

Visualización de datos con R (II)

FSC

February 2, 2019

Contents

La importancia de la visualización de datos	1
R for data science: tidyverse	1
Visualizando datos con R: <i>ggplot2()</i>	2
Ejercicio 1:	2
Creando un plot con <i>ggplot2()</i>	3
Geometría	7
Aesthetics	8
Global vs local aesthetics	8
Capas	10
Labels and titles	17
Colores dinámicos que dependen de una variable	18
Añadiendo anotaciones	19
Varios plots en la misma ventana	25

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*](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*](https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen)

R for data science: tidyverse

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.2
```

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

```
## v ggplot2 3.1.0      v purrr   0.2.5
```

```
## v tibble  1.4.2      v dplyr   0.7.8
```

```
## v tidyr   0.8.2      v stringr 1.3.1
```

```
## v readr   1.1.1      v forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
## -- Conflicts ----- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()     masks stats::lag()
```

```
library(ggplot2)
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 3.5.2
```

```
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.5.2
```

```
##
```

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

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

```
##
```

```
##      combine
```

Visualizando datos con R: *ggplot2()*

Hay otras librerías para visualizar datos en R, como por ejemplo las funciones de la instalación 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 sí, 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 tuviéramos una serie temporal cada combinación (estado, año) sería la “key” de la tabla y aparecería también 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 geometría y el aesthetic mapping.

```
##
```

```
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
```

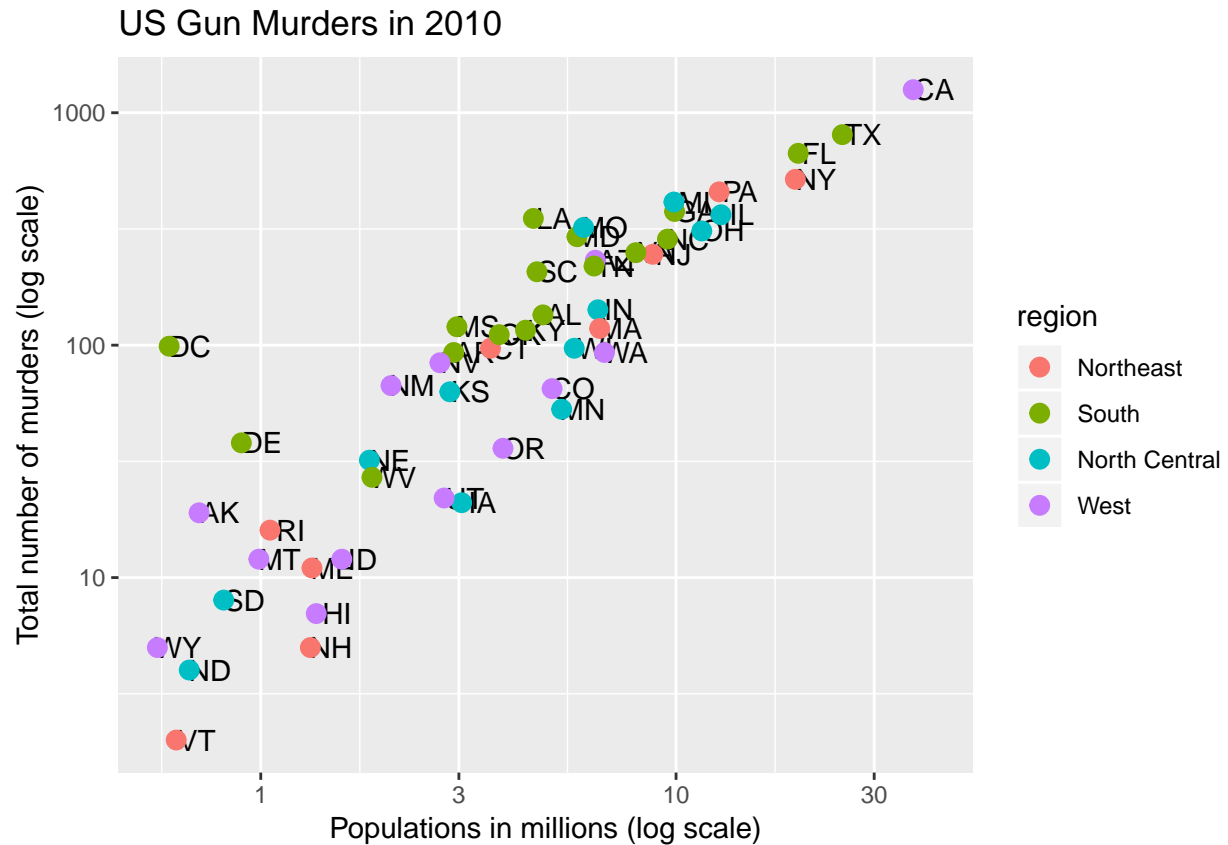
```
##
```

```
##      between, first, last
```

```
## The following object is masked from 'package:purrr':
```

```
##
```

```
##      transpose
```

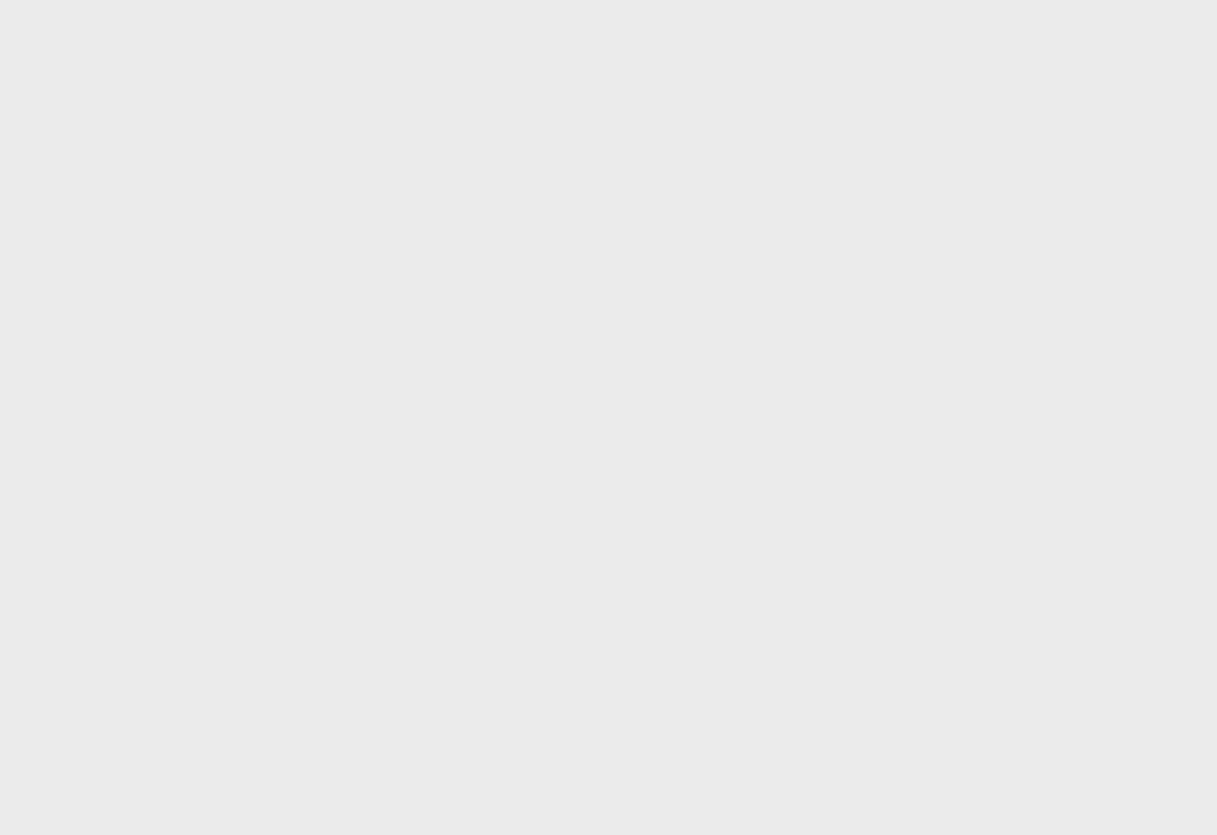


- Data: murders
- Geometria: Scatterplot
- Aesthetics Mapping:
 - ++ x-axis: population size
 - ++ y-axis: total number of murders
 - ++ text: states
 - ++ colors: the four different regions

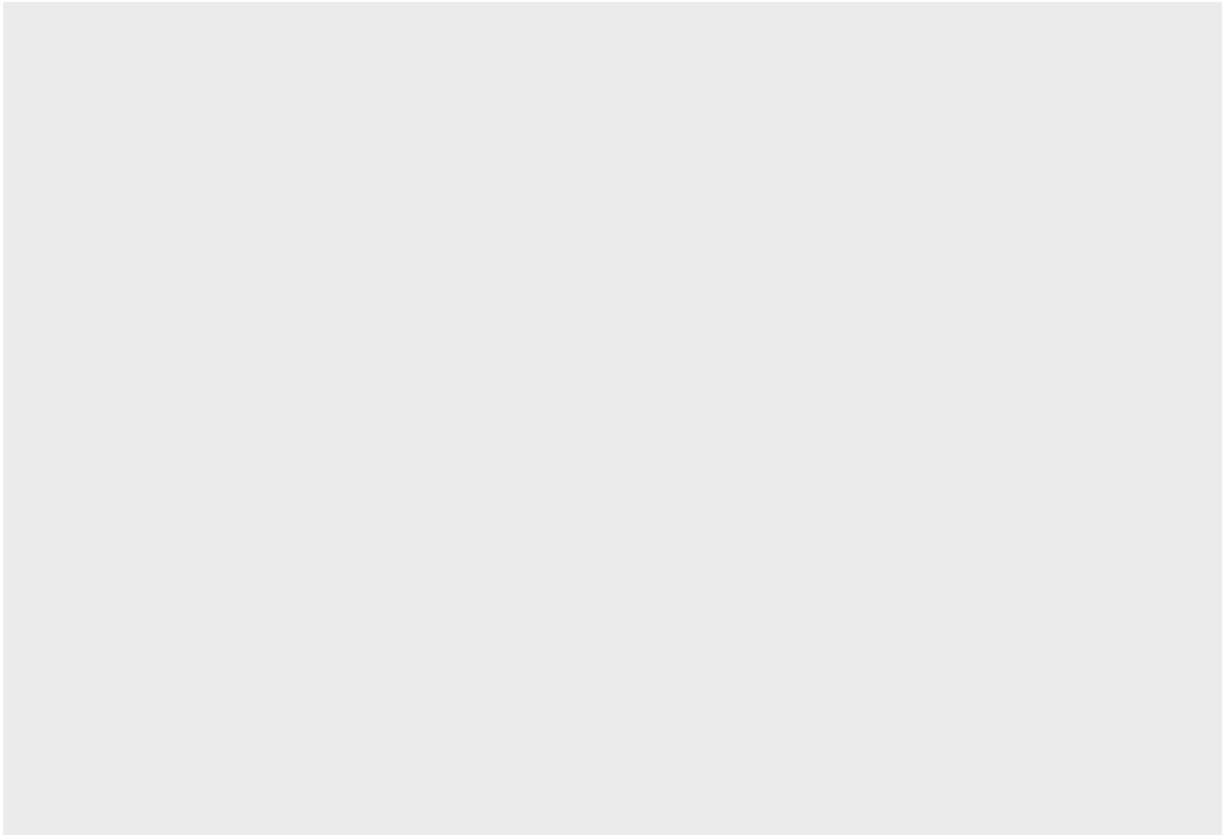
Creando un plot con *ggplot2()*

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

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



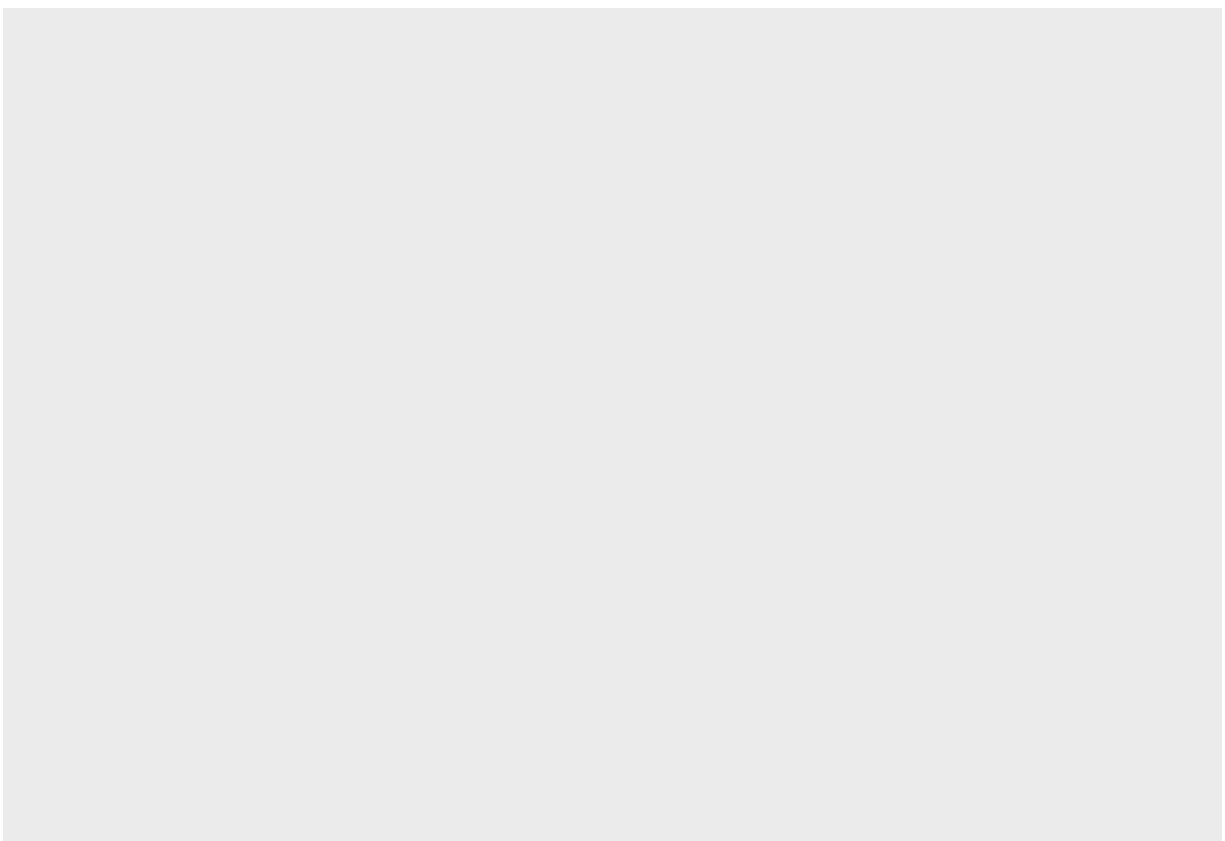
```
#alternativamente con pipe:  
murders %>% ggplot()
```



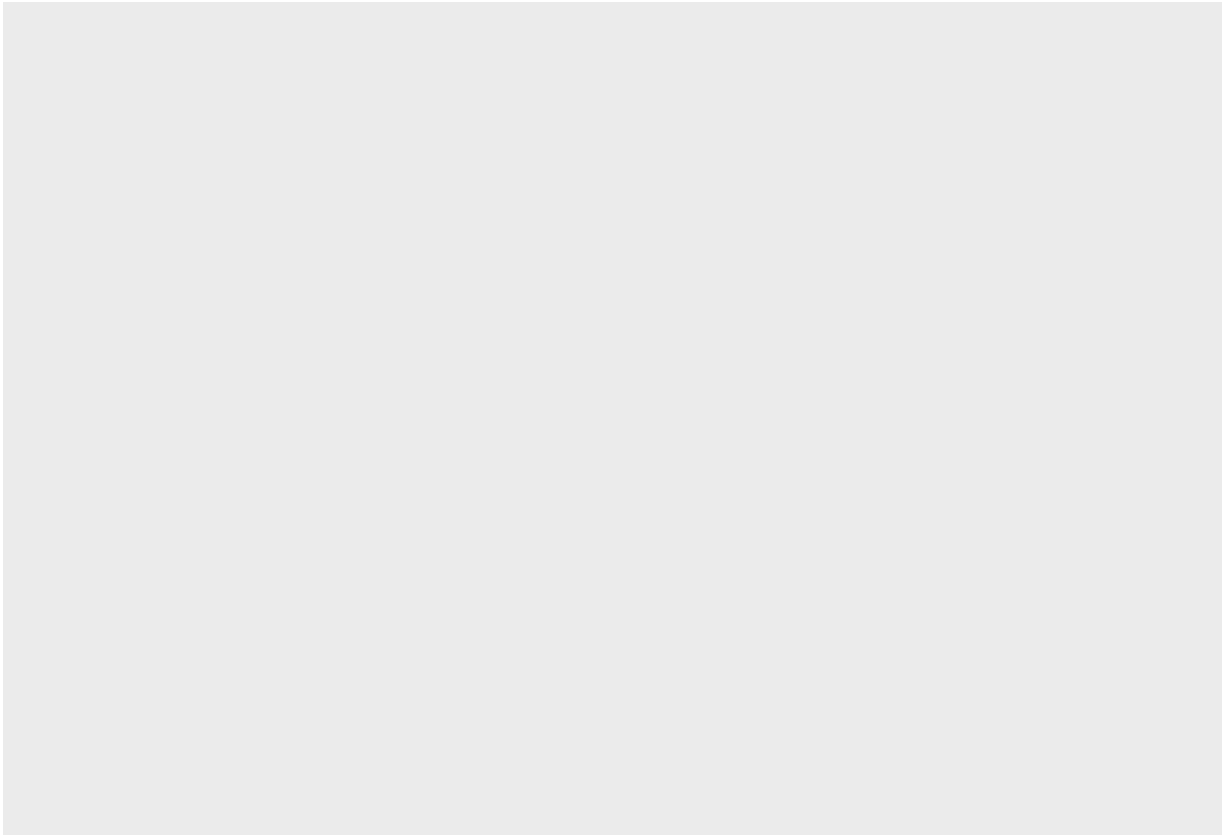
```
p <- ggplot(data = murders)
class(p)
```

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

```
print(p)
```



p



Geometria

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

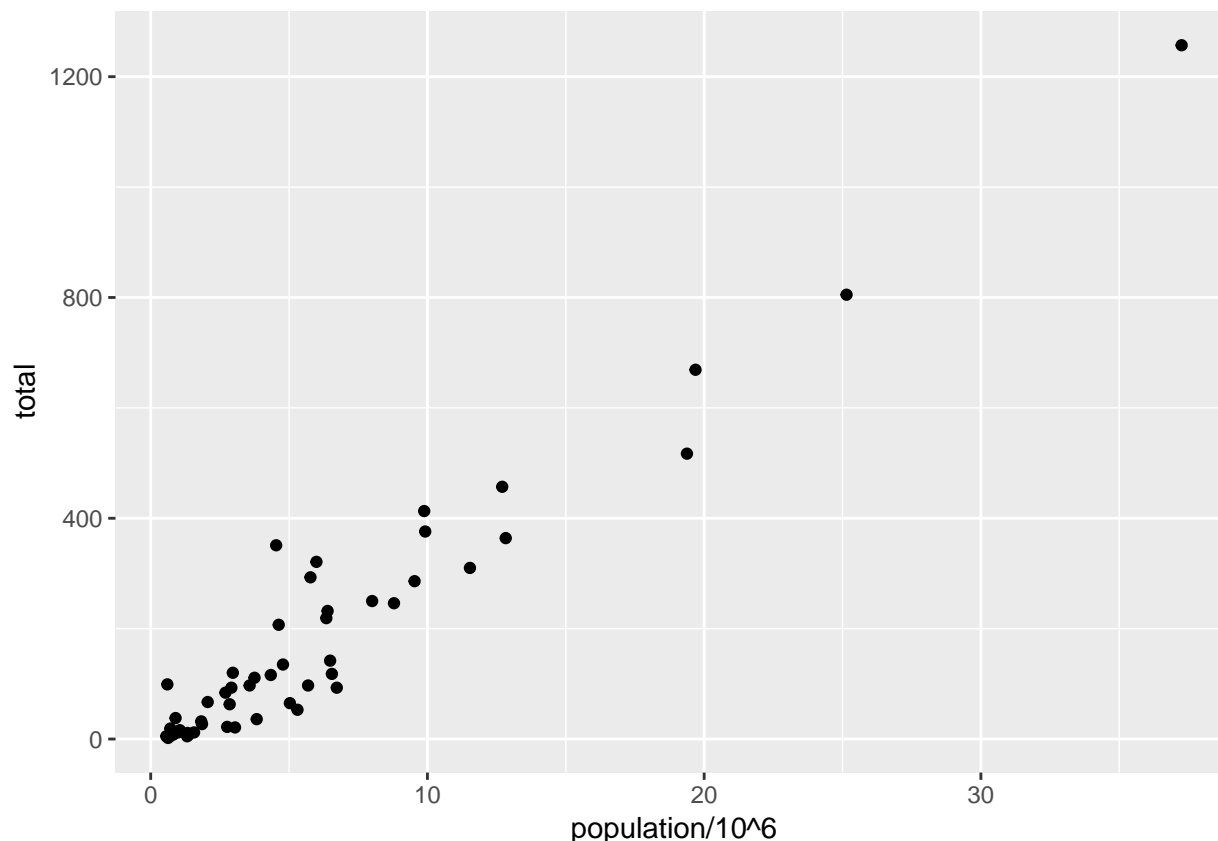
Cada geometría necesita una serie de parámetros fijos para poder pintar y otros opcionales

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
```

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

```
murders %>% ggplot() +  
  geom_point(aes(x = population/106, 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 nuestros 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) características 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 qué valor pone en cada pixel de la pantalla y cuáles son sus características. Esto es lo que tratamos de hacer con `aes()`: mapear valores en características de un gráfico

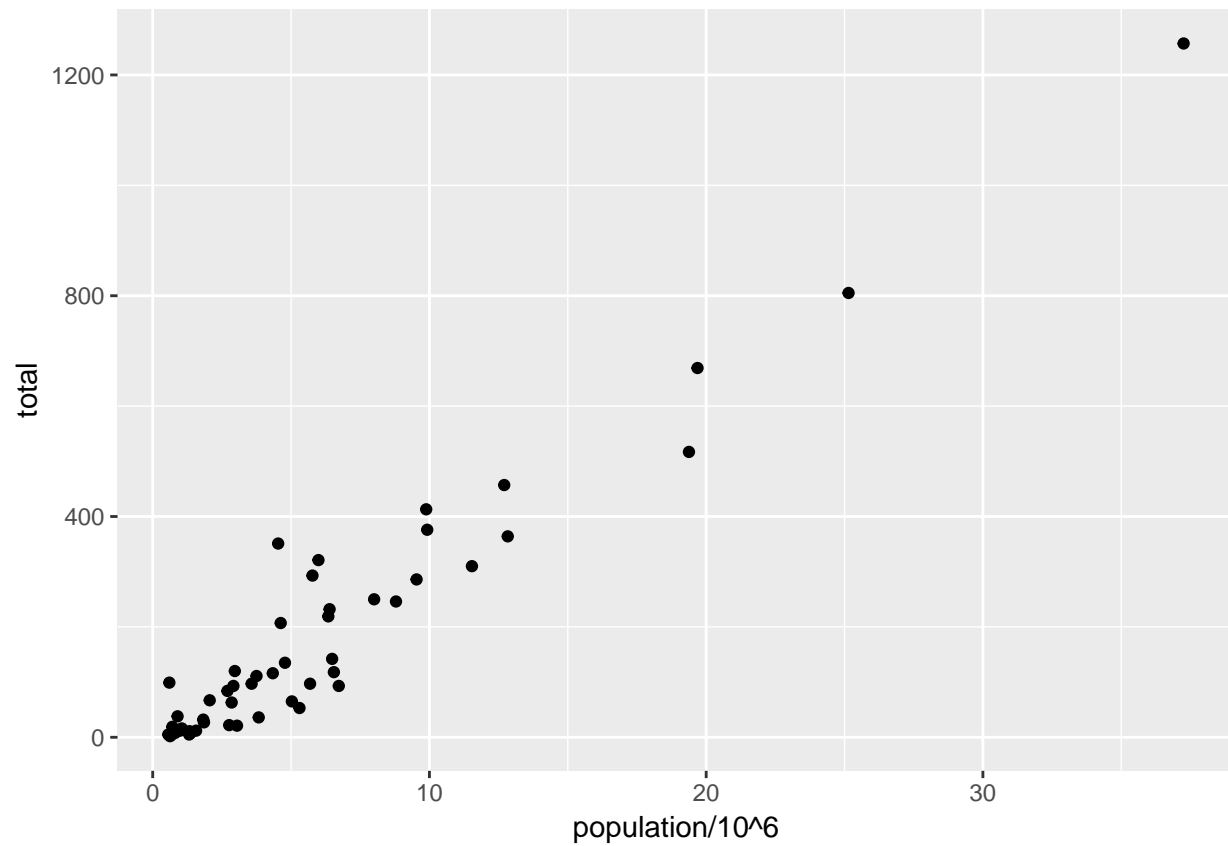
La función `aes()` es la que le indica a la geometría que necesita pintar en el plot y cómo. Además se le pueden dar muchas propiedades como color, tamaño, etc, en función de los datos. Cada geometría requiere un cierto tipo de mapeo datos/visualización.

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 heredaría estas características globales.

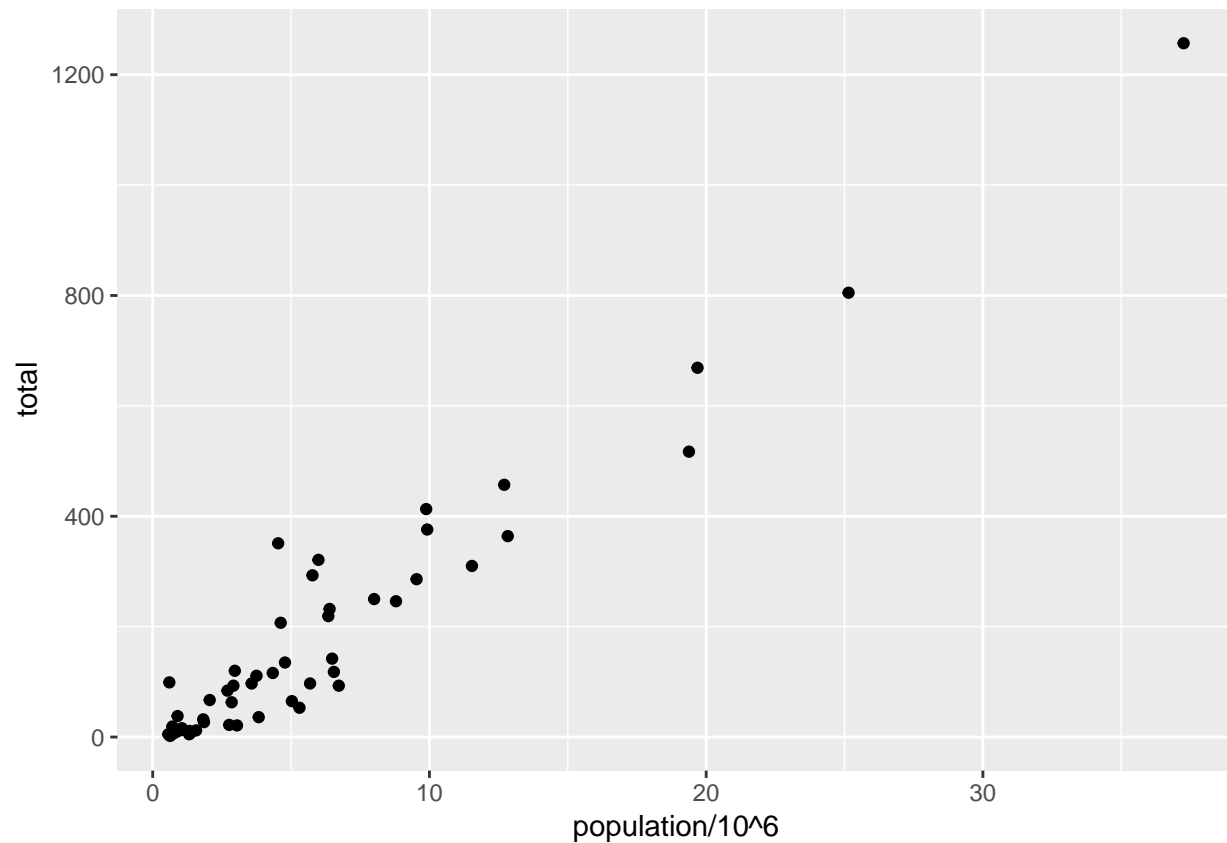
Si le hubieramos pasado las estéticas al plot anterior no haría falta dárselas a `geom_point` utilizaría las del objeto `ggplot2`


```
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))
```



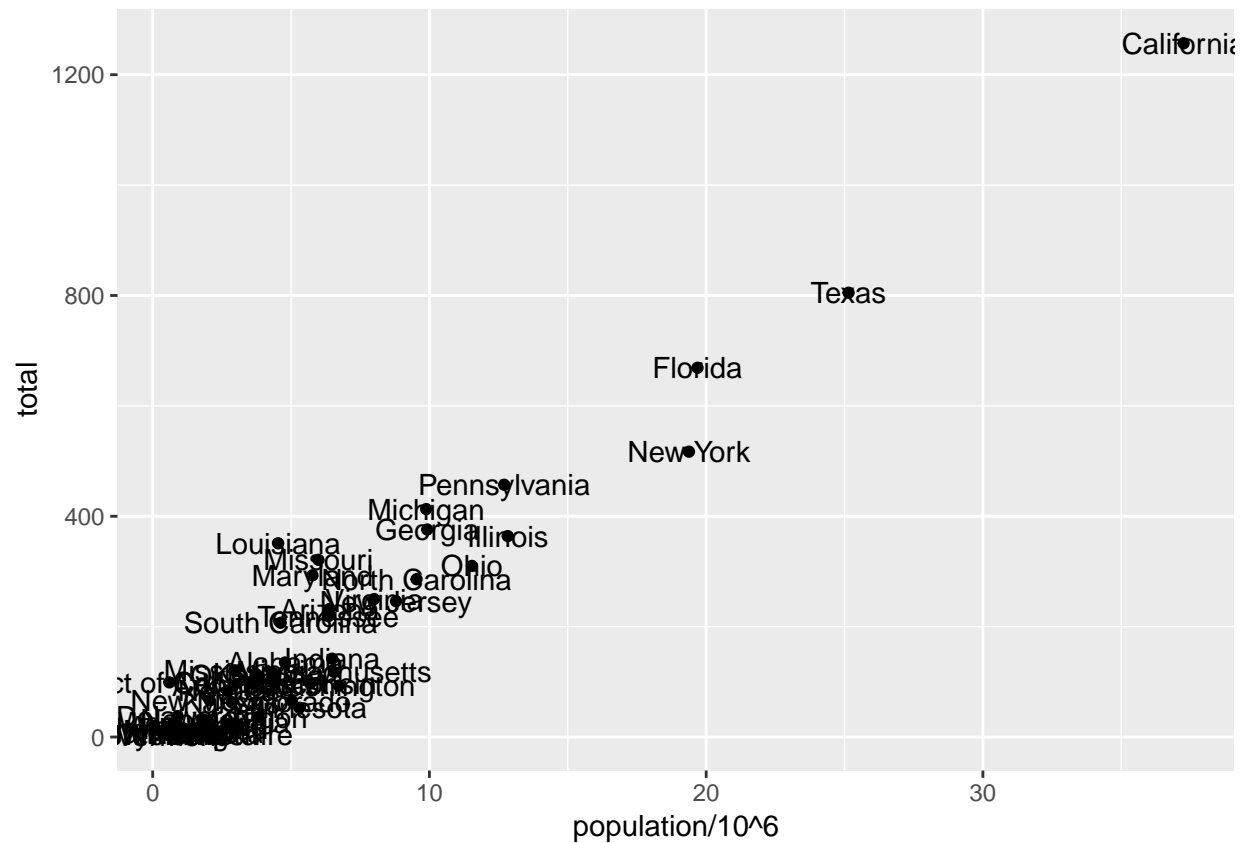
Capas

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 geometría. Después podemos seguir añadiendo características.

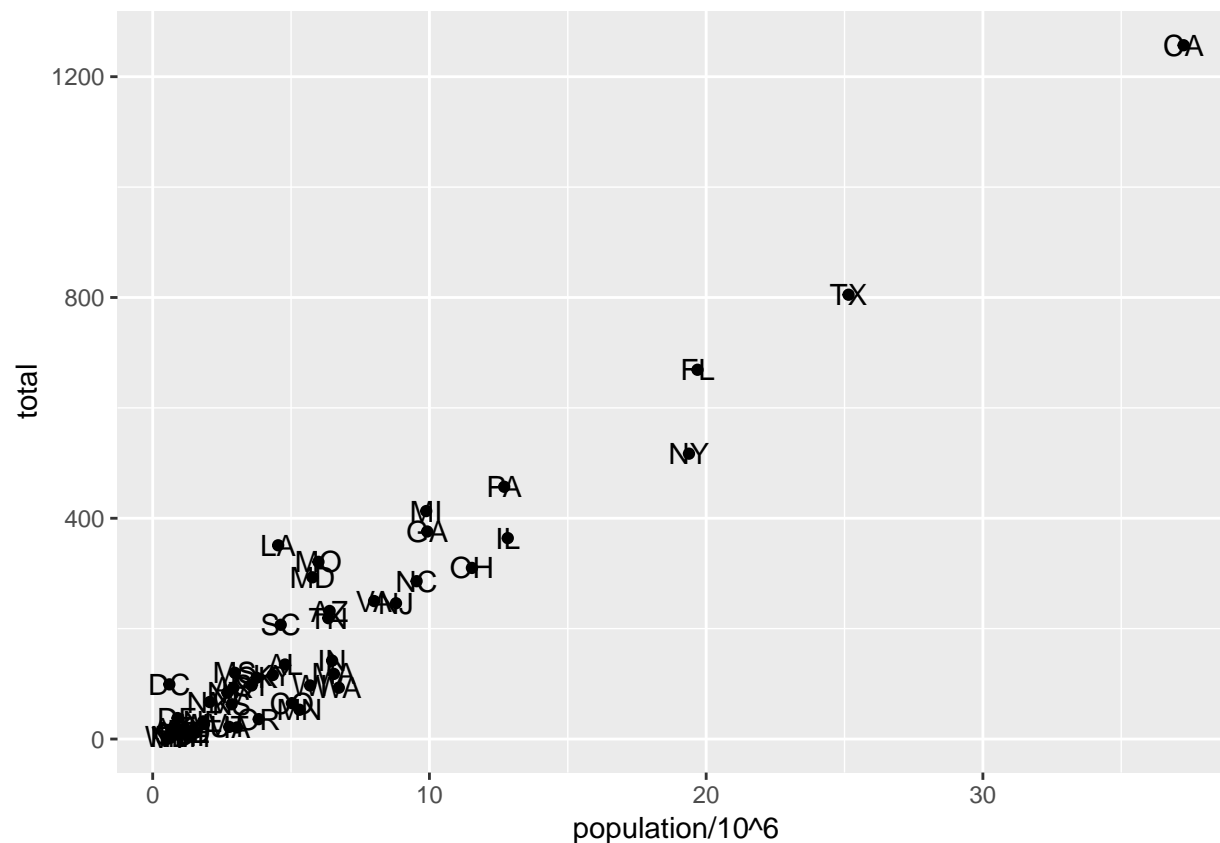
Otra capa que queremos añadir es texto para cada punto. Se utiliza la función *geom_text()*

```
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))
```



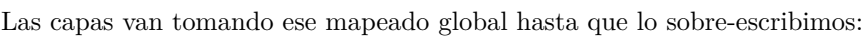
Fuera de `aes()` no se reconocen las variables de los objetos para las que queremos mapear características estéticas

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

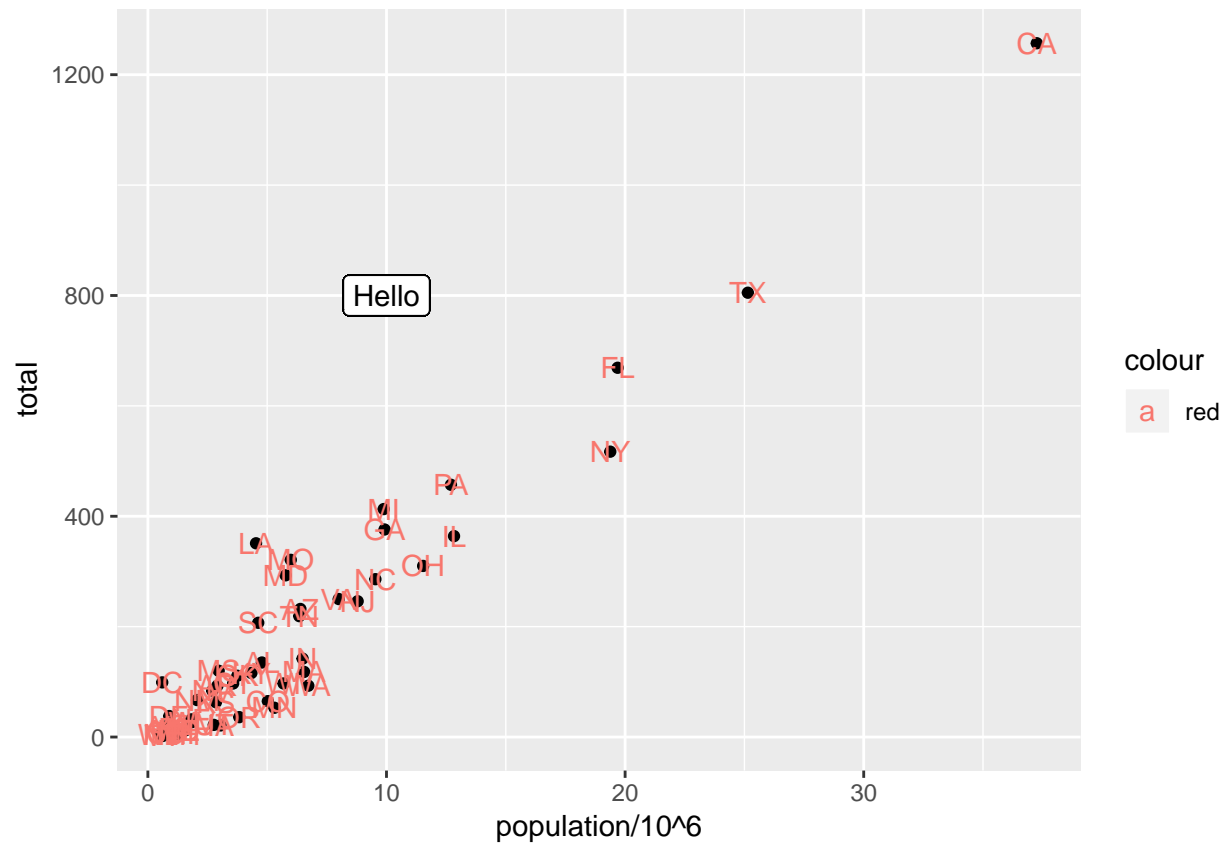
```
## Error in layer(data = data, mapping = mapping, stat = stat, geom = GeomText, : object 'abb' not found
```

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")
```

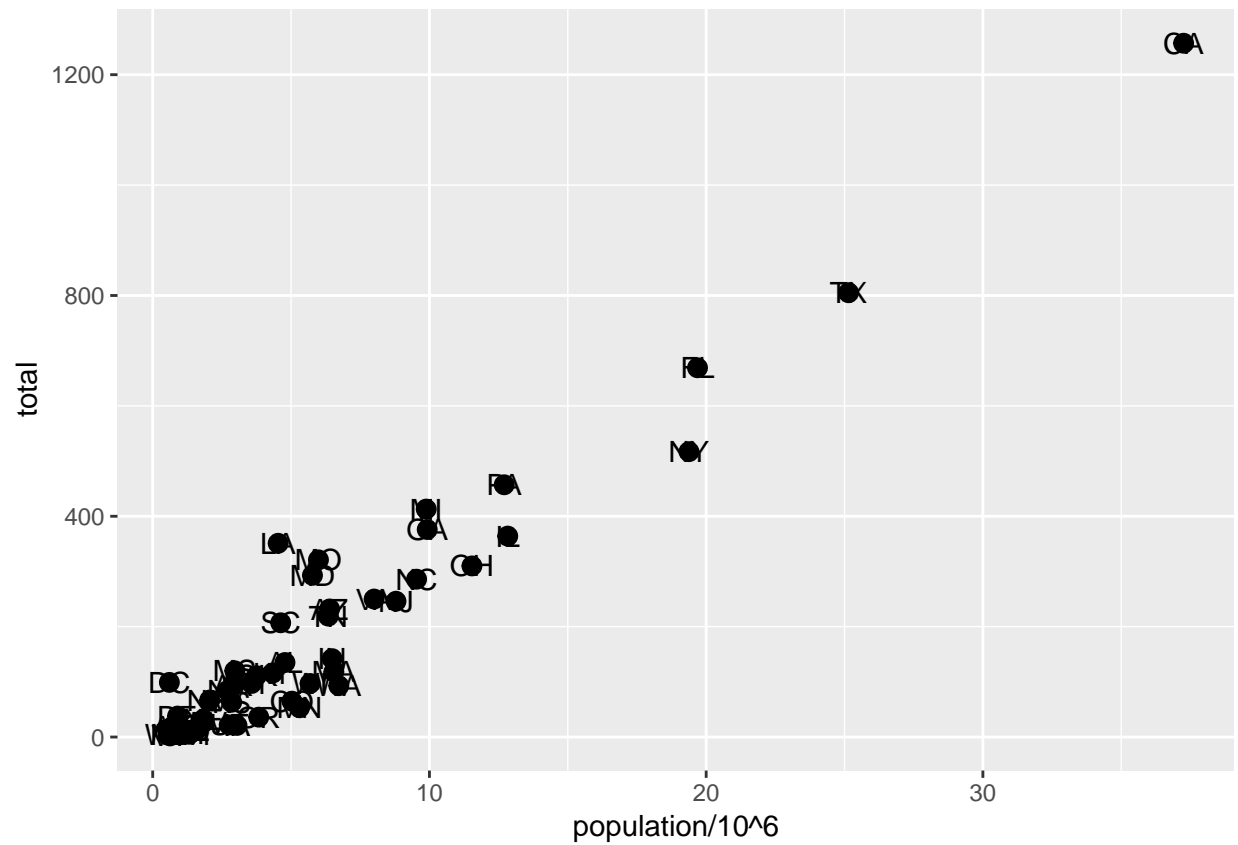


13

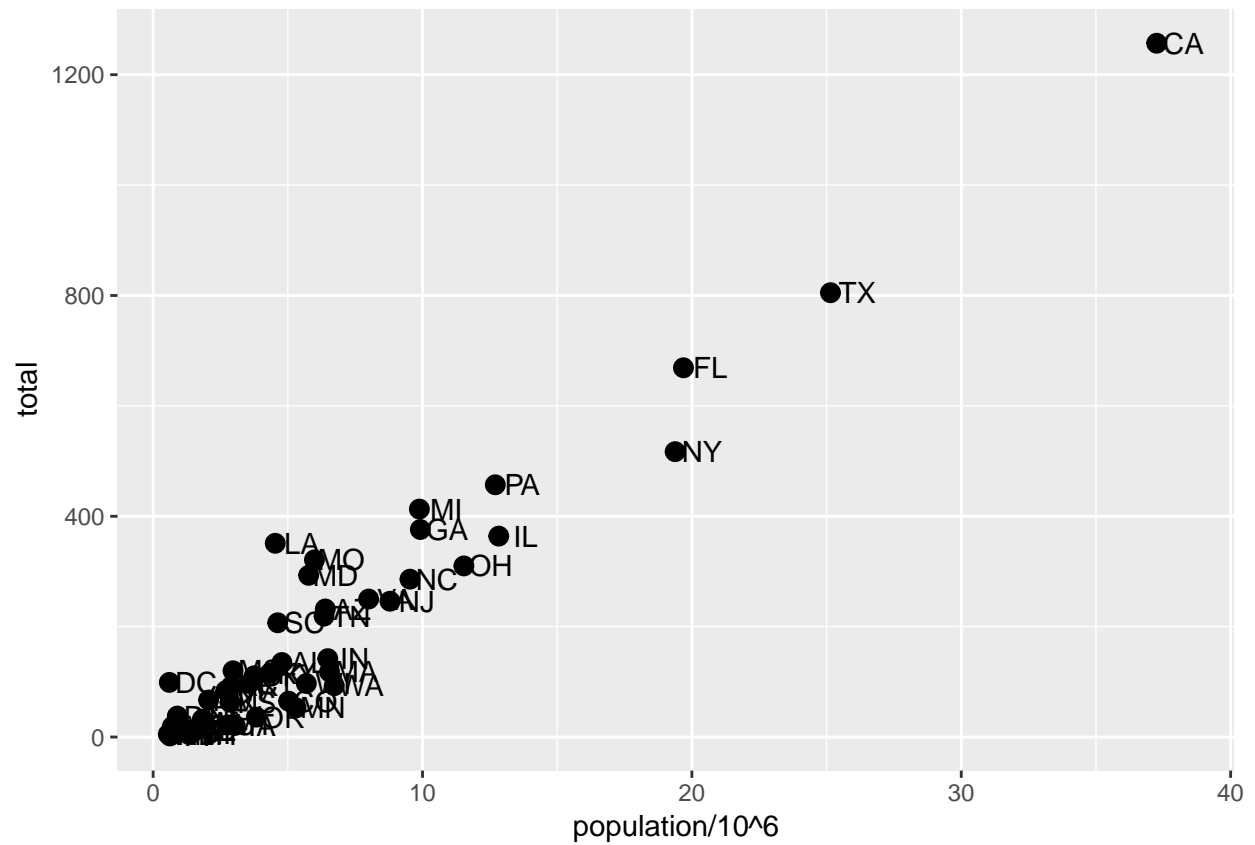


Cada función requiere diferentes parámetros unos que son *aesthetics* y otros fijos. La diferencia es que *aesthetics* mapea datos a propiedades estéticas (para uno o para todos los datos) mientras que lo que está fuera de *aes()* afecta a todo el plot, no hay un mapeo dato->estética.

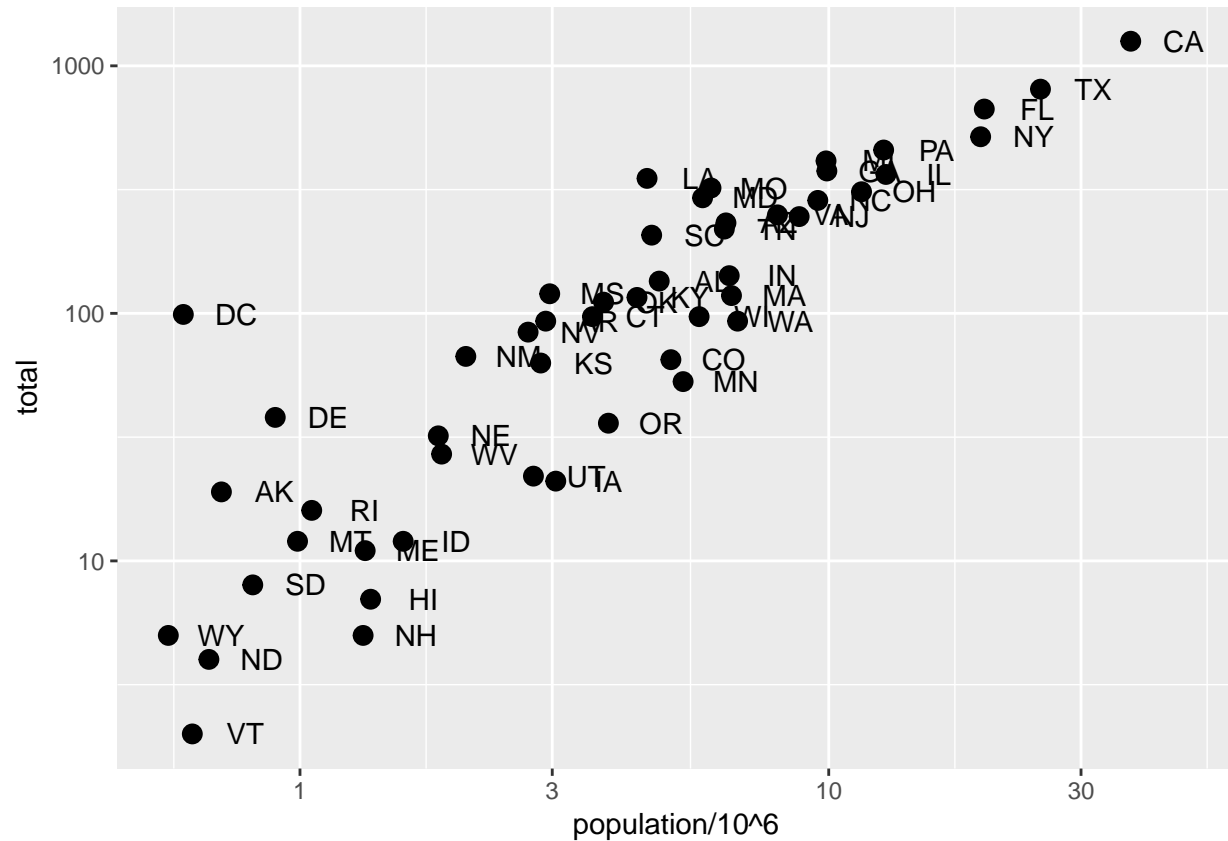
```
p<-murders %>% ggplot()
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```



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

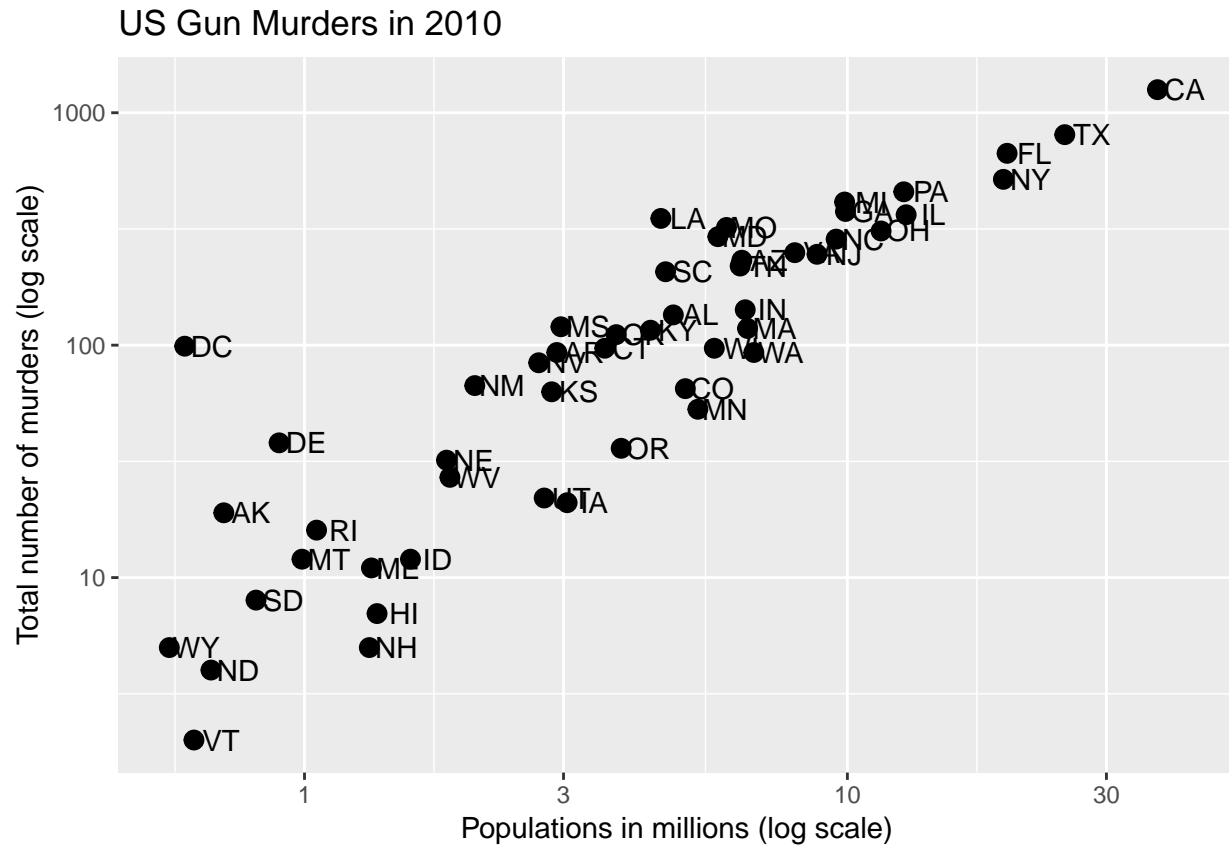


```
p<- murders %>% ggplot(aes(x=population/10^6,y=total,label=abb))
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.1) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```

Labels and titles

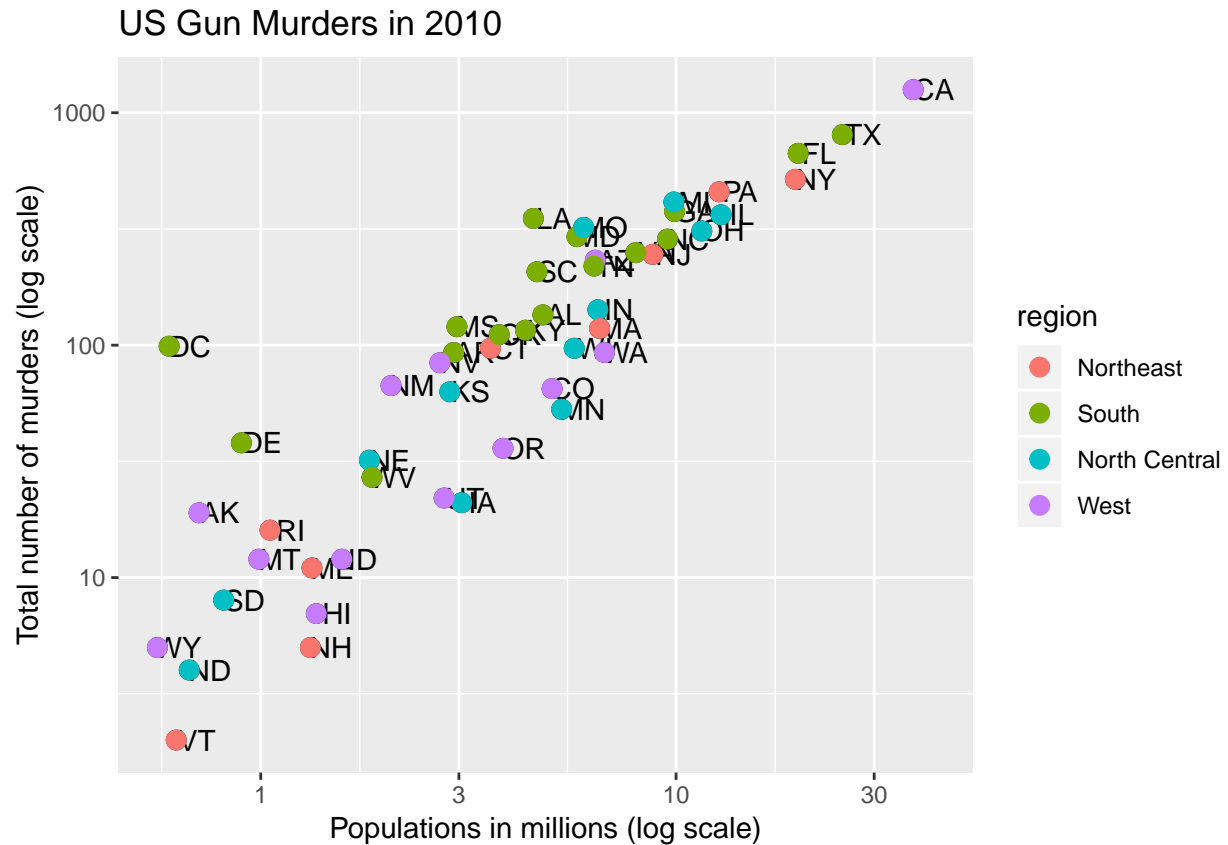
```
p + geom_point(size = 3) +  
  geom_text(nudge_x = 0.05) +  
  scale_x_log10() +  
  scale_y_log10() +  
  xlab("Populations in millions (log scale)") +  
  ylab("Total number of murders (log scale)") +  
  ggtitle("US Gun Murders in 2010")
```



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)
```

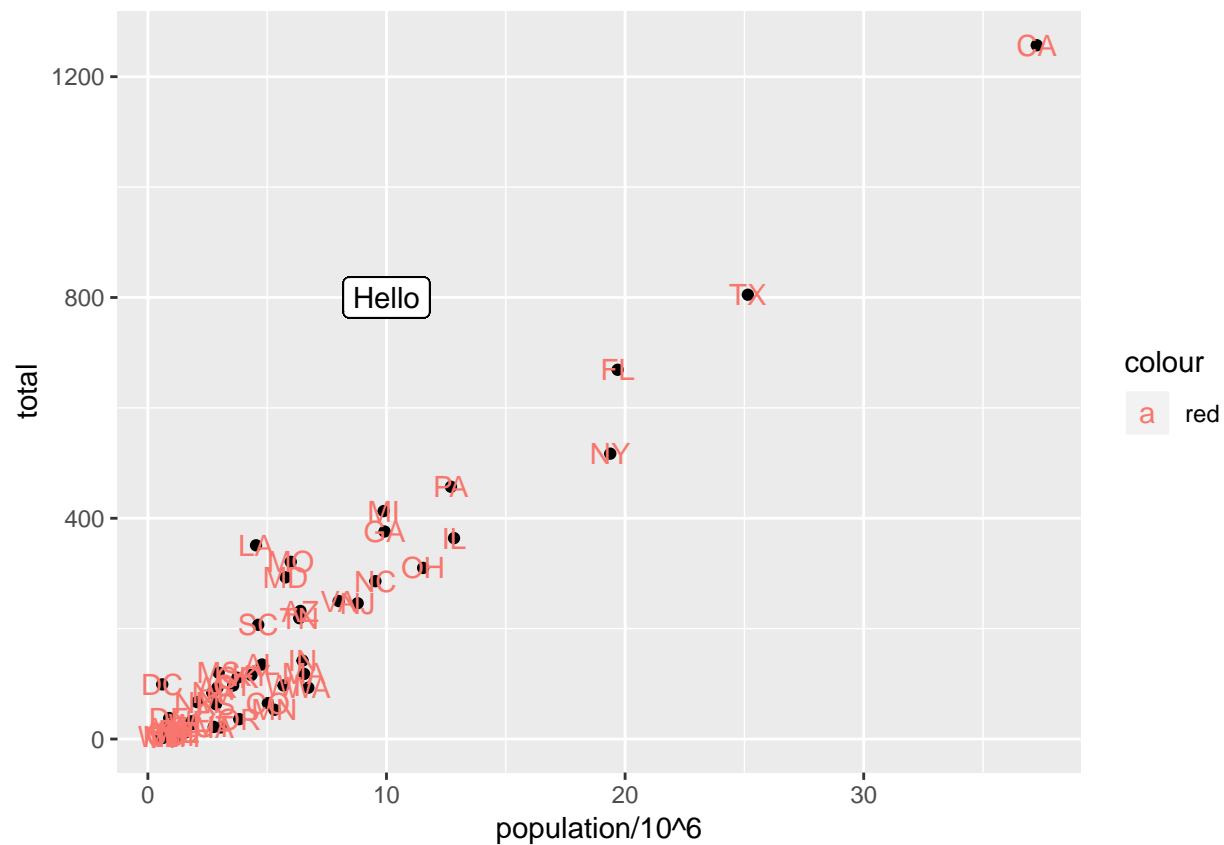


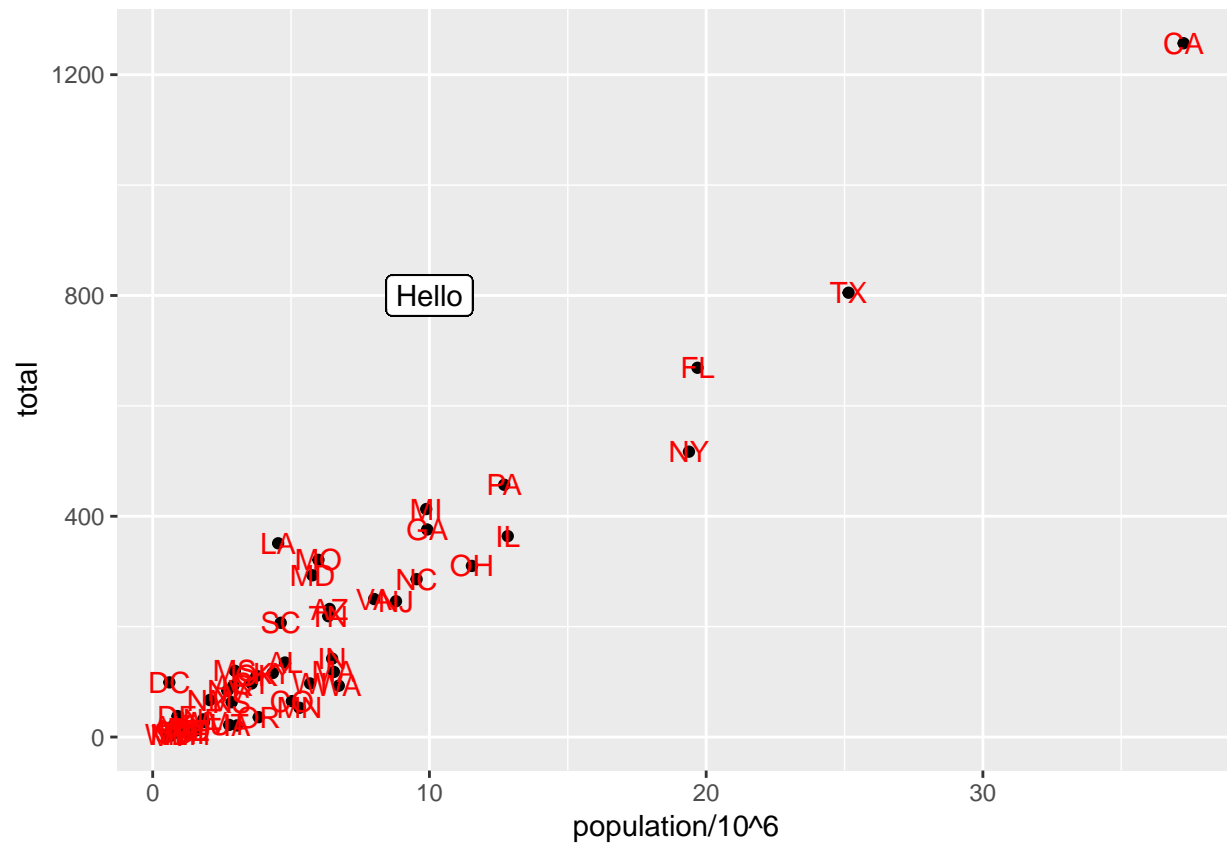
Los valores de x,y (las coordenadas) que precisa esta función `geom_point` las hereda de lo que hemos ya definido en `p`. El mapping está en primera posición porque es lo que `geom_point` espera.

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"))
```





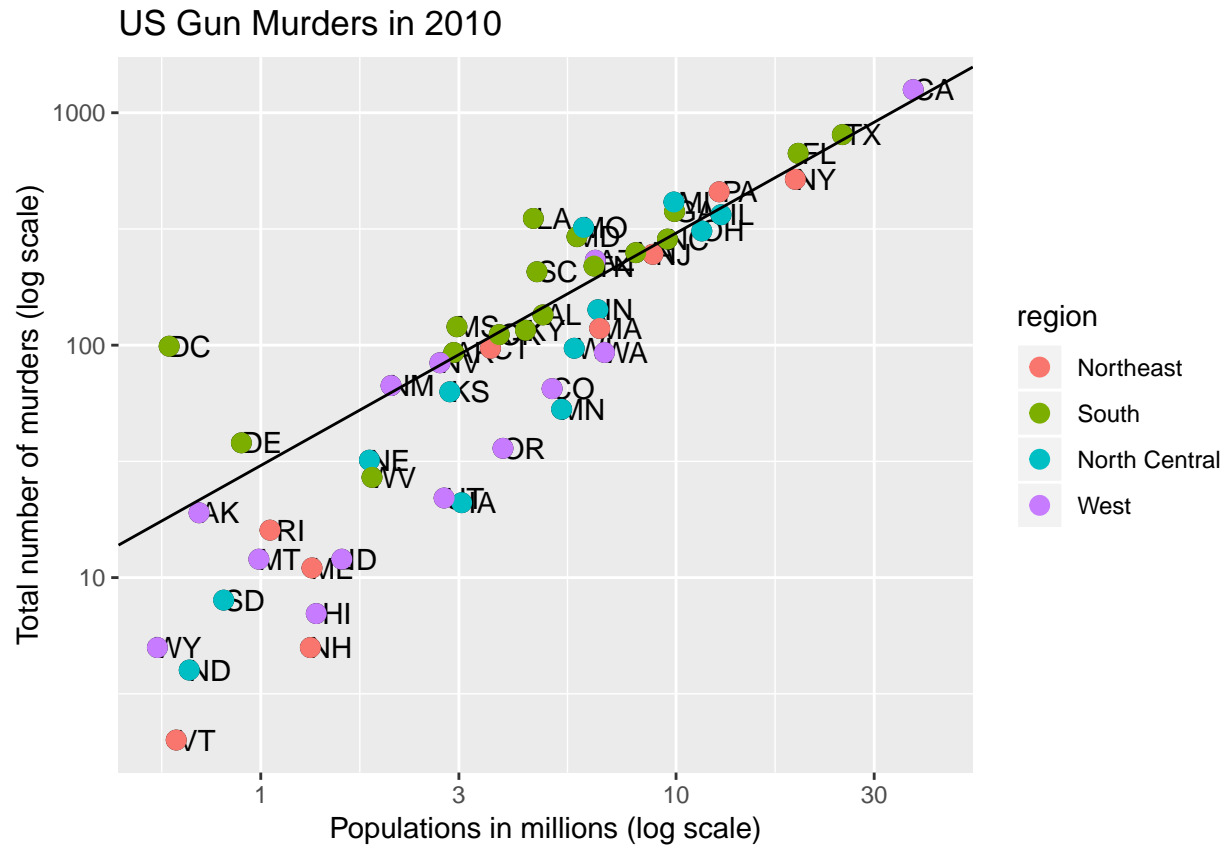
Imaginemos que queremos añadir al plot una línea que tenga como pendiente el rate medio de asesinatos en USA.

Recordemos que usando dplyr podemos conseguir:

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

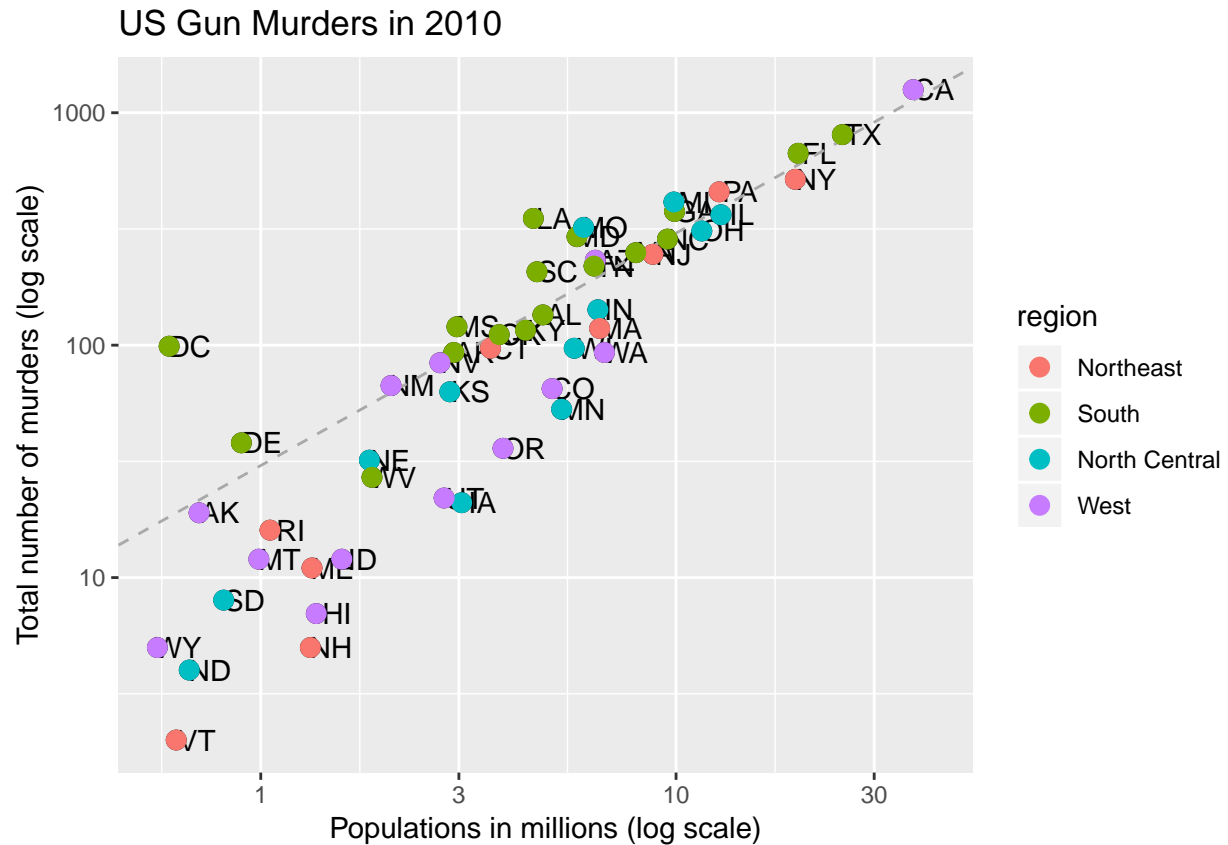
Le añadimos una línea con pendiente 1 e intercepta el log10 de ese ratio medio:

```
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
```

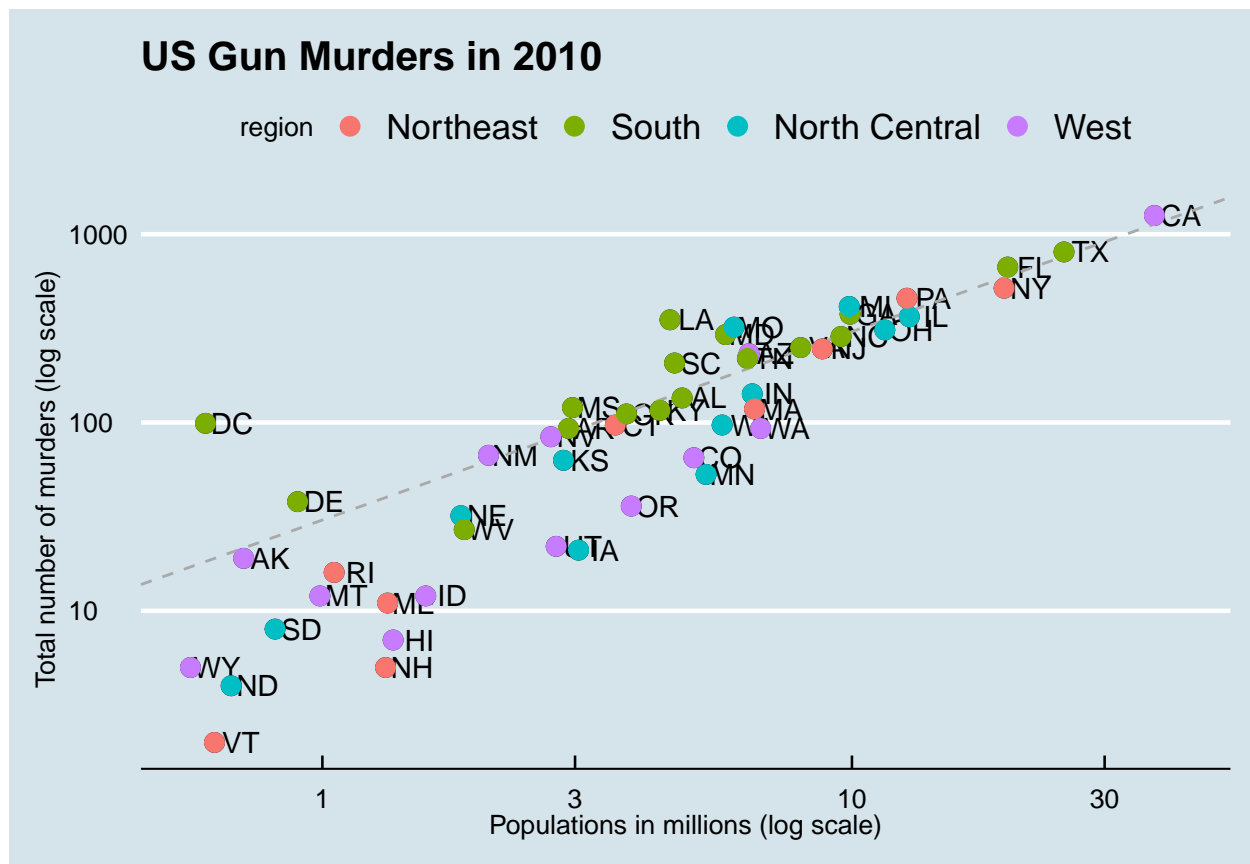


Add-on packages

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)

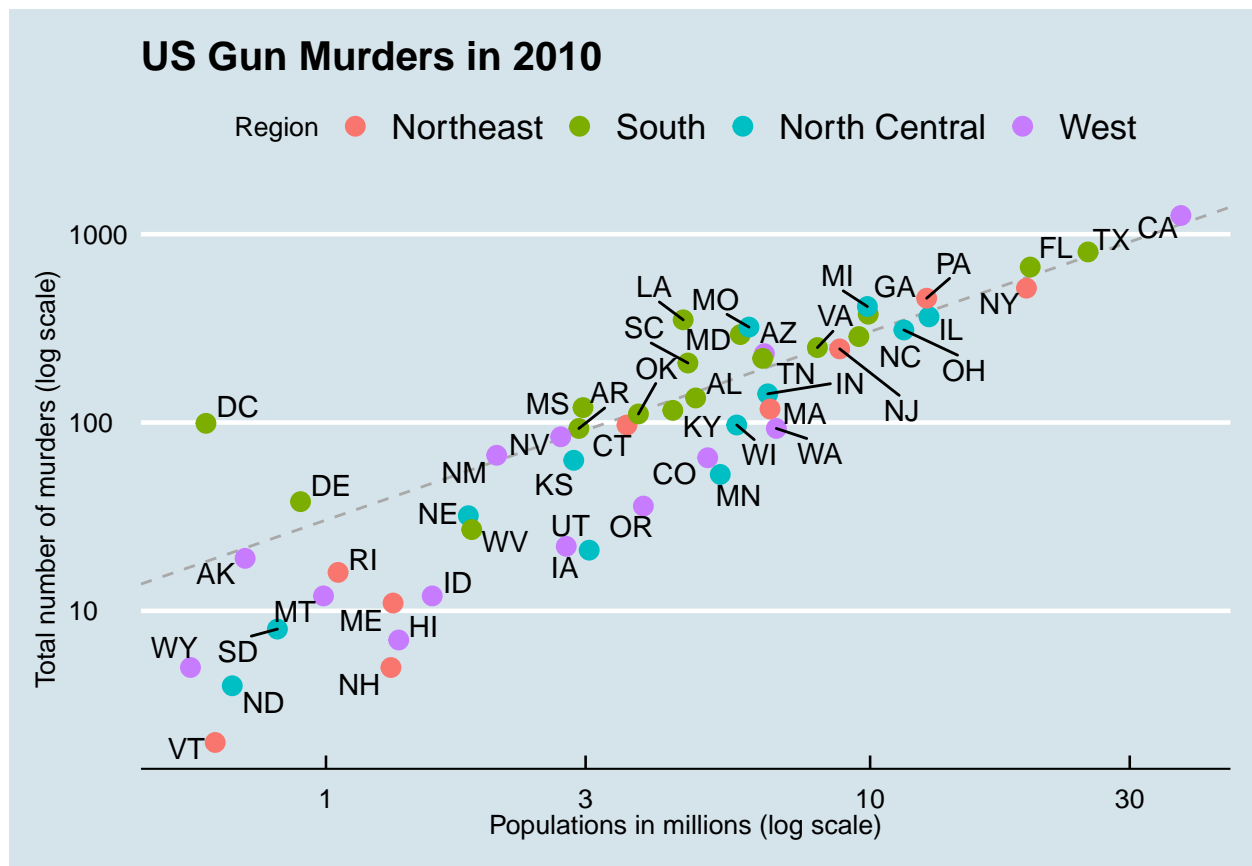
p + theme_economist()
```



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

