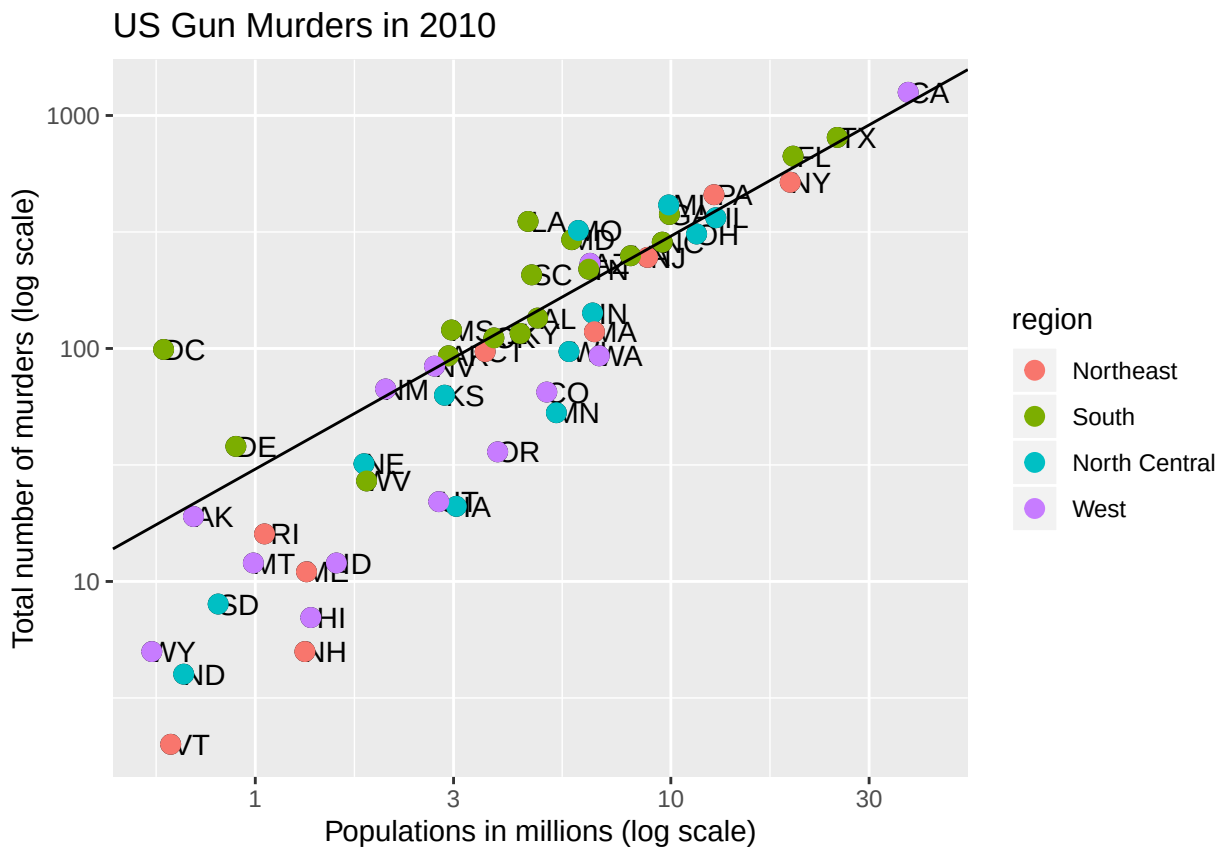
Imaginemos que queremos añadir al plot una linea que tenga como pendiente el rate medio de asesinatos en USA.

Recordemos que usando dplyr podemos conseguir:

```
r <- murders %>%
  summarize(rate = sum(total) /  sum(population) * 10^6) %>% .$rate
```

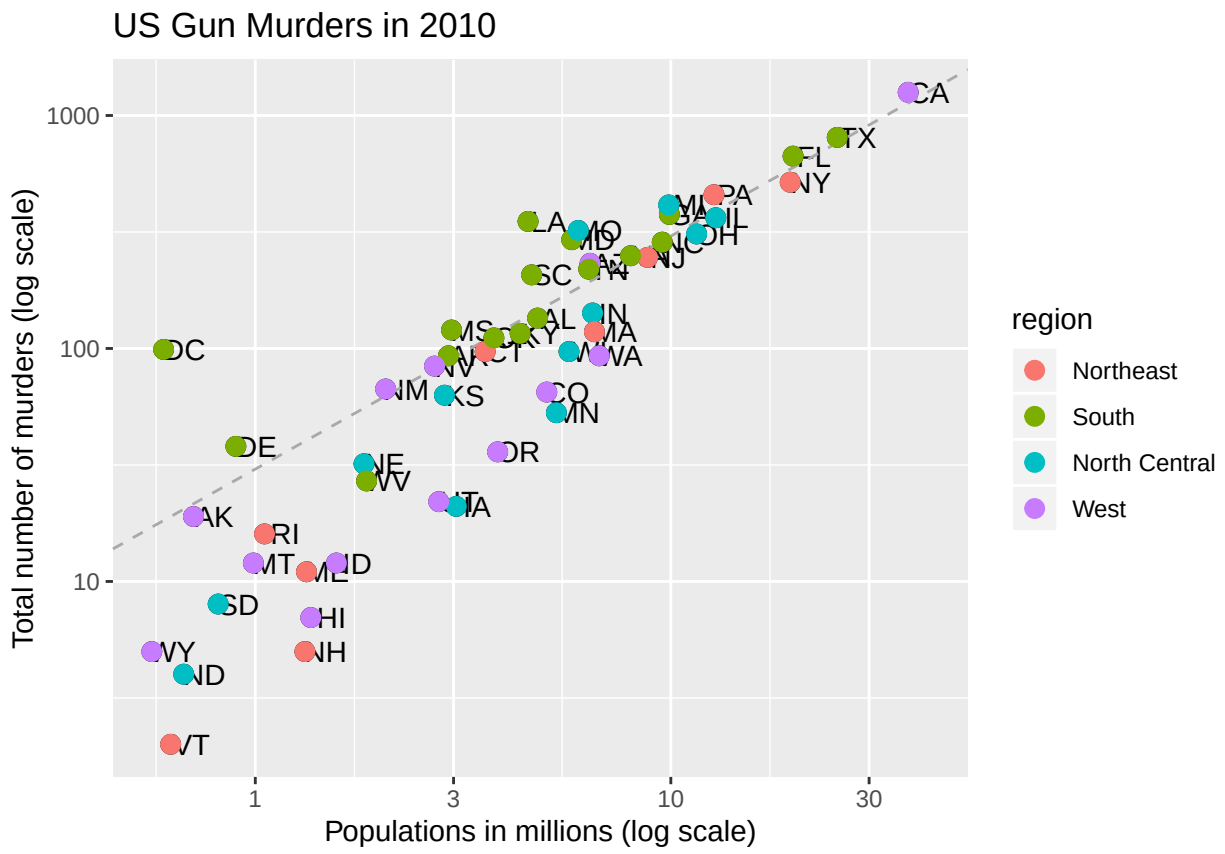
Le añadimos una linea con pendiente 1 e intercepta el log10 de ese ratio medio:

```
p + geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r))
```



Y podemos cambiar los argumentos de esta linea:

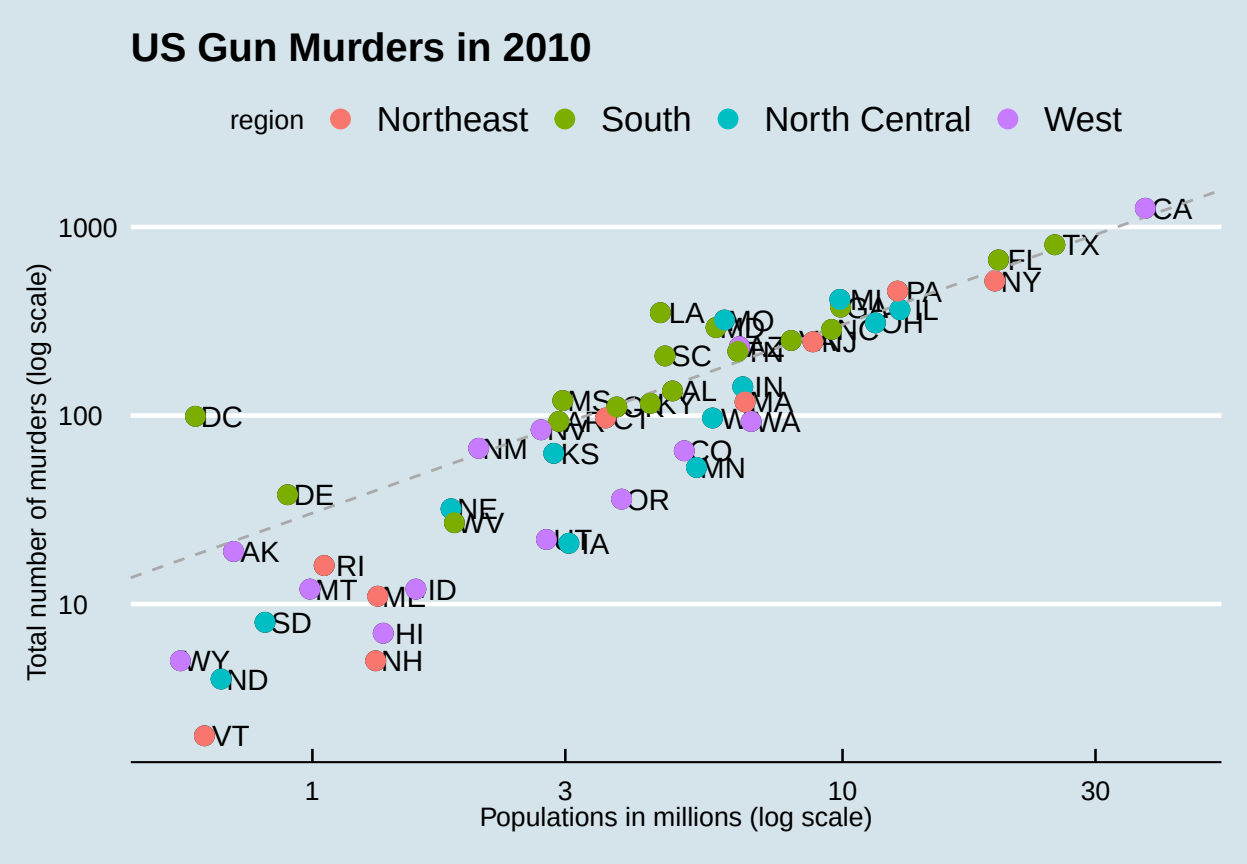
```
p <- p +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3)
p
```



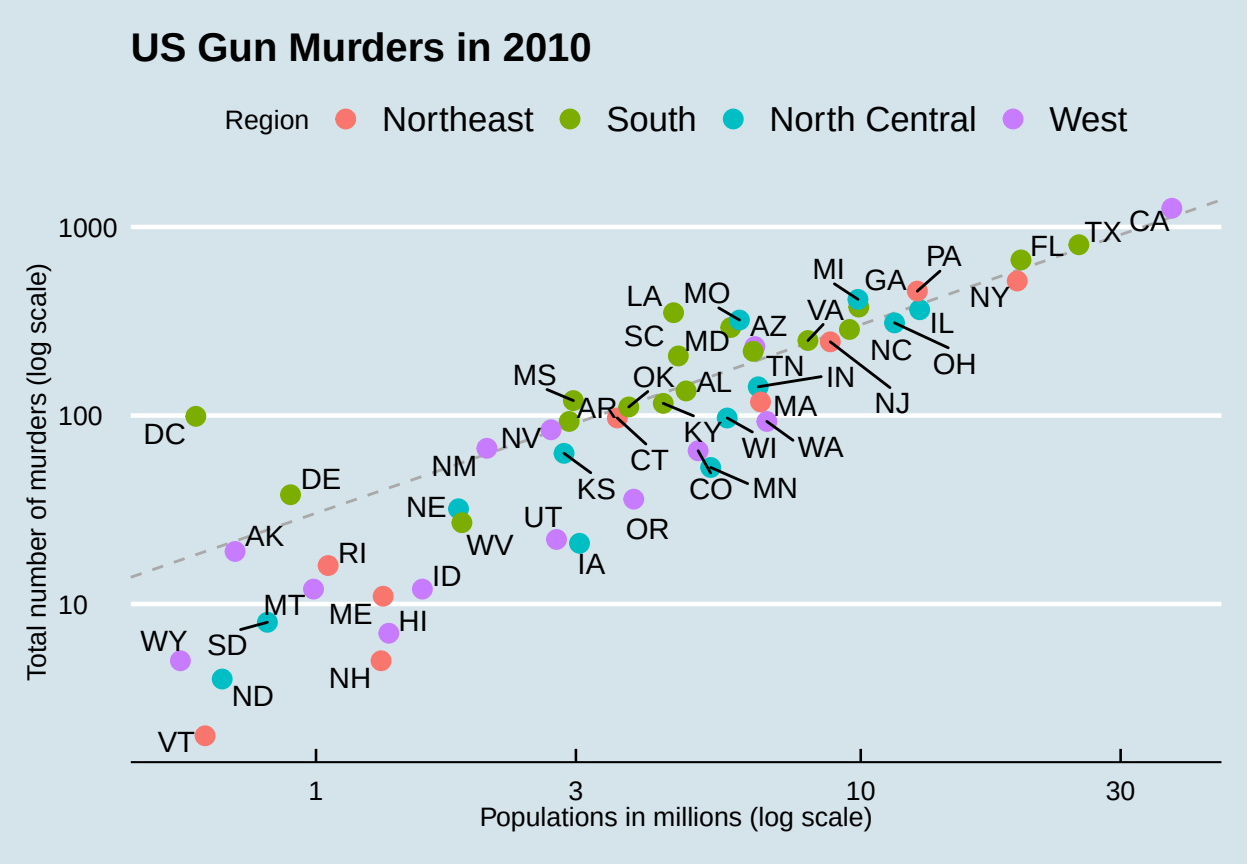
Add-on packages

Otra de las ventajas de ggplot2() es, una vez mas, la existencia de muchos paquetes que nos proporcionan estas features ya implementadas. Por ejemplo, con el paquete ggtheme() podemos cambiar el background y el estilo de nuestro plot por otros ya implementados. O con ggrepel() podemos distanciar los puntos de manera que no caigan unos encima de otros.

```
library(ggrepel)
library(ggthemes)

p + theme_economist()
```




```
library(ggthemes)
library(ggrepel)

### First define the slope of the line
r <- murders %>%
  summarize(rate = sum(total) /  sum(population) * 10^6) %>% .$rate

## Now make the plot
murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```



Varios plots en la misma ventana

```
p <- heights %>% filter(sex=="Male") %>% ggplot(aes(x = height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col="black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col="black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col="black")
```

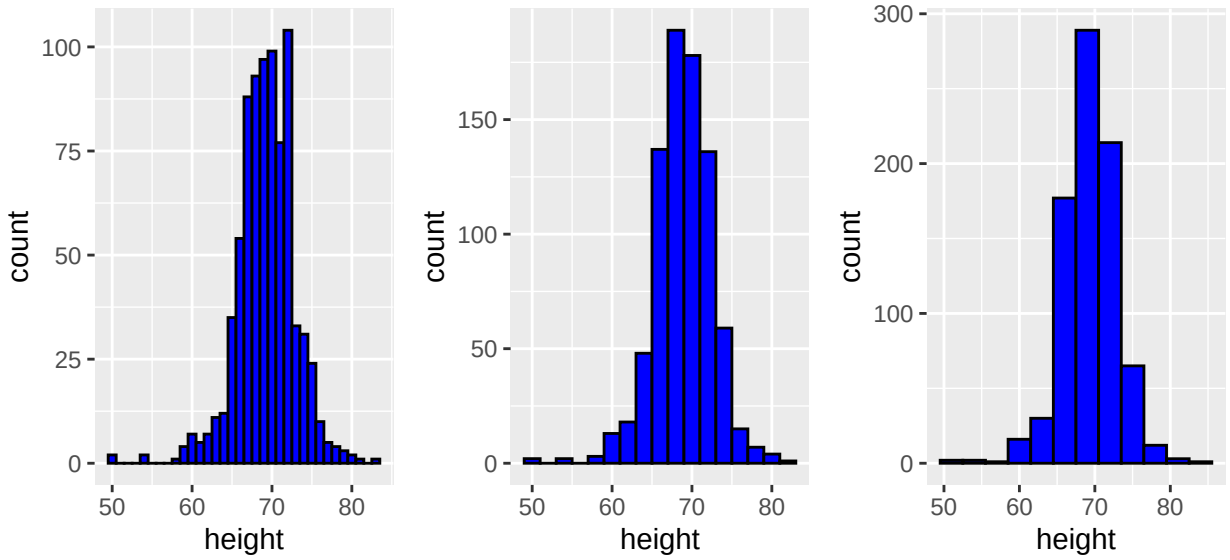
Podemos utilizar la función `grid.arrange` in the **gridExtra** package:

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(p1,p2,p3, ncol = 3)
```


count
height
100
75
50
25
0
50
60
70
80
count
height
150
100
50
0
50
60
70
80
count
height
300
200
100
0
50
60
70
80