ggplot2 en R

2024-09-25

gráficos con ggplot2

ggplot2 es un paquete de código muy popular para crear gráficos a partir de sets de datos, en el siguiente documento vamos a desglosar su composición y explicar su funcionamiento para comprenderlo.

El primer paso es importar las librerías que serán utilizadas. Se hace con la función `library()`, para cargar los paquetes a nuestro espacio de trabajo.

```
## Libraries

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
```

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

```
## Warning: package 'ggrepel' was built under R version 4.3.3
```

```
library(dslabs)
```

dplyr Esta librería se usa para manipular datos a través de métodos que se encaragn de manejar los datos y facilitar su transformación.

ggplot2 La librería más popular en R para hacer gráficos, utiliza una técnica llamada *Gramática de gráficos*, que separa los componentes que estos pueden incluir para manipular cada uno de forma independiente.

ggthemes Una extensión de ggplot2, que incluye más opciones de apariencias para los gráficos, los hace más legibles y claros sin tener que manipular cada dato individualemnte. "Los hace más bonitos".

ggrepel Este paquete incluye alternativas para la parte *geom* de los gráficos que modificamos con ggplot2. Sus funciones mejoran la legibilidad de los gráficos al hacer que no se encime el texto con otro.

dslabs La librería anterior trae consigo bases de datos de ejemplo, entre ellas está *murders*, la cuál incluye los datos de la cantidad de homicidios por estado que hubo en Estados Unidos.

```
# Objetos ggplot
data(murders)
head(murders)
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

La función `data()` carga a nuestro espacio de trabajo la base de datos *murders* incluida en las librerías que importamos, mientras que `head()` nos muestra las primeras 6 filas del dataset.

```
# Se crea un objeto ggplot y se despliega un fondo gris
# (falta definir la geometría)
ggplot(data = murders)
```


El comando `ggplot()` es el que realiza la base de los gráficos a crear y define el conjunto de *data* que se utilizará. Por sí sólo no puede crear un gráfico completo, si no que hace la capa inicial sobre la que se añadiran más elementos. El bien sabido éxito de ggplot2 está en poder definir todos los elementos de las

gráficas de forma individual, es por esto, que el simple uso del comando solo despliega un fondo gris, al cual es necesario que indicarle la parte *aes* que es la información de las variables que serán visibles, y la parte *geom* que define su geometría.

```
# Usando un pipe
murders |> ggplot()
```


Un *pipe* `|>` es un operador que encadena funciones de una forma más clara, convierte la salida en argumento del comando. El output de este código es igual que el pasado [`ggplot(data = murders)`], pero de la igual manera hace falta agregar el resto de los componentes del gráfico, por lo que solo desplegará un lienzo gris.

```
# Asignando a un objeto (no se despliega hasta imprimirlo con print o directo)
p <- ggplot(data = murders)
class(p)
```

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

```r
print(p)
```



```r
#p
```

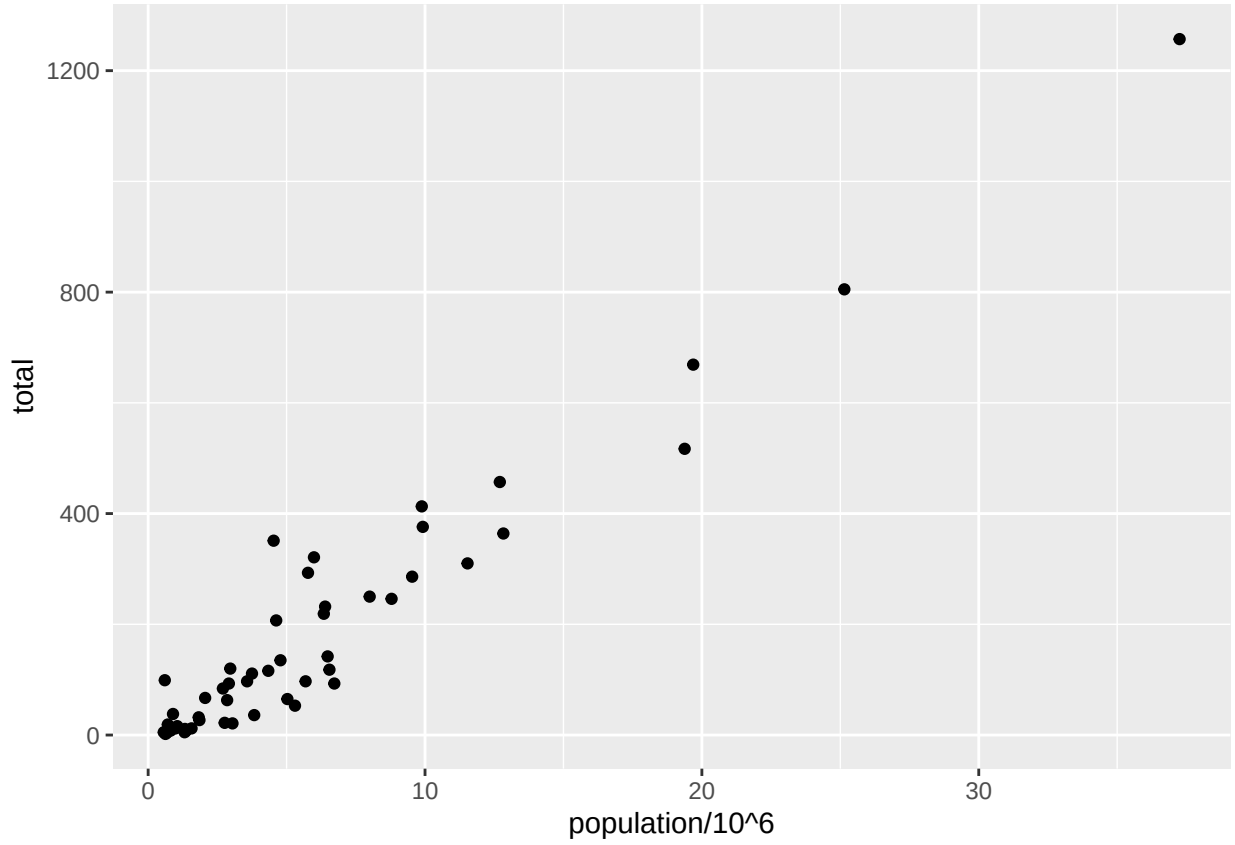
Ahora se creó la base de un gráfico utilizando `ggplot()`, y se le asignó a un objeto `p`, el cual contiene nuestra gráfica. Se puede verificar el tipo de dato que tenemos con el comando `class()`, en este caso, debería regresar un valor *gg*, que representa gráficos provenientes de ggplot2. El comando `print()` solo imprime nuestro lienzo, o bien, podemos colocar el objeto en la consola escribiendo su nombre como está en la línea siguiente. Para fines prácticos se fijo como comentario para no volver a imprimir la base gris.

geometrías y mapeos estéticos

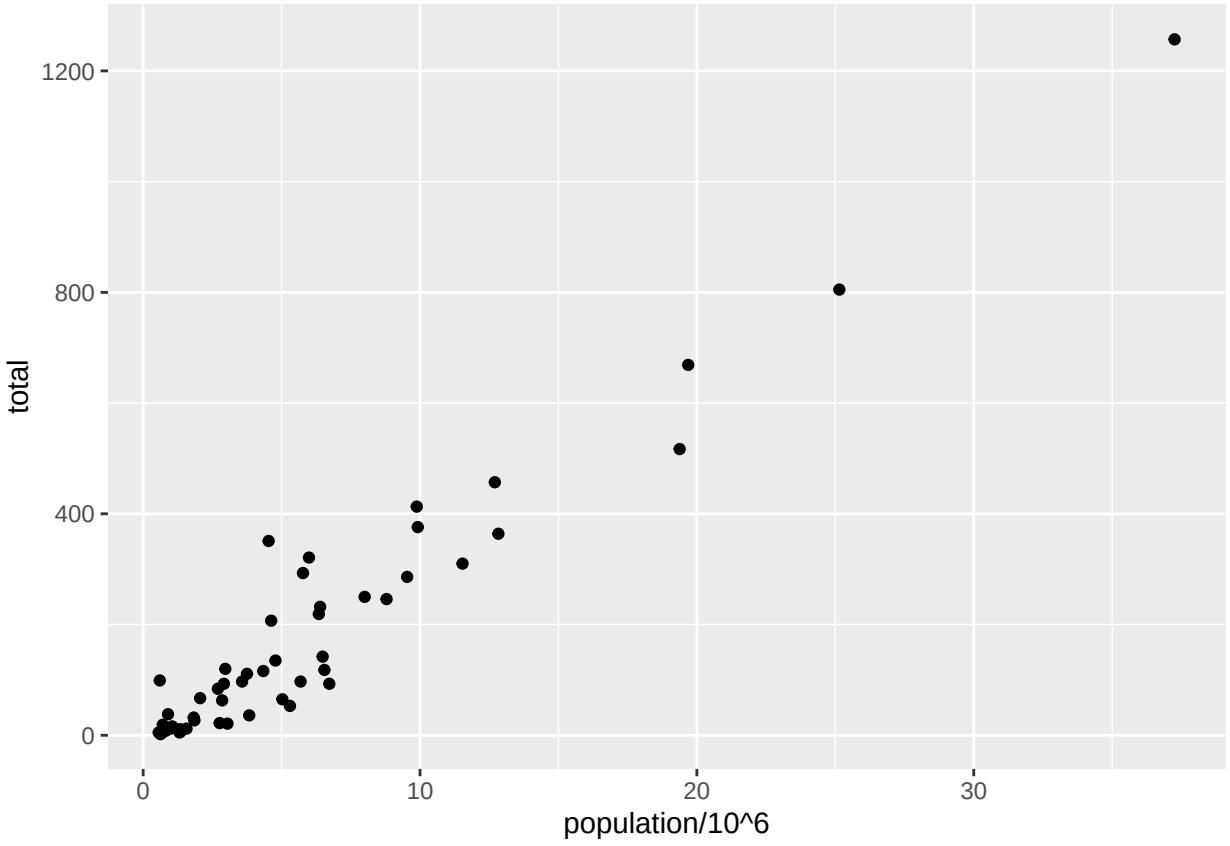
A continuación tenemos tres formas de utilizar la función `ggplot()` para crear un tipo de gráfico en específico. En este caso se va a crear una gráfica de dispersión, la cual añadimos con el comando `geom_point()`. Ya tenemos la primera capa, ahora es necesario indicar las variables con las que queremos que trabaje nuestro gráfico, para ello se utilizará `aes()`, con los argumentos de `x` y `y` que le asginan a cada eje una variable de información.

Así como vimos en códigos pasados, podemos utilizar el operador *pipe* para asignar un dataset a la función `ggplot()`, o bien declarar un objeto *gg* y a este añadirle el comando para definir el gráfico. Ambos tienen el mismo output. Por último, está la forma clásica de declarar un gráfico anidando funciones, empezando con la principal `ggplot()` e incluyendo en sus argumentos el dataset a utilizar, seguido de la función `aes()` con las variables a ser visualizadas, para terminar, se le añade la geometría del gráfico que en este caso es uno de dispersión (scatterplot).

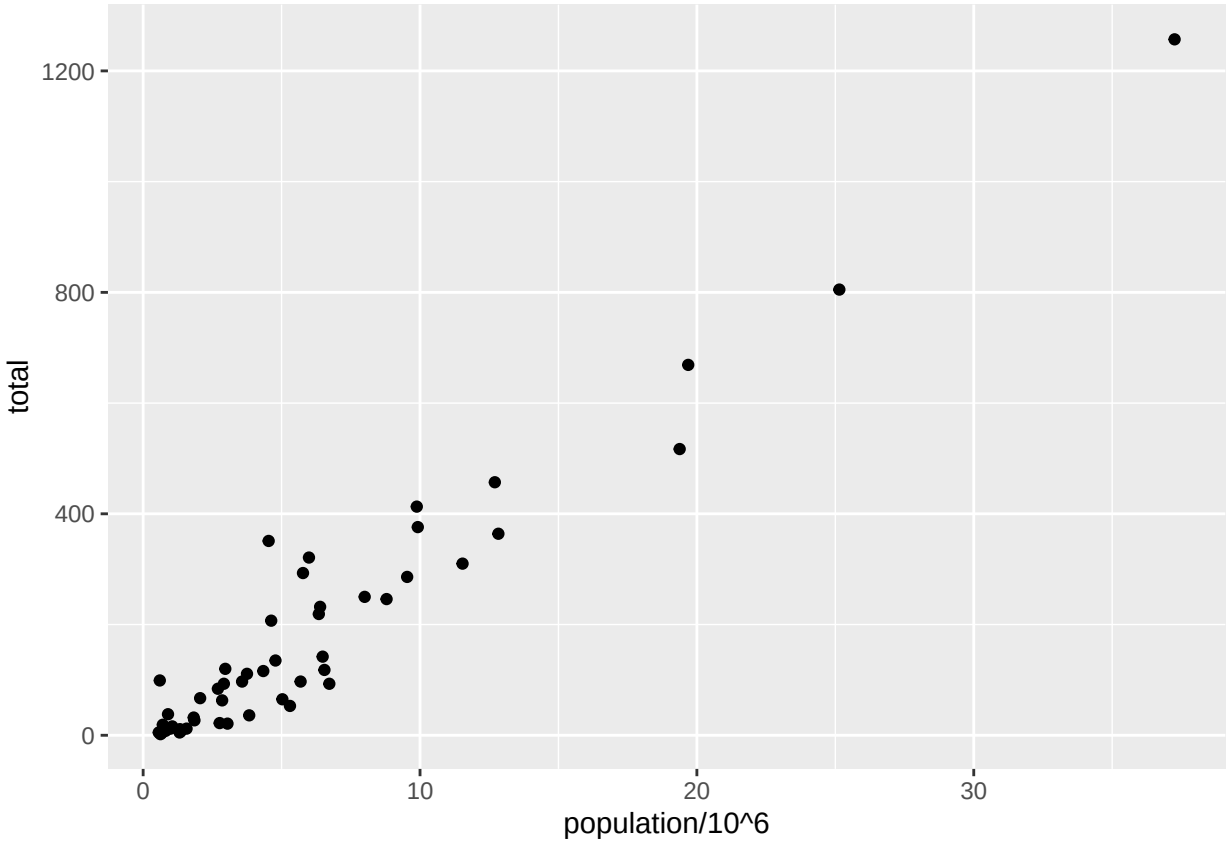
```r
# Geometrías y mapeos estéticos

# Agregando la capa de geometría y mapeo estético utilizando el pipe para asignar el dataset a la funci
murders |> ggplot() +
  geom_point(aes(x = population/10^6, y = total))
```




```r
# Agregando la capa de geometría y mapeo estético a un objeto ggplot
p + geom_point(aes(population/10^6, total))
```




```
# Disposición clásica de un gráfico en ggplot2
ggplot(murders, aes(x = population/10^6, y = total)) +
  geom_point()
```



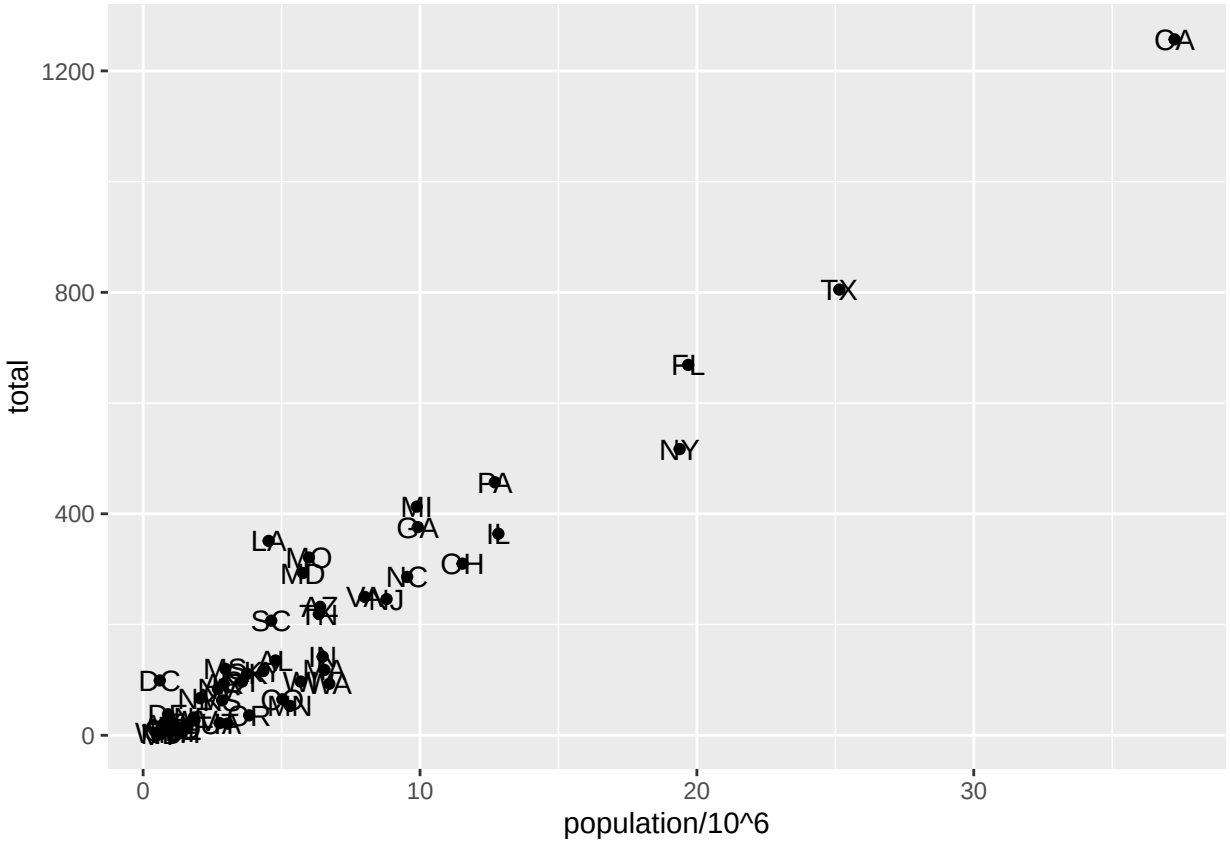
capas

Ya que se creó la base del gráfico, su estructura, y sus variables se visualizan, podemos seguir agregando capas para añadirle componentes a la gráfica. Como tenemos un gráfico de dispersión, podemos pensar en agregar etiquetas para cada punto de información. Las tres siguientes líneas de código son ejemplos de cómo hacerlo, las primeras dos son equivalentes, como vimos con la geometría del gráfico.

En el primer ejemplo se le agrega a lo que teníamos previamente, que es el comando `geom_point()` y el objeto delcarado (que contiene la información sobre la geometría del gráfico y el dataset) la función `geom_text()` con los argumentos de las mismas variables y más elementos gráficos que se añaden individualmente, en este caso, se usa `label` igualado a `abb` que son las abreviaciones de los estados al que pertenece el número de asesinatos. El segundo ejemplo es equivalente, pero en este se asigna el gráfico a un objeto llamado `p_test`, el cual ahora contiene la información que incluimos de nuestra gráfica.

Por último, tenemos una línea de código con un error. Como podemos ver se quiere agregar la capa de `label` al gráfico, pero todas las capas que se quieran agregar deben de ser un argumento dentro de la función `geom_text()` y en este caso está fuera, por lo que no compilaría.

```
# Capas

# Agregando una etiqueta a cada punto para identificar el estado
p + geom_point(aes(population/10^6, total)) +
  geom_text(aes(population/10^6, total, label = abb))
```

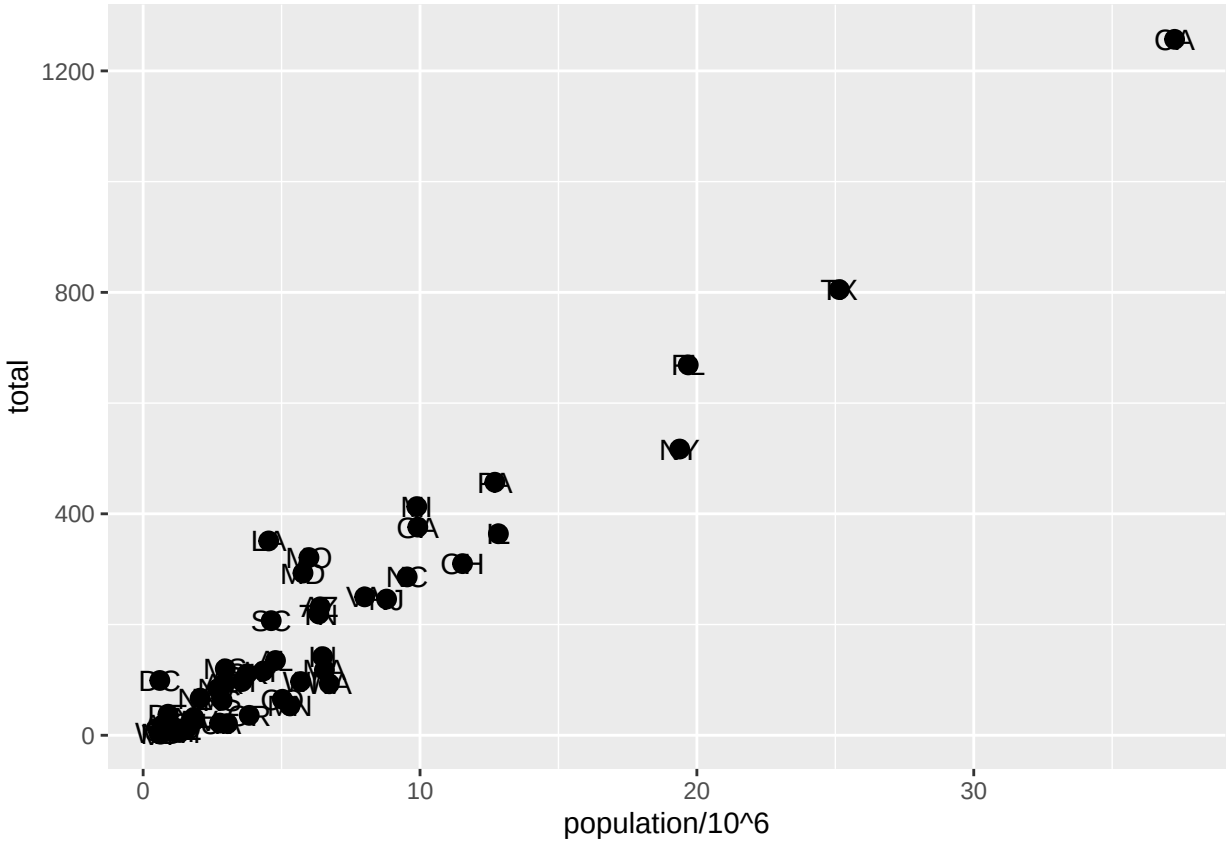



```
# Equivalente al anterior
p_test <- p + geom_text(aes(population/10^6, total, label = abb))

# ERROR: debe estar label dentro de aes()
# p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

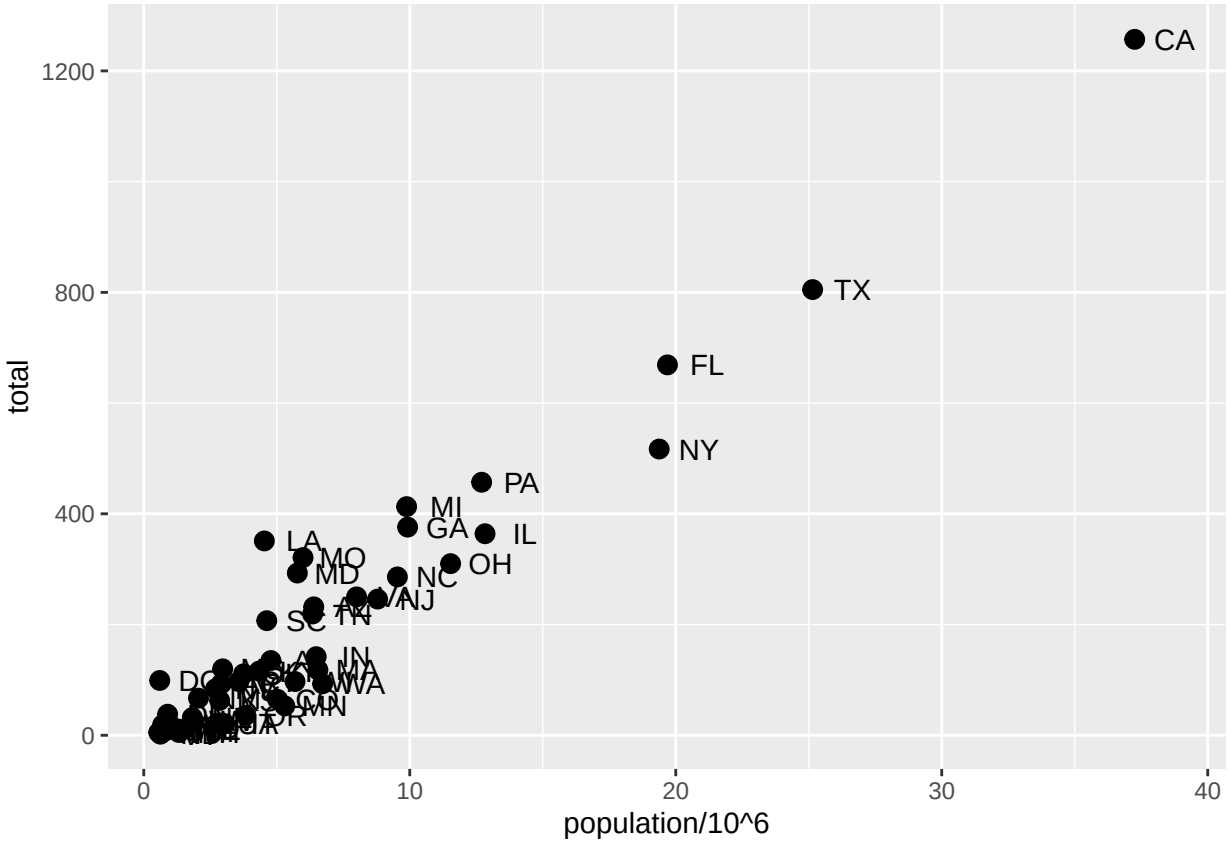
probar varios argumentos

El éxito de los gráficos claros y legibles radica en elementos visibles y ajustados individualmente para cada gráfico, en este ejemplo lo primero que se hace es agregarle las funciones al objeto *ggplot* argumentos para fijar el tamaño de las etiquetas en `geom_point()` con el comando `size`, además de las abreviaciones que ya agregamos. En adición a eso, con `nudge_x` desplazamos las etiquetas ciertas unidades para que sean fácilmente entendidas.

```
## Cómo probar varios argumentos

# Cambiar el tamaño de la etiqueta con size
p +
  geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```




```
# Mover la etiqueta con nudge_x
p +
  geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```



mapeos estéticos globales vs locales

El *mapeo estético* se refiere a la forma en la que se conectan las variables de información en los datasets con los elementos visuales del gráfico. Existen las globales, son las que se ponen dentro de la función `ggplot()` y por tanto se situan en la capa base que afecta a todo el gráfico. Por el otro lado, el mapeo estético local se aplica solo a una capa específica y se puede personalizar por capa.

En el ejemplo de abajo podemos ver cómo primero aplica `aes()` a la función que crea el lienzo base del gráfico, y con esto globaliza las variables para todas las capas. En la segunda parte, anula esto al incluir la información de las variables en las funciones que definen la estética del gráfico en otras capas, `geom_point()` y `geom_text()`.

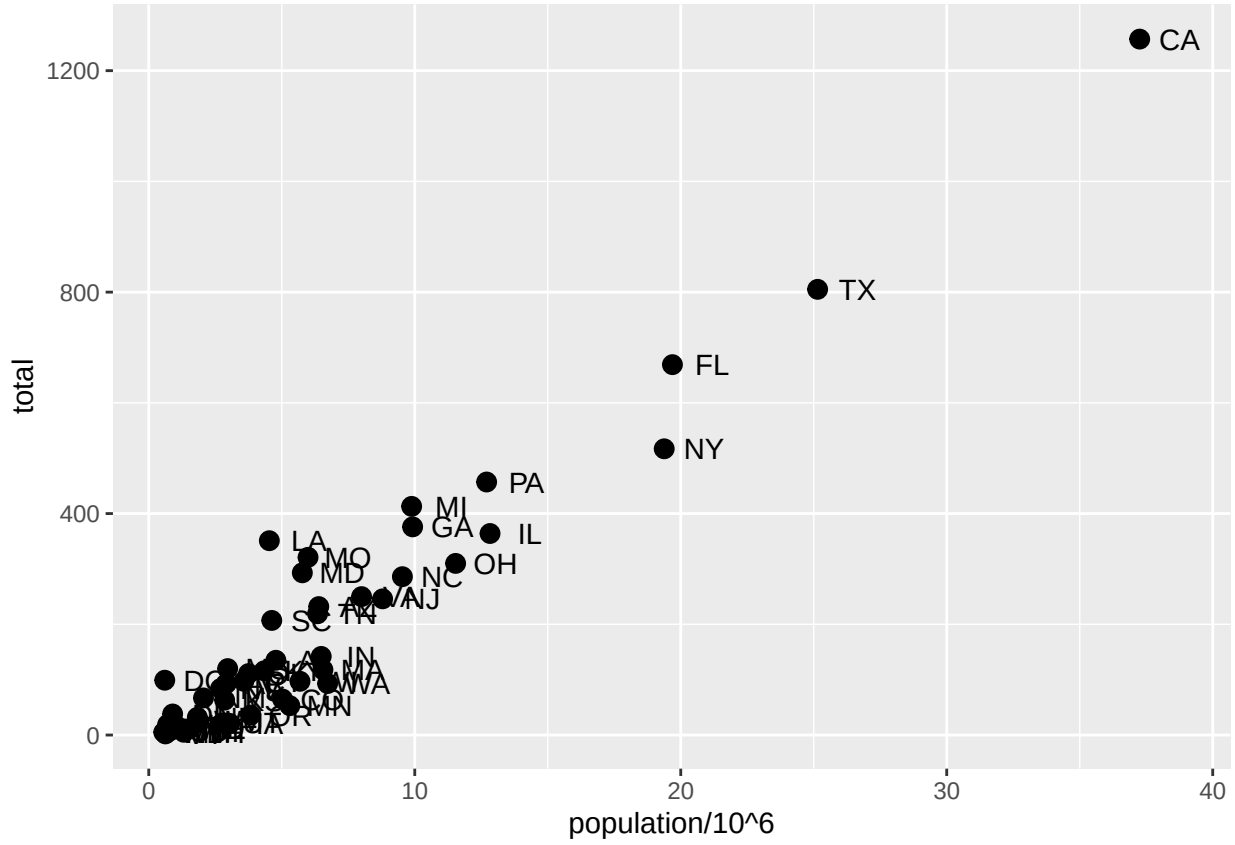
```
# Mapeos estéticos globales versus locales
```

La función `args()` se utiliza para visualizar los argumentos que deben ir dentro de una función, en este caso usamos `args(ggplot)`.

```
# Argumentos del objeto ggplot
args(ggplot)
```

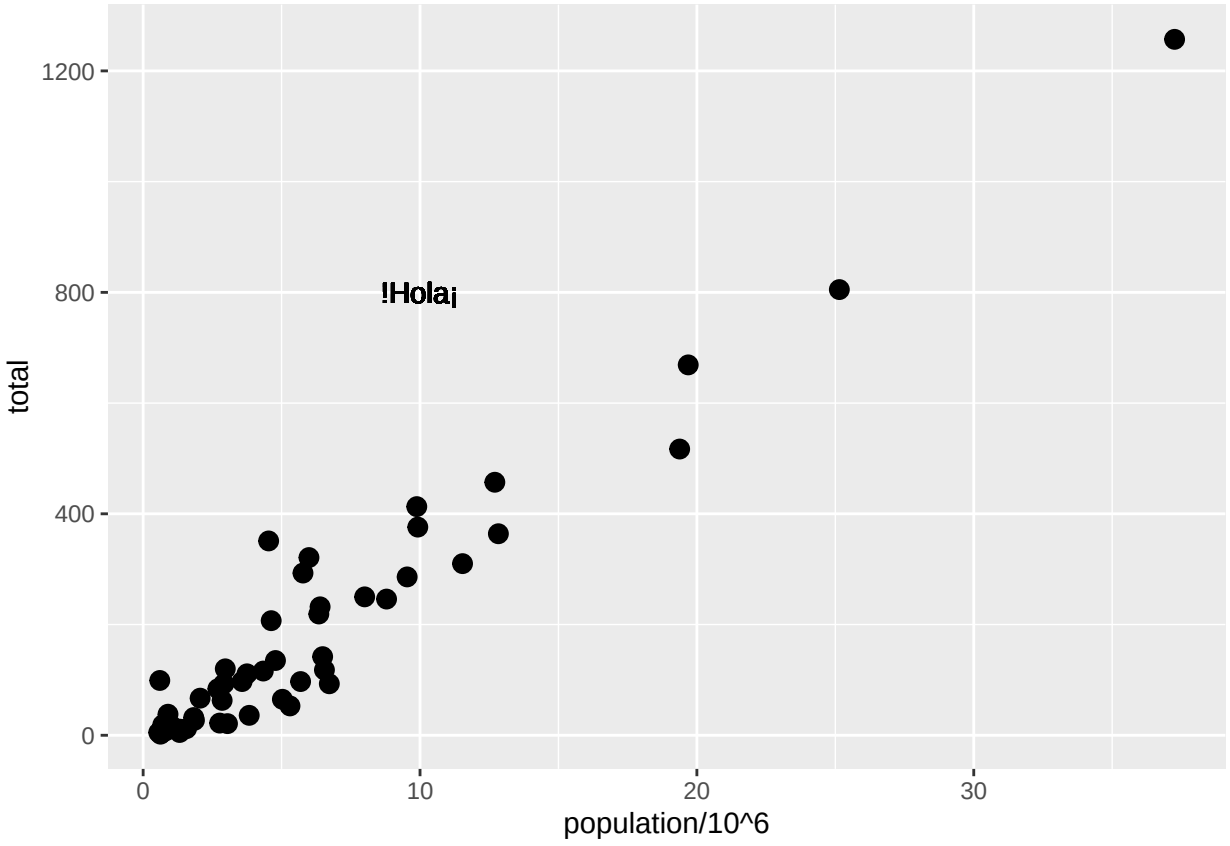
```
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## NULL
```

```
# Se asigna el mapeo estético al objeto de manera global
p <- murders |> ggplot(aes(population/10^6, total, label = abb))
p +
  geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```




```
# Se anula el mapeo global definiendo un nuevo mapeo dentro de cada capa
p +
  geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "!Hola¡"))
```

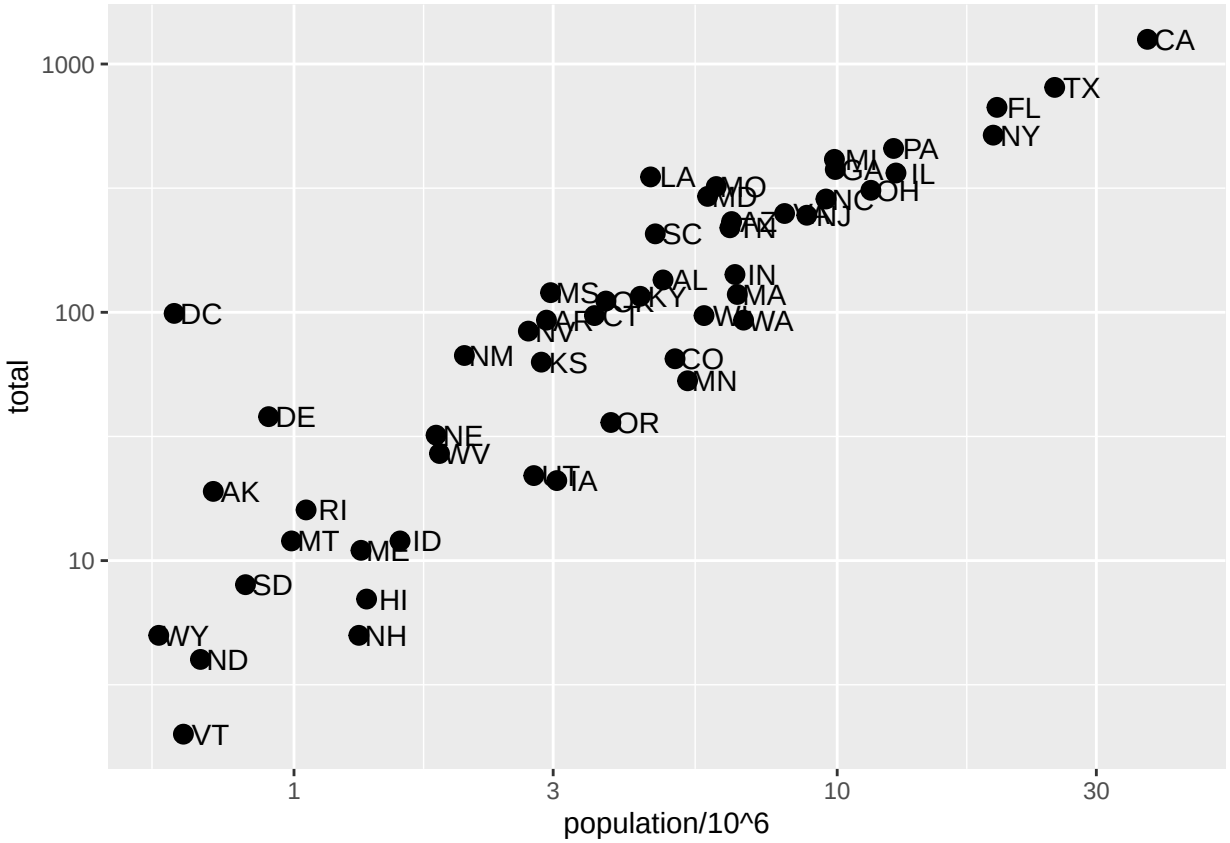
```
## Warning in geom_text(aes(x = 10, y = 800, label = "!Hola¡")): All aesthetics have length 1, but the d
## i Please consider using ‘annotate()‘ or provide this layer with data containing
##   a single row.
```



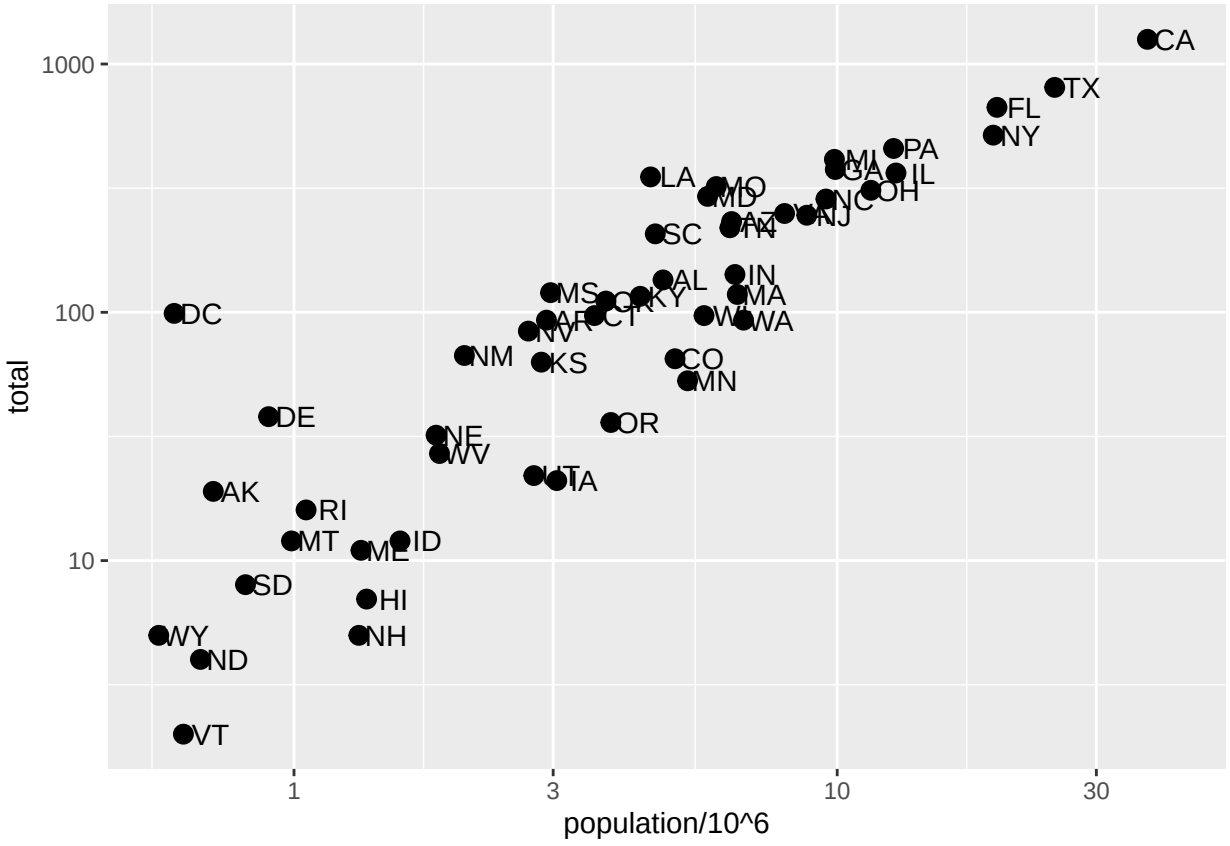
escalas

Para visualizar mejor los datos, es conveniente ajustar la escala en la que se presentan. Para ello se utiliza `scale_x_continuous(trans = "log10")` y `scale_y_continuous(trans = "log10")` agregados a las funciones que ya teníamos, esto transforma la escala a una logarítmica de base 10 porque el argumento así lo indica, pero la siguiente forma de hacerlo es con `scale_x_log10() + scale_y_log10()`, la cual transforma la escala directamente a una logarítmica. Eso significa que nuestros datos estarán distribuidos siguiendo una secuencia en bases de 10, (10, 100, 1000, etc...) en ambos ejes, haciendo más fácil de interpretar la información.

```
# Escalas

# Cambiar las escalas con scale_x_continuous
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



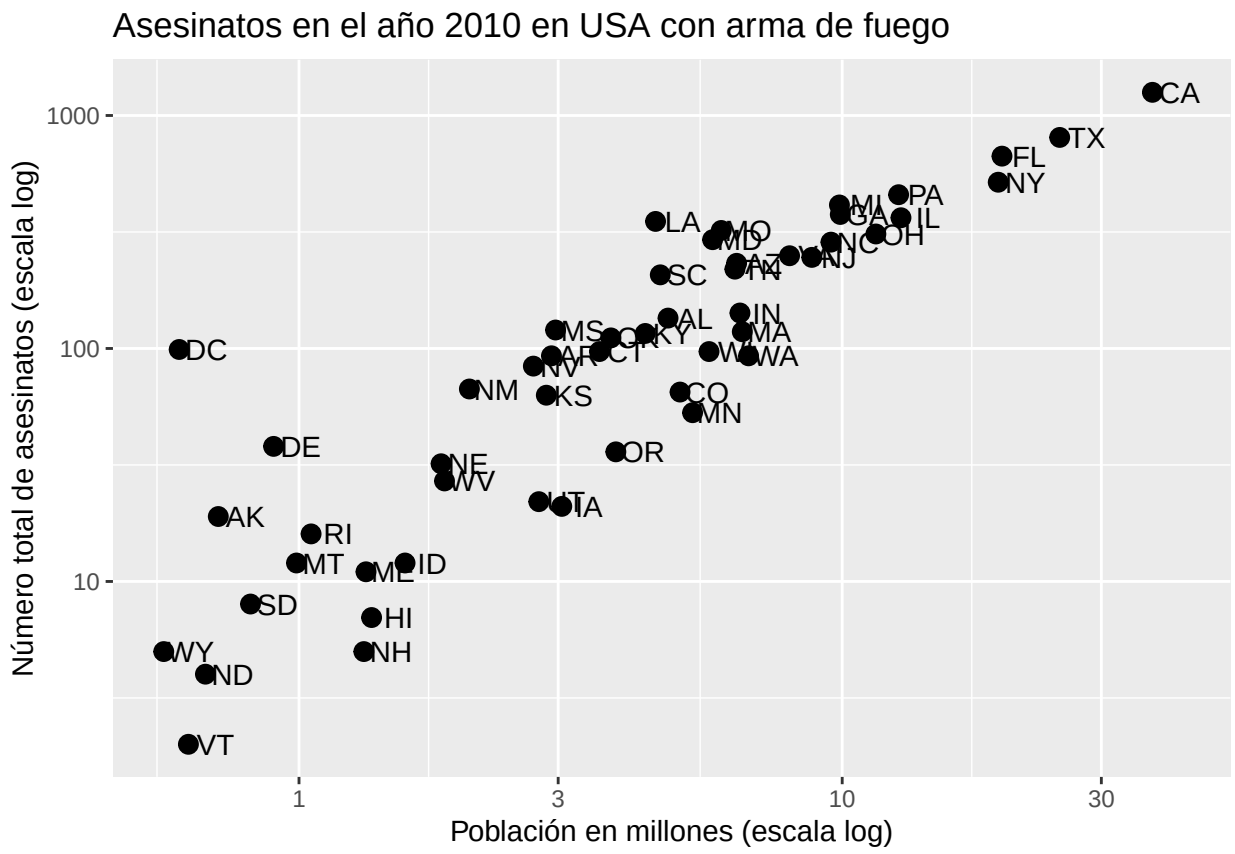

```
# Cambiar las escalas scale_x_log10
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```



etiquetas y títulos

Un buen gráfico debe incluir un título que represente correctamente la información que presenta, además de etiquetas que expliquen las variables que visualizamos. Para añadirlos usamos las funciones `xlab("") + ylab("") + ggtitle("")` agregadas al resto de funciones que diseñan el gráfico.

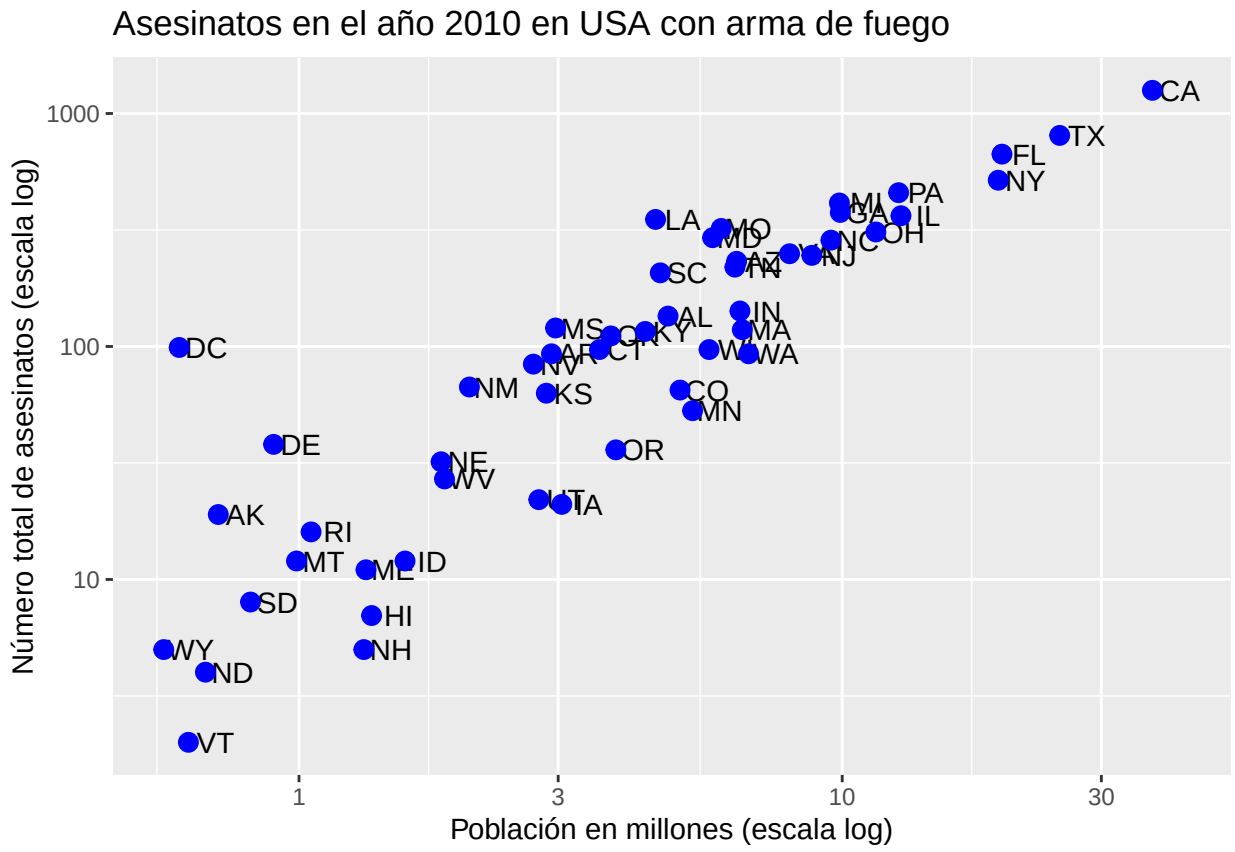
```
# Etiquetas y títulos

# Cambiar las etiquetas con xlab y el título con ggtitle
p +
  geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Número total de asesinatos (escala log)") +
  ggtitle("Asesinatos en el año 2010 en USA con arma de fuego")
```



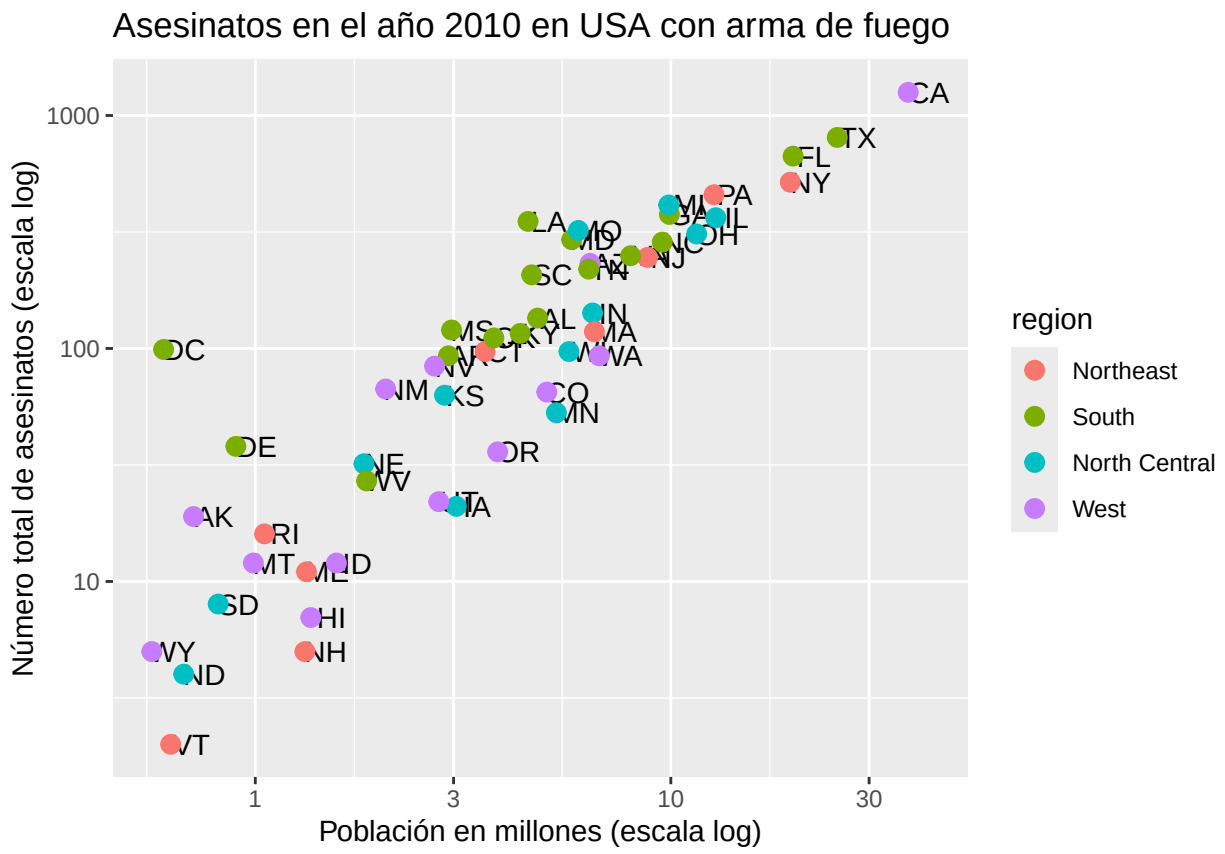
colores

Algo importante para que los gráficos no sean monótonos y aburridos es agregar color. Para hacerlo se utiliza el argumento `color` dentro de la función que define la geometría, lo que afectará en este caso a todos los puntos del gráfico. En el otro ejemplo se usa `col=region`, que colorea los puntos según la región de la que se trate. Estos comandos son muy útiles en la creación de gráficas de buena calidad.

```
# Categorías como colores

p <- murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Número total de asesinatos (escala log)") +
  ggtitle("Asesinatos en el año 2010 en USA con arma de fuego")

# Cambiar color de los puntos a azul
p + geom_point(size = 3, color ="blue")
```




```
# Cambiar color con la variable categórica region
p + geom_point(aes(col=region), size = 3)
```



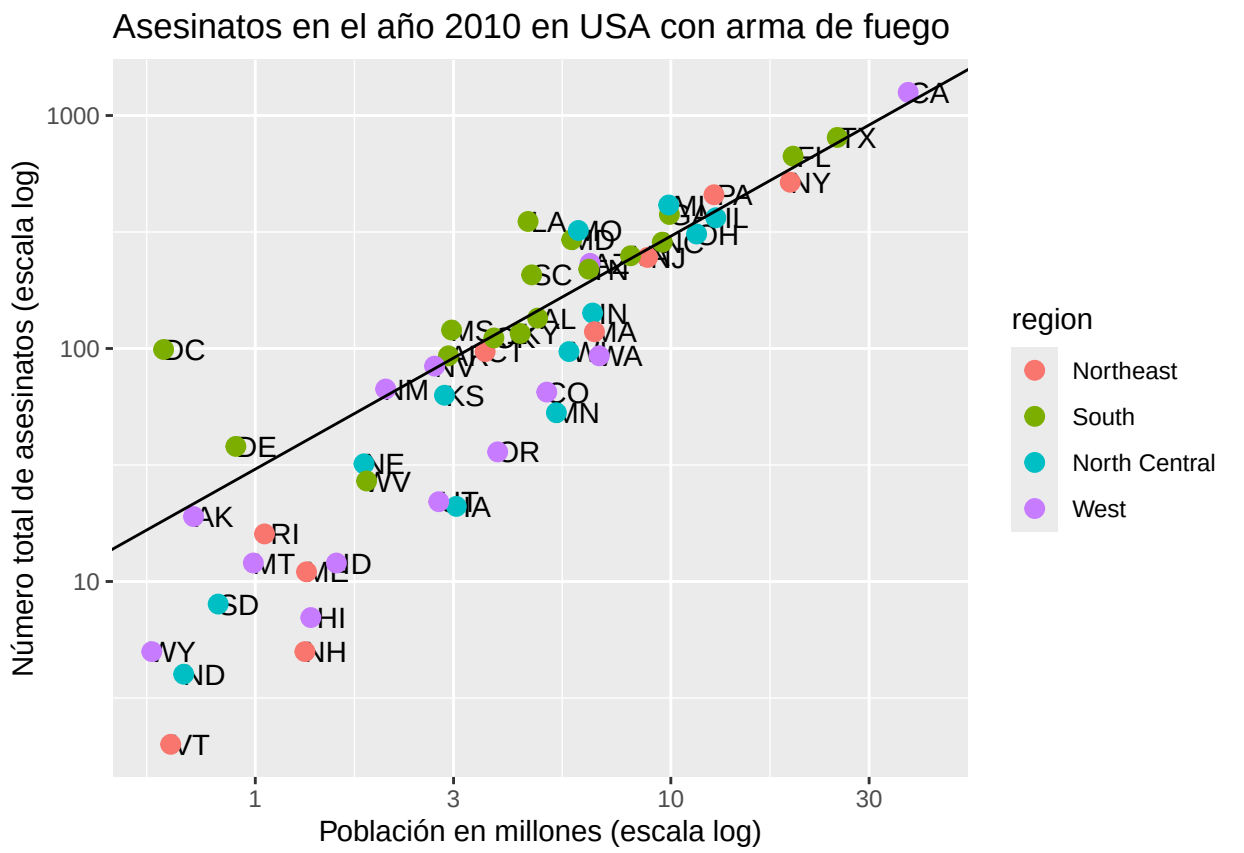
anotación, formas y ajustes

Para un scatterplot es una buena práctica agregar una línea de tendencia que represente el promedio de los datos. Primero se crea el objeto `r` al que se le asigna el resultado del cálculo de la tasa promedio. Se usa el operador pipe para cargar el datasets *murders* para hacer las operaciones, después `summarize()` para calcular un valor resumen del conjunto de datos utilizado y `sum()` para sumar todos los valores del argumento. Por último el operador pipe en `pull(rate)`, que extrae el valor calculado y lo asigna a la variable r.

Ya que tenemos la variable, necesitamos construir la línea. `geom_abline()` añade una línea recta al gráfico con una pendiente que, en este caso, vale r. Además se escala a forma logarítmica de base 10. Después de crear la línea podemos estilizar sus propiedades redefiniendo el objeto *p* con una nueva línea que contenga argumentos dentro de la misma función que modifiquen su estilo. En el ejemplo se utiliza `lty = 2`, que cambia el estilo de la línea a una discontinua, y `color = "darkgrey"` que cambia el color al que está especificado. Para finalizar, en la última línea volvemos a redefinir a *p* para reiterar su geometría, ya que después de redefinirlo su valor previo se elimina y debemos incluirlo nuevamente. Con todo esto formando nuestra línea de tendencia, agregamos `scale_color_discrete(name = "Region")` que agrega un índice a los colores utilizados para darle color a las diferentes regiones.

```
# Anotación, formas y ajustes

# Añadir una línea que represente la tasa promedio de asesinatos en todo el país
r <- murders |>
  summarize(rate = sum(total)/ sum(population) * 10^6) |>
  pull(rate)

# agregar la línea al objeto p
```

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

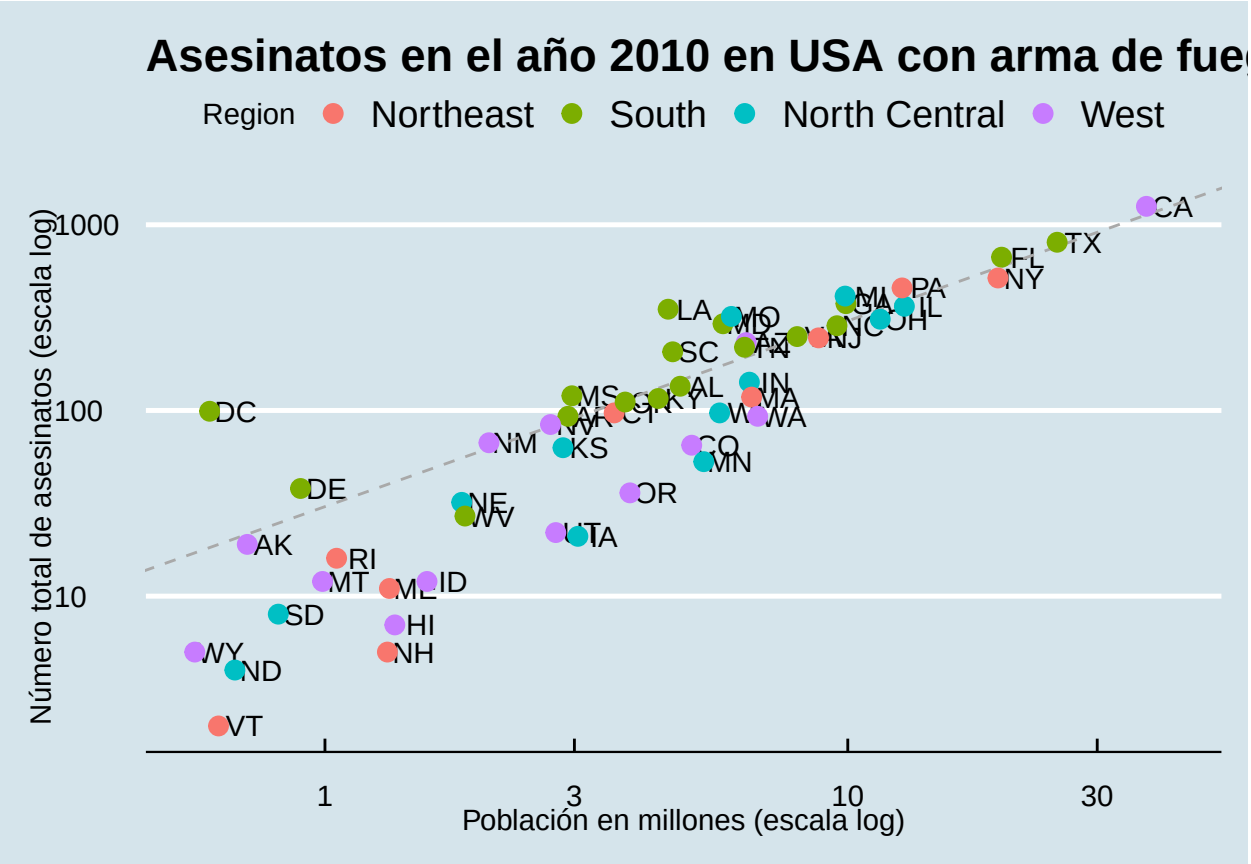



```
# redefinir p para modificar su estética
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) # añadirle a p lo que se perdió en la redefinición

# agregar índice de color a Región
p <- p + scale_color_discrete(name = "Region")
```

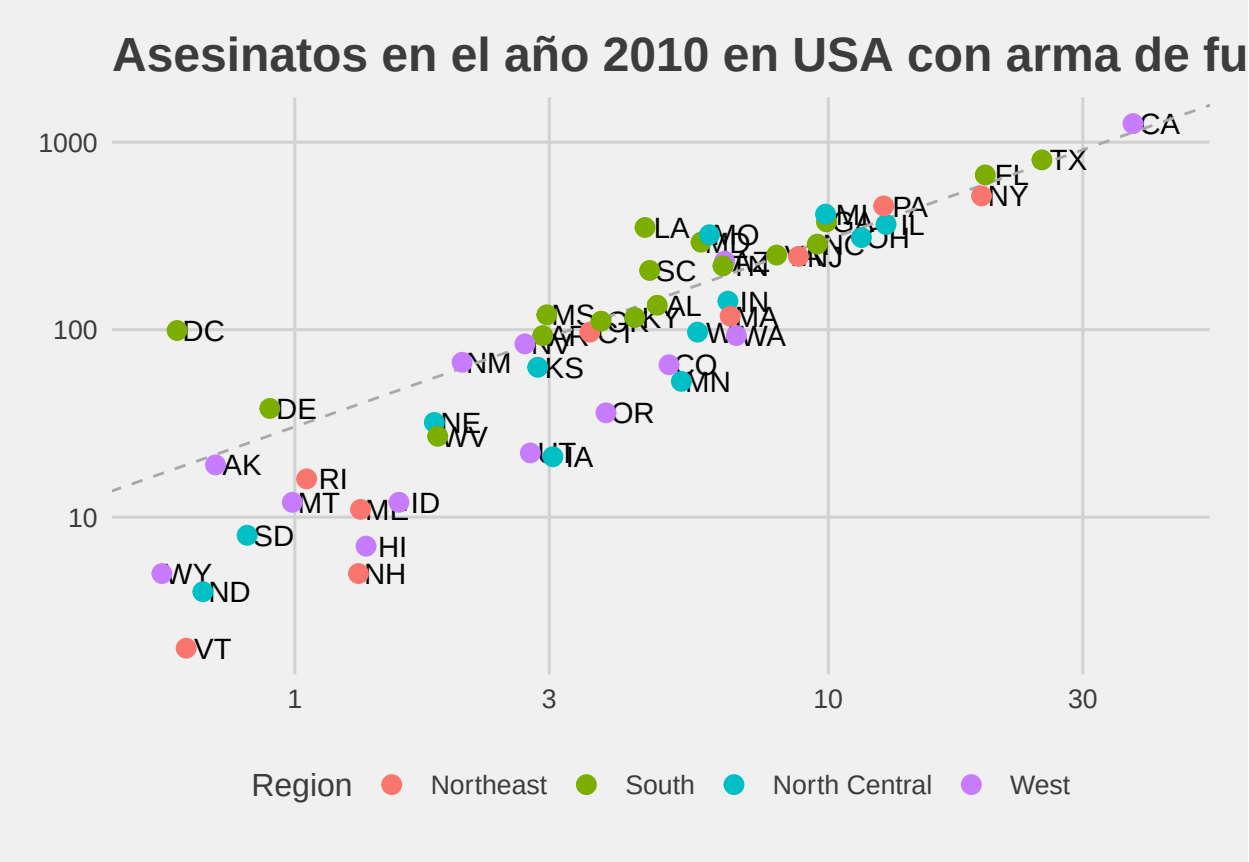
paquetes complementarios

Las librerías que importamos al principio traen consigo temas que se pueden aplicar a los gráficos para alterar su paleta de colores. La función `ds_theme_set()` establece por defecto un tema que afecta a todas las gráficas creadas a partir de ese punto. En la siguiente línea se le añade al objeto *p* el tema llamado *economist*, y después el tema *fivethirtyeight*. Estos crean diferentes versiones estéticas del mismo gráfico y a gusto personal se elige el más adecuado.

```
# Paquetes complementarios

# Se establece un tema por defecto usando una función del paquete dslabs
ds_theme_set()

# Se aplica el tema economist
p + theme_economist()
```



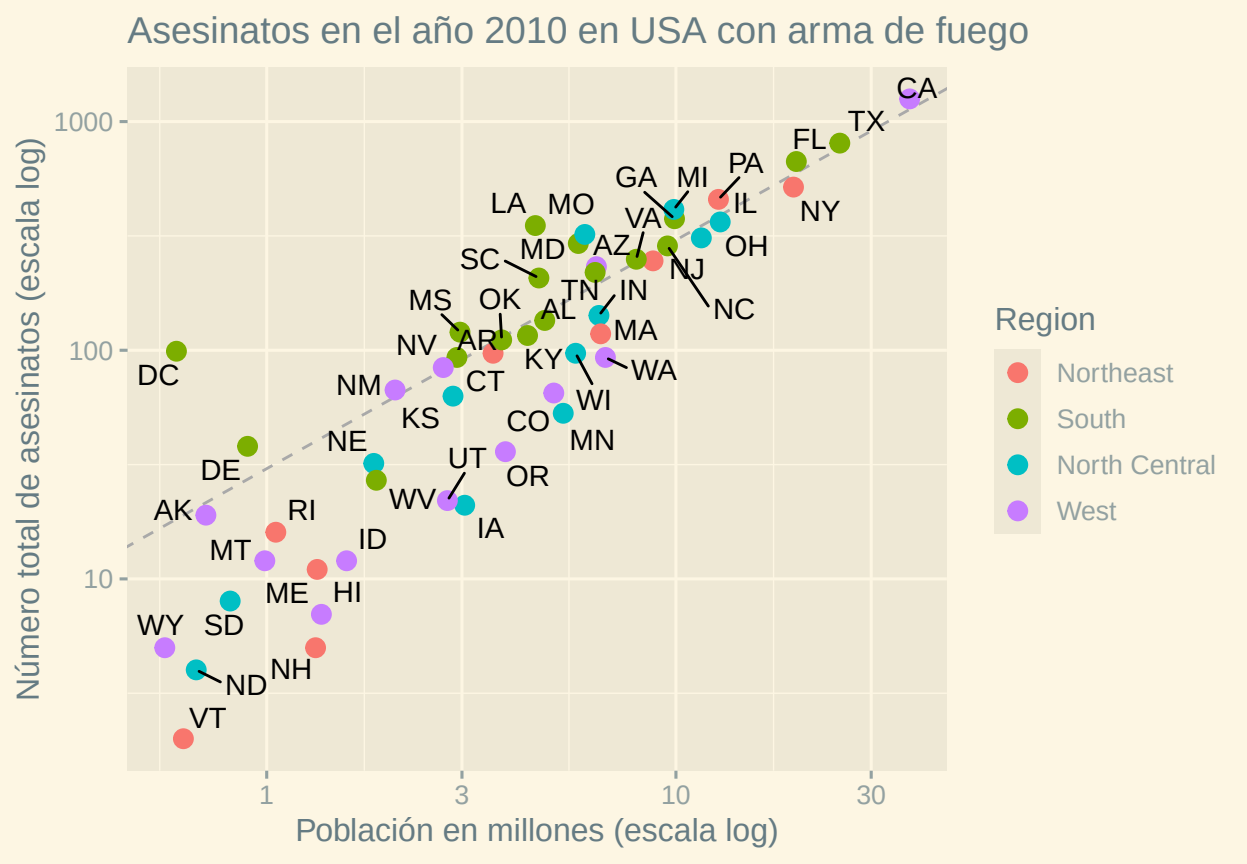

```
# Se aplica el tema fivethirtyeight
p + theme_fivethirtyeight()
```



combinarlo todo

Ya que conocemos cómo agregar de manera individual diferentes elementos de los gráficos, podemos combinar todo para obtener represantaciones visuales de información, que sean de alta calidad y sobre todo útiles para el análisis de datos.

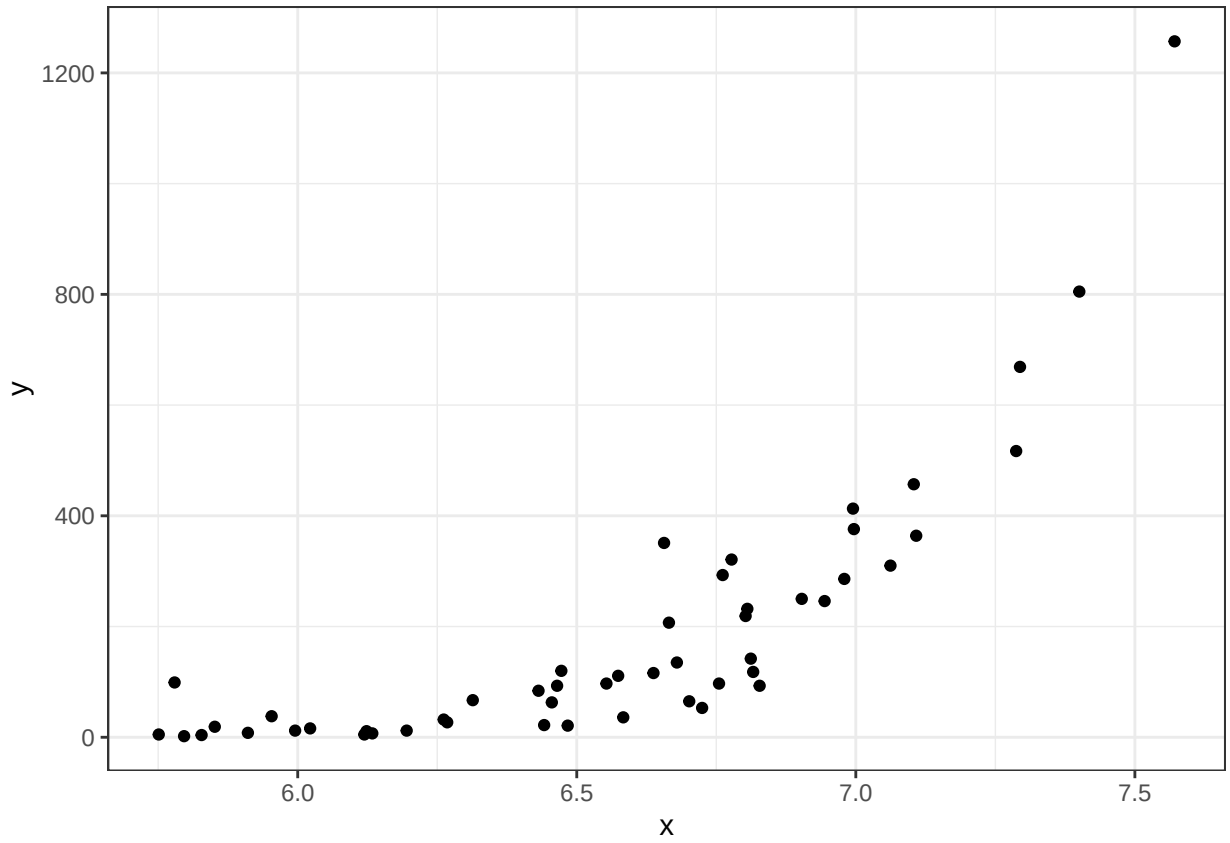
```
# Combinarlo todo

r <- murders |>
  summarize(rate = sum(total)/ sum(population) * 10^6) |>
  pull(rate)

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("Población en millones (escala log)") +
  ylab("Número total de asesinatos (escala log)") +
  ggtitle("Asesinatos en el año 2010 en USA con arma de fuego") +
  scale_color_discrete(name = "Region") +
  theme_solarized_2()
```



qplot

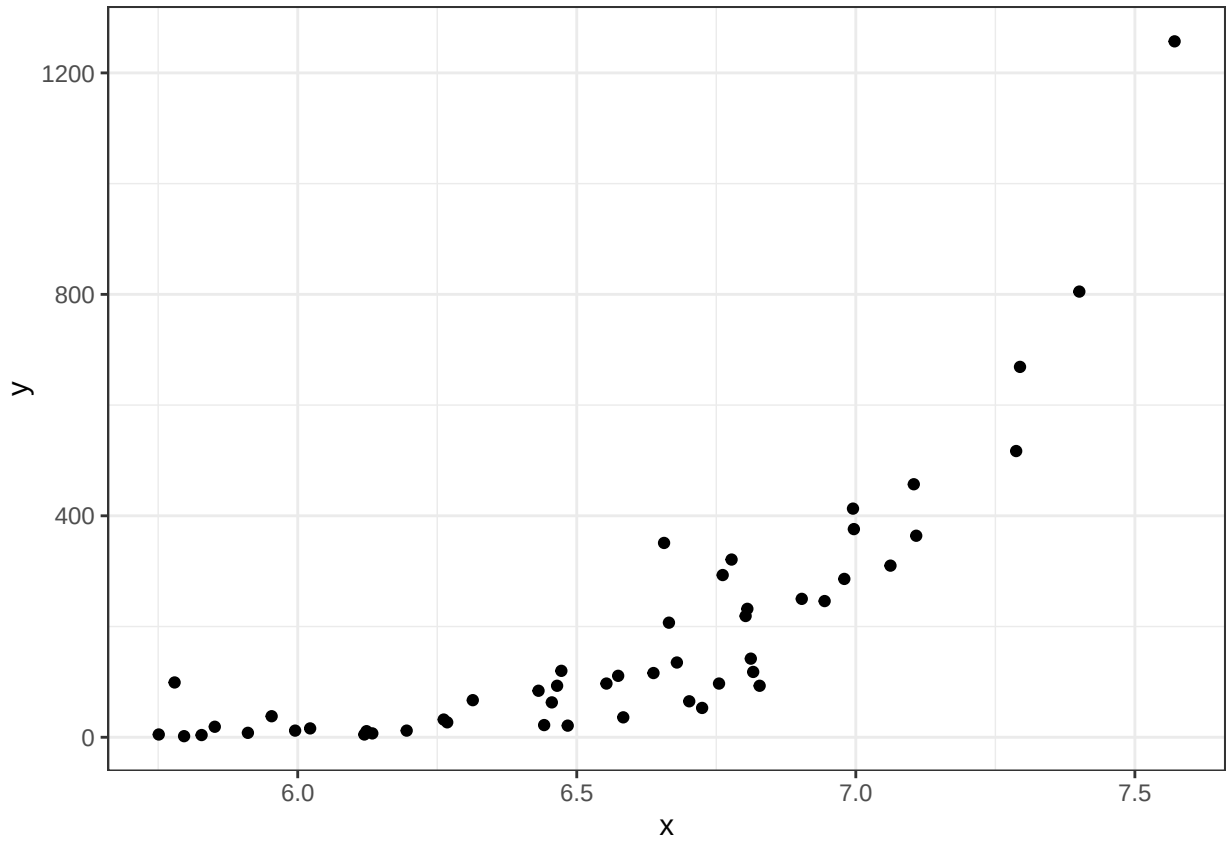
qplot es una versión simplificada de ggplot para hacer los llamados gráficos rápidos, no permite modificar individualmente cada elemento, pero es más práctico para ciertos sets de datos. En nuestro ejemplo vemos las dos formas de hacerlo. Primero se preparan los datos, usando `data(murders)` para cargar el dataset y asignando a las variables `x` y `y` los diferentes atributos a analizar. La primera forma presentada es la manera estándar, usando `data.frame(x,y)` para crear, redundantemente, un dataframe con dos columnas determinadas por los argumentos de `x` y `y`. Después este se asigna como argumento de la función `ggplot()` usando el operador pipe y carga este dataframe para crear una gráfico como lo estudiamos anteriormente. En la segunda forma, únicamente usa el comando `qplot()` con los argumentos de las mismas variables sin especificar nada más. Se utiliza muchísimo menos código y tenemos el mismo output, lo que resulta muy práctico dependiendo de la situación que queramos modelar.

```
# Gráficos rápidos con qplot

data(murders)
x <- log10(murders$population)
y <- murders$total

# Estructura clásica de crear un gráfico con ggplot
data.frame(x = x, y = y) |>
  ggplot(aes(x, y)) +
  geom_point()
```




```
# Gráfico rápido
qplot(x, y)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



cuadrícula

Para finalizar, agregamos una cuadrícula detrás del gráfico para facilitar su comprensión, importando la librería `gridExtra` que contiene funciones que nos lo facilitarán. Para el ejemplo utilizamos `qplot()` para crear dos gráficos rápidos y asignar cada uno a una variable diferente [`p1` y `p2`], los cuales serán argumentos de la función `grid.arrange()` junto con `ncol=2`. Este comando nos permite organizar múltiples gráficos en una misma cuadrícula, los modelos están definidos por `p1` y `p2`, y por último especificamos el número de columnas usando `ncol`.

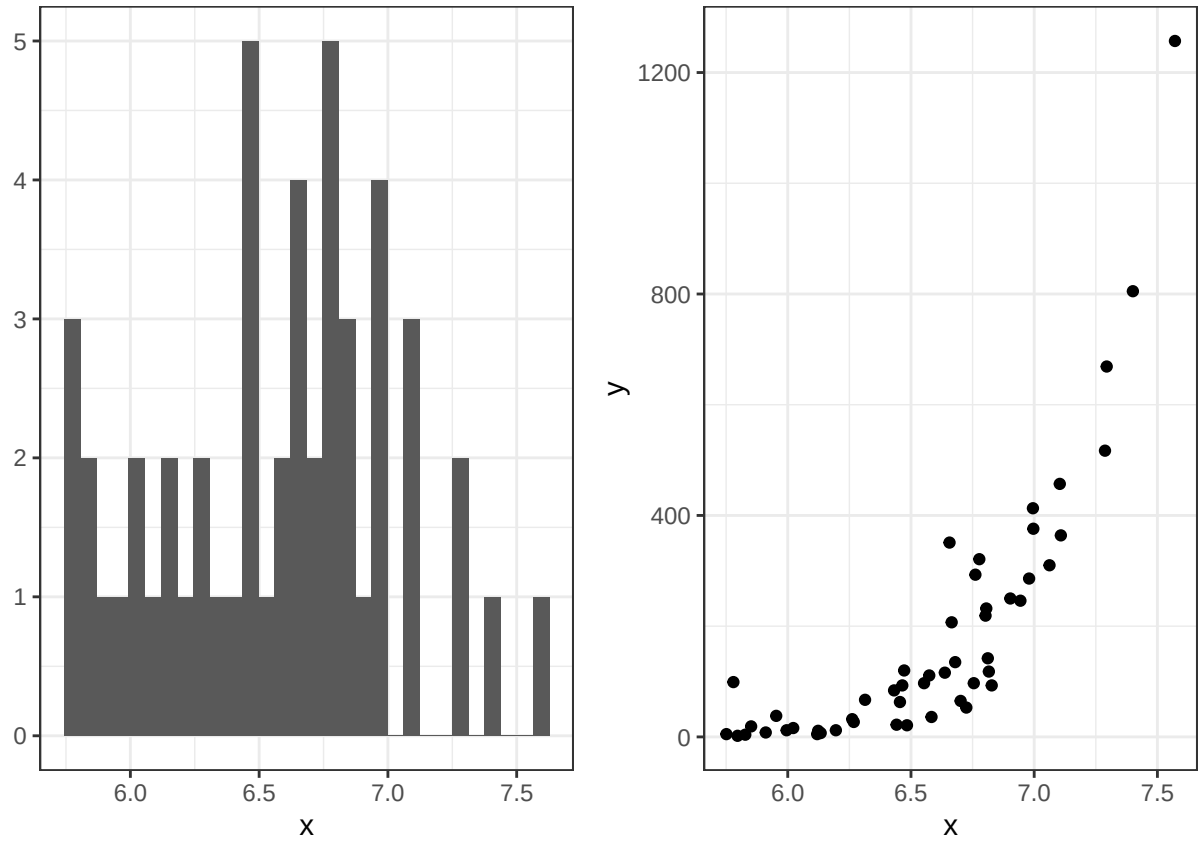
```
# Cuadrículas de gráficos

library(gridExtra)
```

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

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

```
p1 <- qplot(x)
p2 <- qplot(x,y)
grid.arrange(p1, p2, ncol = 2)
```

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



Eso es todo para el uso básico de ggplot2, una gran herramienta para visualizar datos de forma clara, organizada y compresible. :))

referencias

RPubs - *Introducción a dplyr. (2014). Rpubs.com.* https://rpubs.com/joser/dplyr/

RPubs - *Introducción a la Graficación con ggplot2.* (2018). Rpubs.com. https://rpubs.com/rdelgado/429190

RPubs - *Extensión ggthemes y ggdark.* (2023). Rpubs.com. https://rpubs.com/LizMeza/1008240#:~:text=GGThemes%20es%

Slowikowski, K. (2024. *Getting started with ggrepel.* R-Project.org. https://cran.r-project.org/web/packages/ggrepel/vignettes/ggrepel.html

Rodriguez, M. - *Manipulación de datos.* (2023). Rpubs.com. https://rpubs.com/Maus/994127

Sánchez, R (2024). *El operador pipe %>% | Programación en R.* Gitbooks.io. https://rsanchezs.gitbooks.io/rprogramming/content/chapter9/pipeline.html

IONOS. (2023). *Tipos de datos de R que deberías conocer. IONOS Digital Guide.* IONOS. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/tipos-de-datos-de-r/#:~:text=Para%20una%20comprobaci%C3%B3n%2

El paquete ggplot2. (2024). R CHARTS | Una Colección de Gráficos Hechos Con El Lenguaje de Programación R. https://r-charts.com/es/ggplot2/