



MANUAL: EL PAQUETE DE `ggplot2`

Madin Rivera, Alberto.

Najera Rodríguez, Itzel Lizbeth.

abr. 15, 2023

Índice general

INTRODUCCIÓN	3
PARTE 1: PERSONALIZACIÓN	1
CAPÍTULO 1: TÍTULO, SUBTÍTULO, CAPTION Y TAG EN <code>ggplot2</code>	3
1.1 TÍTULO	4
1.2 SUBTÍTULO	6
1.3 CAPTION	8
CAPÍTULO 2: COLOR DE FONDO EN <code>ggplot2</code>	11
2.1 GRÁFICO POR DEFECTO	11
2.2 COLOR DEL PANEL	12
2.3 COLOR DEL BORDE DE PANEL	13
2.4 COLOR DE FONDO	15
2.5 COLOR DEL BORDE DEL GRÁFICO	16
2.6 CAMBIANDO LOS COLORES CON TEMAS	17
CAPÍTULO 3: PERSONALIZACIÓN DEL GRID EN <code>ggplot2</code>	19
3.1 PERSONALIZACIÓN DE LA REJILLA	20
3.2 MAJOR GRID	21
3.3 MINOR GRID	24
3.4 NÚMERO DE LÍNEAS	27
3.5 ELIMINAR EL GRID	31

CAPÍTULO 4: ANOTACIONES DE TEXTO EN <code>ggplot2</code>	35
4.1 AGREGANDO TEXTOS CON <code>geom_text</code>	35
4.2 AGREGANDO ETIQUETAS CON <code>geom_label()</code>	39
4.3 EVITAR SOLAPAMIENTOS CON LA LIBRERÍA <code>ggrepel</code>	41
4.4 USO DE MARKDOWN Y HTML CON <code>ggtext</code>	42
CAPÍTULO 5: MÁRGENES EN <code>ggplot2</code>	45
5.1 INCREMENTAR LOS MÁRGENES	46
5.2 ELIMINAR LOS MÁRGENES	48
CAPÍTULO 6: LEYENDAS EN <code>ggplot2</code>	51
6.1 DATOS DE MUESTRA	51
6.2 AGREGANDO UNA LEYENDA	52
6.3 CAMBIAR O ELIMINAR EL TÍTULO DE LA LEYENDA	56
6.4 CAMBIAR O REORDENAR LAS ETIQUETAS DE LA LEYENDA	60
6.5 CAMBIAR LA POSICIÓN DE LA LEYENDA	62
6.6 PERSONALIZACIÓN DE LA LEYENDA	66
6.7 ELIMINAR LA LEYENDA	68
CAPÍTULO 7: SISTEMAS DE COORDENADAS EN <code>ggplot2</code>	71
7.1 COORDENADAS CARTESIANAS CON <code>coord_cartesian()</code>	71
7.2 COORDENADAS FIJAS (MISMA ESCALA) CON <code>coord_flixed()</code>	74
7.3 ROTAR LOS EJES CON <code>coord_flip()</code>	75
7.4 TRANSFORMACIONES CON <code>coord_trans()</code>	77
7.5 COORDENADAS POLARES CON <code>coord_polar()</code>	79
7.6 PROYECCIONES DE MAPAS CON <code>coord_quickmap()</code> Y <code>coord_map()</code>	81
CAPÍTULO 8: LÍNEAS DE REFERENCIA, SEGMENTOS, CURVAS Y FLECHAS EN <code>ggplot2</code>	89
8.1 LÍNEAS VERTICALES CON <code>geom_vlines()</code>	89
8.2 LÍNEAS HORIZONTALES CON <code>geom_hline</code>	91
8.2.1 LÍNEAS DIAGONALES CON <code>geom_abline()</code>	92
8.3 SEGMENTOS CON <code>geom_segment()</code>	93
8.4 CURVAS CON <code>geom_curve()</code>	95
8.5 AGREGANDO FLECHAS	97

PARTE 2: TODOS LOS GRÁFICOS DE GGPLOT2	99
CAPÍTULO 9: DIAGRAMAS DE DISTRIBUCIÓN CON <code>ggplot2</code>	101
9.1 BOX PLOT EN <code>ggplot2</code>	101
9.2 BOX PLOT POR GRUPO EN <code>ggplot2</code>	109
9.3 BOX PLOT CON PUNTOS DE DATOS EN <code>ggplot2</code>	118
9.4 HISTOGRAMA EN <code>ggplot2</code> CON EL MÉTODO DE STURGES	128
9.5 HISTOGRAMA CON DENSIDAD EN <code>ggplot2</code>	131
9.6 HISTOGRAMA POR GRUPO EN <code>ggplot2</code>	134
9.7 NÚMERO Y ANCHO DE LAS BARRAS O CLASES DE UN HISTOGRAMA EN <code>ggplot2</code>	147
9.8 DIAGRAMA DE VIOLÍN CON MEDIA EN <code>ggplot2</code>	152
9.9 GRÁFICO DE VIOLÍN CON PUNTOS EN <code>ggplot2</code>	158
9.10 GRÁFICO DE VIOLÍN POR GRUPO EN <code>ggplot2</code>	165
9.11 RIDGELINE EN <code>ggplot2</code>	184
9.12 BEESWARM EN <code>ggplot2</code> CON EL PAQUETE <code>ggbeeswarm</code>	200
CAPÍTULO 10: DIAGRAMAS DE CORRELACIÓN CON <code>ggplot2</code>	207
10.1 GRÁFICO DE DISPERSIÓN POR GRUPO EN <code>ggplot2</code>	207
10.2 GRÁFICO DE DIPERSIÓN EN <code>ggplot2</code>	216
10.3 GRÁFICO DE DIPERSIÓN CON HISTOGRAMAS MARGINALES EN <code>ggplot2</code>	223
10.4 GRÁFICO DE DISPERSIÓN CONECTADO EN <code>ggplot2</code>	233
10.5 GRÁFICO DE DISPERSIÓN CON ELIPSSES EN <code>ggplot2</code>	238
10.6 GRÁFICO HEXBIN EN <code>ggplot2</code>	250
10.7 CONTOURS EN <code>ggplot2</code>	260
CAPÍTULO 11: DIAGRAMAS DE EVOLUCIÓN CON <code>ggplot2</code>	273
11.1 STREAMGRAPH CON <code>ggplot2</code>	273
11.2 DIAGRAMAS DE ÁREAS (AREA CHART) EN <code>ggplot2</code> CON <code>geom_area()</code>	287
11.3 DIAGRAMA DE MÚLTIPLES LÍNEAS EN <code>ggplot2</code>	292
11.4 FUNCIONES CON <code>ggplot2</code>	302
11.5 DIAGRAMA DE LÍNEAS EN <code>ggplot2</code>	309

CAPÍTULO 12 PARTE DE UN TODO	317
14.1 DIAGRAMA DE BARRAS APILADAS EN <code>ggplot2</code>	317
14.2 DIAGRAMA DE SECTORES CON PORCENTAJES EN <code>ggplot2</code>	328
14.3 DIAGRAMA VORONOI EN <code>ggplot2</code> CON EL PAQUETE <code>ggvoronoi</code>	333
14.4 DIAGRAMA DE WAFFLE (SQUARE PIE) EN <code>ggplot2</code>	337
15. CALENDARIO LUNAR CON <code>ggplot2</code>	345
15.1 COLOR DE LAS LUNAS	346
15.2 TAMAÑO DE LAS LUNAS	346
16. BIBLIOGRAFÍA	349

Dedicado para Itzel L.

N.R.

Para mi Familia

M.R

Para mis compañeros y amigos de Universidad

Para mis compañeros de trabajo

Para todos los colegas que se

encuentran aprendiendo

a crear diagramas

INTRODUCCIÓN¹

¹Ramón Berrendero, José. (s/n), *"Una breve introducción a ggplot2"*. Universidad Autónoma Metropolitana. Disponible en: <http://verso.mat.uam.es/~joser.berrendero/R/introggplot2.html>

El paquete `ggplot2` es la alternativa más popular a los gráficos de R base. Se basa en el Grammar of Graphics y su mayor ventaja es su flexibilidad, ya que permite crear y personalizar gráficos agregando capas. Con esta librería se puede crear gráficos de gran calidad de manera muy sencilla.

La librería `ggplot2` de R es un sistema organizado de visualización de datos. Forma parte del conjunto de librerías llamado `tidyverse`. El objetivo de este manual es introducir el uso, principalmente a través de ejemplos.

Los elementos necesarios para representar un diagrama en `ggplot2` son los siguientes:

1. Un *data frame* que contenga los datos que se quieren visualizar.
2. Los *aesthetics*, es decir, una lista de relaciones entre las variables del conjunto de datos y determinados aspectos del gráfico (como por ejemplo: coordenadas, formas, colores, entre otros más).
3. Los *geometries*, que especifican los elementos geométricos (es decir: puntos, líneas, círculos, mapas, entre muchos más) que se van a representar.

Normalmente estos elementos se van añadiendo de forma consecutiva en distintas capas (*layers*). Para añadir una nueva capa se usa el signo `+`. La estructura general del código para obtener un diagrama es como se muestra en la siguiente línea de código:

```
ggplot(data = "nombre del fichero de datos") +  
  geom_nombre1(aes(aesthetics_1 = variable_1,  
                    aesthetics_2 = variable_2,  
                    ...,  
                    aesthetics_n = variable_n)) +  
  geom_nombre2(...)
```

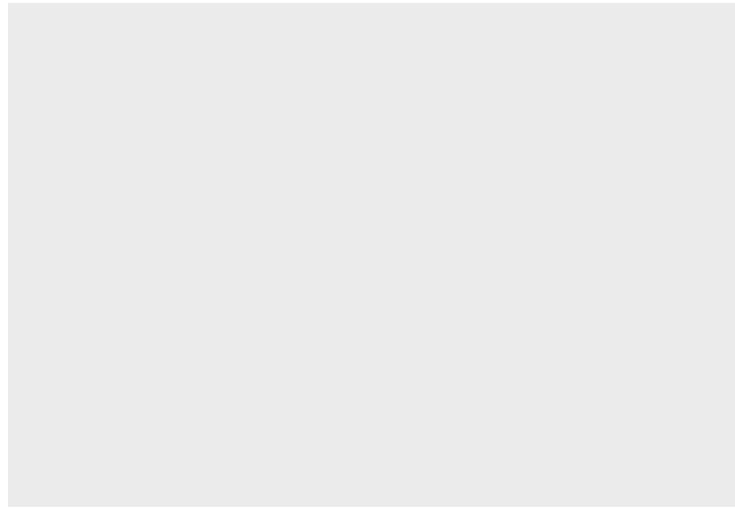
La función `ggplot()` se usa para generar el sistema de coordenadas (por defecto, rectangulares) y posteriormente se añaden los `geom_` con sus correspondientes `aesthetics`. En principio, los `aesthetics` se pueden asignar individualmente para cada `geom_`.

Como un ejemplo introductorio, se utilizará el conjunto de datos `notas` que contiene las notas medias en 1000 colegios de una prueba de nivel que realizó la Comunidad de Madrid en 2009 y 2010, junto con el tipo de colegio (concertado, privado o público). Lo primero es cargar `ggplot2` y leer los datos.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
notas = read.table('http://verso.mat.uam.es/~joser.berrendero/datos/notas.txt',  
                   sep = ' ',  
                   dec = ',',  
                   header=TRUE)
```

Para crear un gráfico se usa la función `ggplot()`

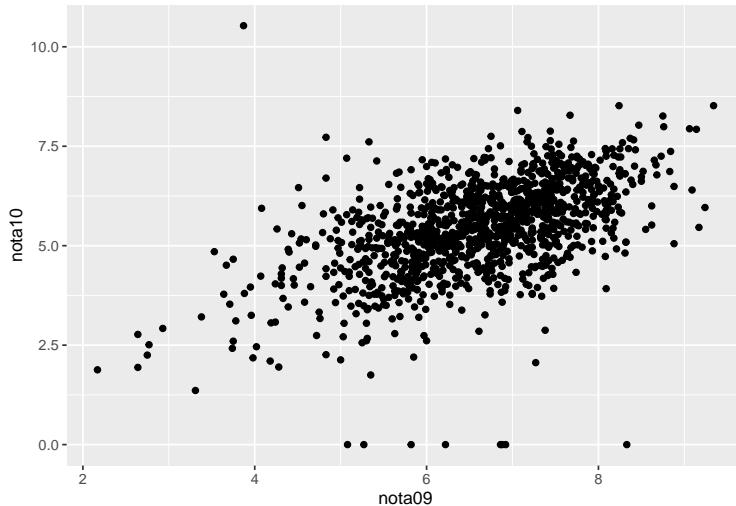
```
ggplot(data = notas)
```



De momento solo se ha asignado al gráfico el conjunto de datos que se quiere visualizar. No se representa gráficamente nada hasta que no se añadan más capas.²

Las capas sirven para proporcionar información sobre cómo se quiere visualizar los datos. Esto se lleva a cabo a través de una función `geom_`. Este es un ejemplo sencillo usando la función `geom_point()` que es el correspondiente a un diagrama de dispersión (cada tipo de gráfico tiene el suyo). En primer lugar, mediante la función `aes()` se asignan las coordenadas `x` e `y` a los valores de las notas de 2009 y 2010, respectivamente.

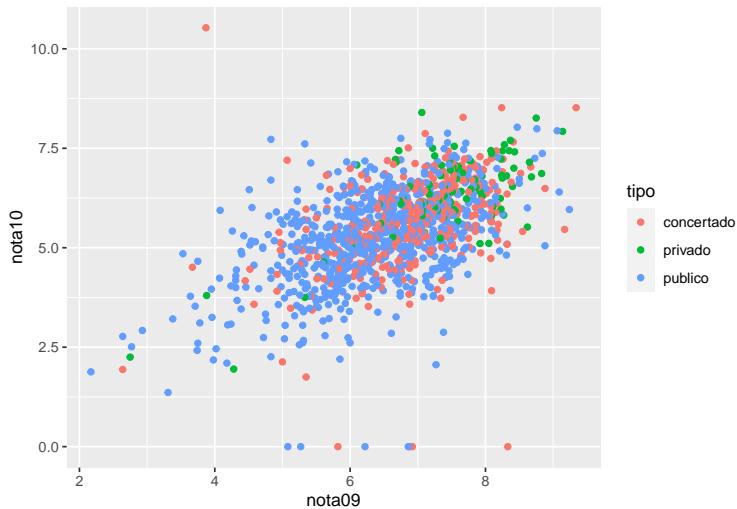
```
ggplot(data = notas) +  
  geom_point(aes(x = nota09, y = nota10))
```



²Los datos a representar siempre tienen que formar parte de un `data frame`.

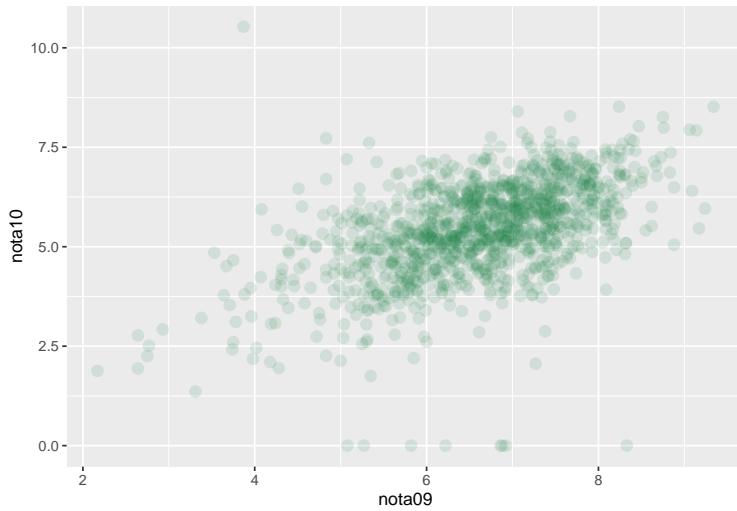
Si se desea añadir información sobre el tipo de colegio, se puede especificar el color en el que se representa el punto que dependa de la variable **tipo** con el argumento **col**.

```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10, col = tipo))
```



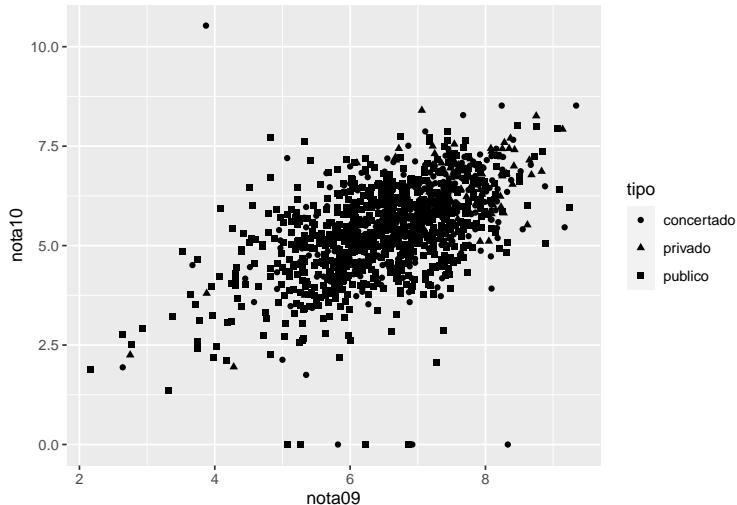
Se observa cómo la función **aes()** (que corresponde a *aesthetics*) asigna variables a ciertos aspectos del gráfico. Estos aspectos se pueden fijar en lugar de hacerlos depender de una variable. En este caso, el valor del argumento se asigna fuera de la lista de *aesthetics*. En el siguiente ejemplo se fija el color de los puntos, su tamaño, y el grado de transparencia (*lo cual es útil cuando hay demasiados datos a visualizar*).

```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",      # color de los puntos
             size = 3,              # tamaño de los puntos
             alpha = 1/8)           # nivel de transparencia de los puntos
```



En general, cada `geom_` admitirá un conjunto determinado de *aesthetics*. En el caso particular de `geom_point()`, mediante la función `aes()` se puede que una variable dependa de otros aspectos como la forma o el tamaño de los puntos.

```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10, shape= tipo))
```



```
# una forma para cada tipo de colegio
```

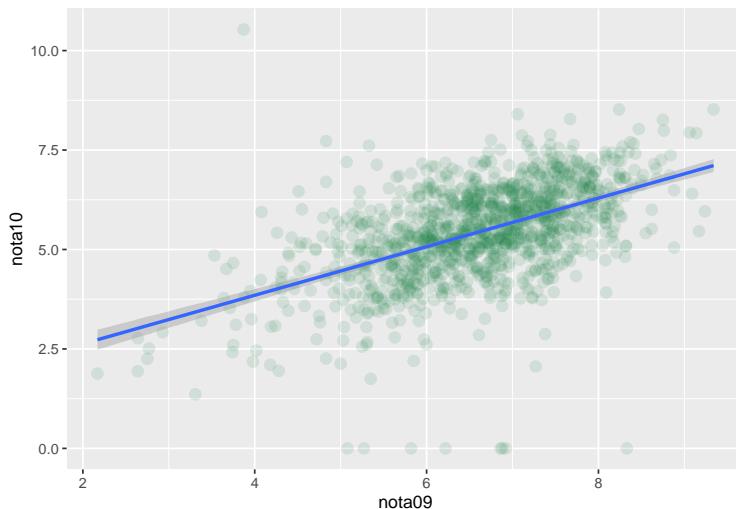
Es necesario **consultar la página de `ggplot2`**³ para saber qué `aesthetic` admite cada `geom_` en particular.

Algunos `geom_` requieren llevar a cabo transformación estadística (técticamente, un `stat`) que toma un conjunto de datos y crea otro nuevo. De esta forma, combinando `geom_` y

³Se puede acceder al enlace al dar click en las letras en negritas.

stat, se pueden obtener una gran variedad de visualizaciones. A veces se puede asignar una transformación estadística a un **geom_** mediante el argumento **method**. Por ejemplo, la función **geom_smooth()** usa distintos métodos para estimar la media condicionada (*curva de regresión lineal*) de una variable por otra, incluyendo el método de mínimos cuadrados ordinarios, que es el más conocido:

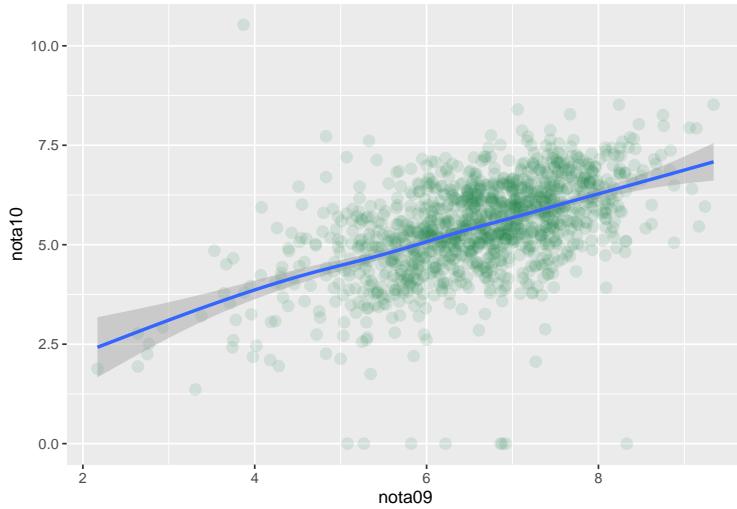
```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10), method = "lm")
```



```
# Añade la recta de mínimos cuadrados
```

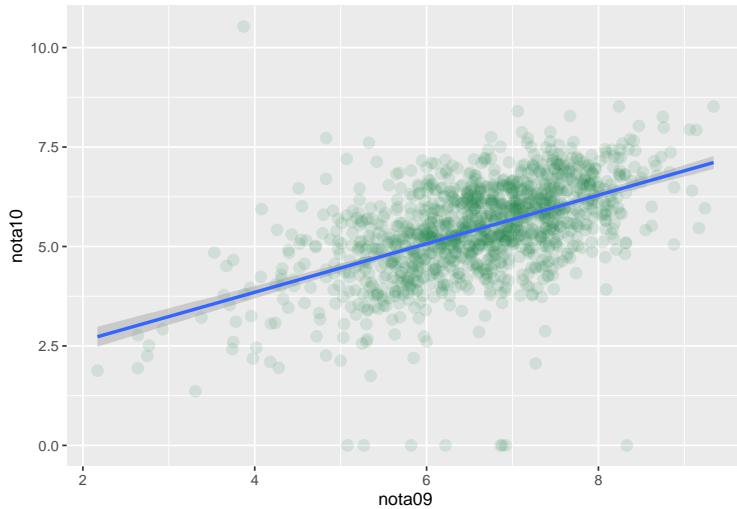
La opción por defecto no es la recta de mínimos cuadrados ordinarios, sino un método de suavizado no paramétrico:

```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10)) # Opción por defecto
```



La asignación de las coordenadas `x` e `y` a las variables podría también haberse hecho de una vez en la primera línea (pero el código tiene la ventaja de que permite hacer una asignación de variables diferentes en cada `geom_` en vez de mantener la asignación fija para todos ellos):

```
ggplot(data = notas, aes(x = nota09, y = nota10)) +  
  # Esta asignación se fija para todos los geoms  
  geom_point(col = "#1D8348", size = 3, alpha = 1/8) +  
  geom_smooth(method = "lm")
```



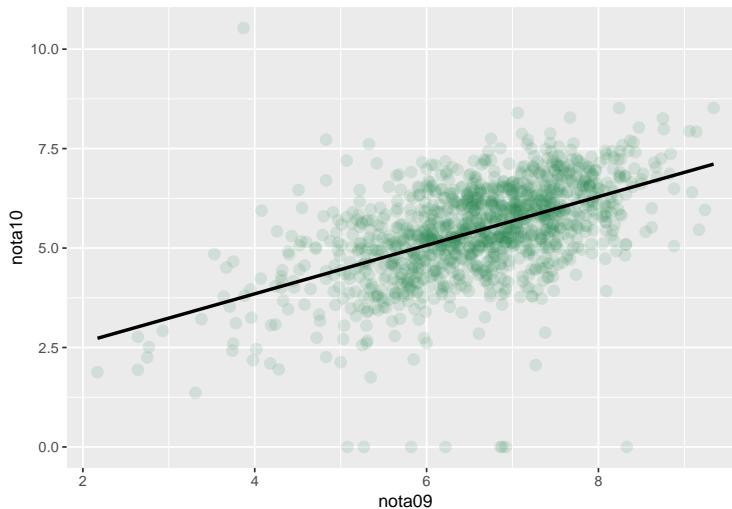
Se observa que se representan unas bandas de confianza. En el siguiente ejemplo se eliminarán y, además, se modificarán otros aspectos estéticos del diagrama:

```
ggplot(data = notas) +  
  geom_point(aes(x = nota09, y = nota10),
```

```

            col = "#1D8348",
            size = 3,
            alpha = 1/8) +
geom_smooth(aes(x = nota09, y = nota10),
            method = "lm",
            col = "black",
            # Se representa la recta en color negro
            se = FALSE)

```



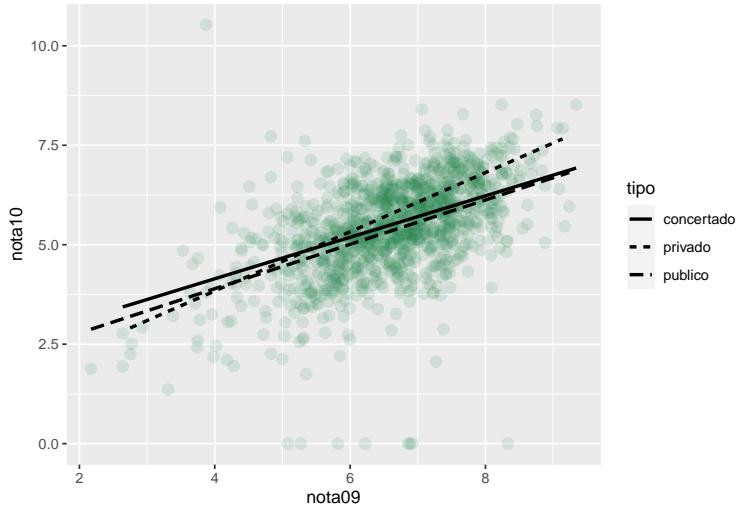
```
# Se eliminan las bandas de confianza (se=standard error)
```

Si se asigna el tipo de colegio por línea, se obtienen las rectas de mínimos cuadrados para los tres colegios:

```

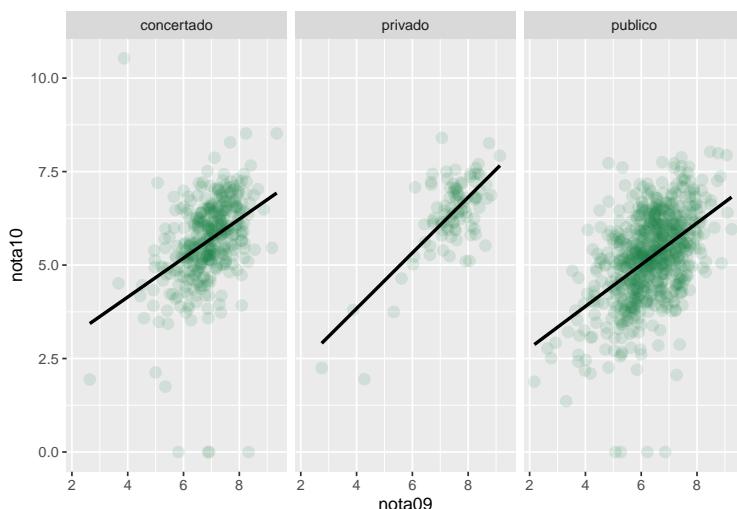
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10, linetype = tipo),
              # calcula una recta para cada tipo
              method = 'lm',
              col = 'black',
              se=FALSE)

```



En ocasiones, puede ser conveniente representar un gráfico para cada uno de los tipos de colegio por separado. Los `facet_`, otro elemento importante de `ggplot2`, determina que se debe de representar un diagrama condicionado a cada uno de los valores de alguna de las variables. Se añade de la siguiente manera.

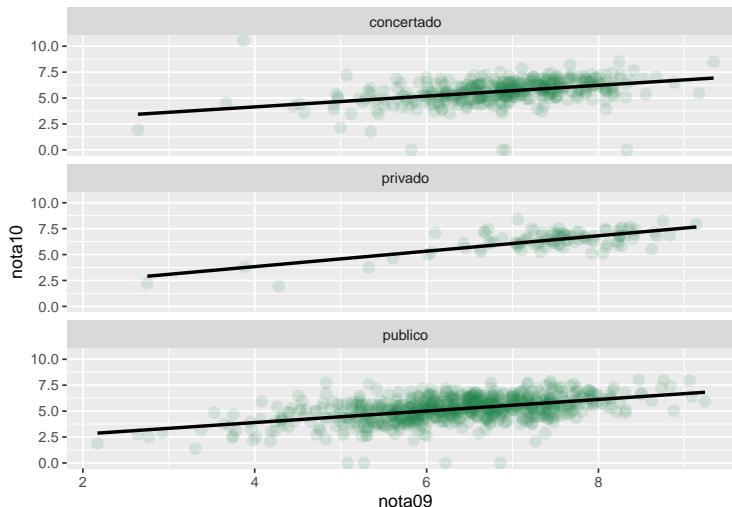
```
ggplot(data = notas) +  
  geom_point(aes(x = nota09, y = nota10),  
             col = "#1D8348",  
             size = 3,  
             alpha = 1/8) +  
  geom_smooth(aes(x = nota09, y = nota10),  
              method = "lm",  
              col = "black",  
              se=FALSE) +  
  facet_wrap(~ tipo) # un gráfico para cada tipo de colegio
```



El paquete `ggplot2` permite personalizar los gráficos con temas. Es posible personalizar cada detalle de un gráfico, como los colores, tipos de líneas, fuentes o alineación, entre otros, usando los componentes de la función `theme()`. Además, la librería proporciona otras funciones que permiten añadir títulos, subtítulos, líneas, flechas o texto.

Como se puede ver, la función `facet_wrap()` utiliza una fórmula análoga a las utilizadas en general para ajustar modelos en R. Uno puede comparar lo anterior con el siguiente diagrama y su línea de código:

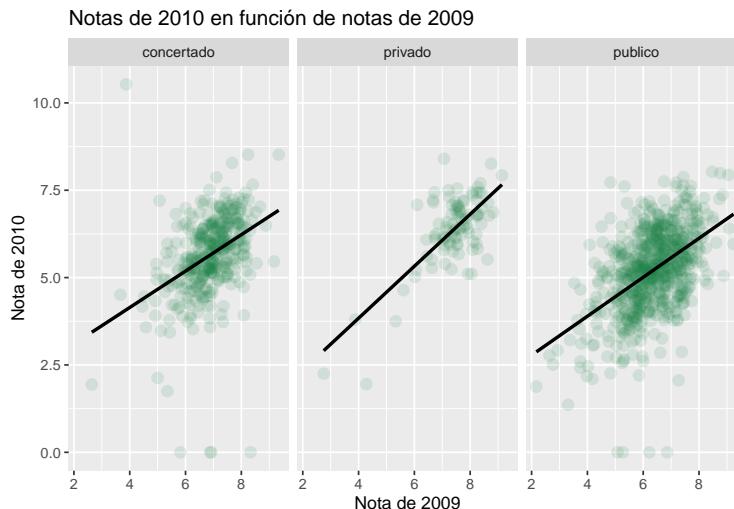
```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10),
              method = "lm",
              col = "black",
              se=FALSE) +
  facet_wrap(~ tipo, nrow = 3)
```



Se puede añadir un título o cambiar las etiquetas de los ejes añadiendo una nueva capa con la función `labs()`:

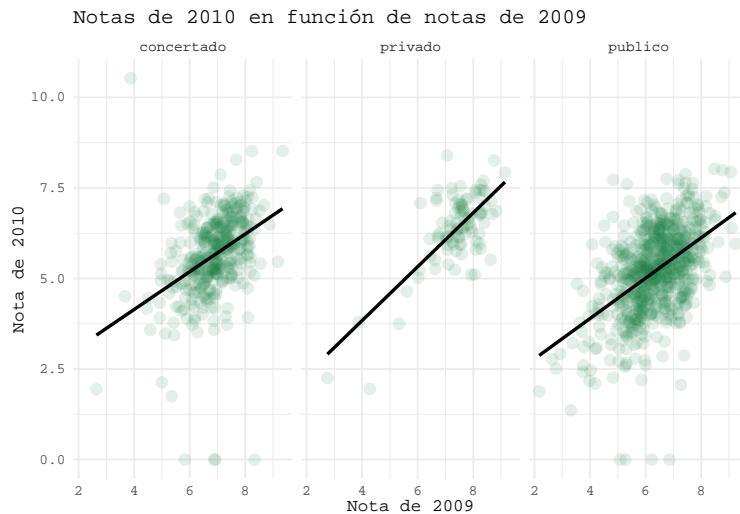
```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10),
              method = "lm",
```

```
        col = "black",
        se = FALSE) +
facet_wrap(~tipo) +
labs(title = "Notas de 2010 en función de notas de 2009",
x = "Nota de 2009",
y = "Nota de 2010")
```



Se han usado estilos de gráficos por defecto en todos los códigos anteriores, con un característico fondo gris que no puede ser del agrado de todo el mundo. Se puede cambiar añadiendo una nueva capa. En el siguiente ejemplo se usará la función `theme_minimal()`. También se usarán argumentos opcionales que pueden llegar a cambiar el tipo de letra.

```
ggplot(data = notas) +
  geom_point(aes(x = nota09, y = nota10),
             col = "#1D8348",
             size = 3,
             alpha = 1/8) +
  geom_smooth(aes(x = nota09, y = nota10),
              method = "lm",
              col = "black",
              se = FALSE) +
  facet_wrap(~ tipo) +
  labs(title = "Notas de 2010 en función de notas de 2009",
       x = "Nota de 2009",
       y = "Nota de 2010") +
  theme_minimal(base_family = "mono")
```



```
# base_family es un argumento (opcional) para cambiar el tipo de letra
```

Hay **muchos otros temas⁴** que se pueden usar y personalizar dentro de `ggplot2`.

⁴Se puede acceder al enlace al dar click en las letras en negritas.

PARTE 1: PERSONALIZACIÓN

En esta primera sección se verá todo lo básico sobre el paquete `ggplot2`. `ggplot2` permite personalizar los gráficos con temas: Es posible personalizar cada detalle de un diagrama, como colores, tipos de líneas, fuentes o alineación, entre otros, usando los componentes de la función `theme()`. Además, la librería proporciona otras funciones que permiten añadir títulos, subtítulos, líneas, flechas, textos, entre muchos otros más.

CAPÍTULO 1: TÍTULO, SUBTÍTULO, CAPTION Y TAG EN ggplot2



Cuando se crean gráficos con ggplot2 se puede agregar un título, un subtítulo, un caption (descripción) o un tag (una etiqueta, por ejemplo para indicar el número de la figura). Existen dos formas de añadir títulos:

- Usando la función `ggtitle()`.
- Usando la función `labs()`.

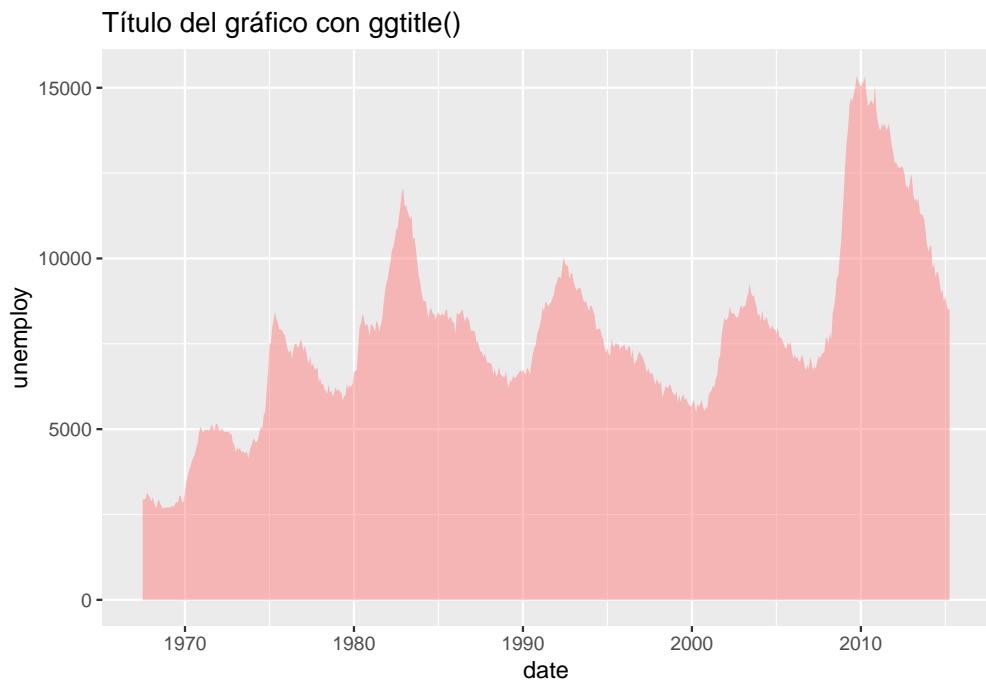
La primera solo para títulos y subtítulos, mientras que la segunda tambien permite agregar captions y tags.

1.1 TÍTULO

1.1.1 OPCIÓN 1. USAR `ggttitle()`

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  ggttitle("Título del gráfico con ggttitle()")
```

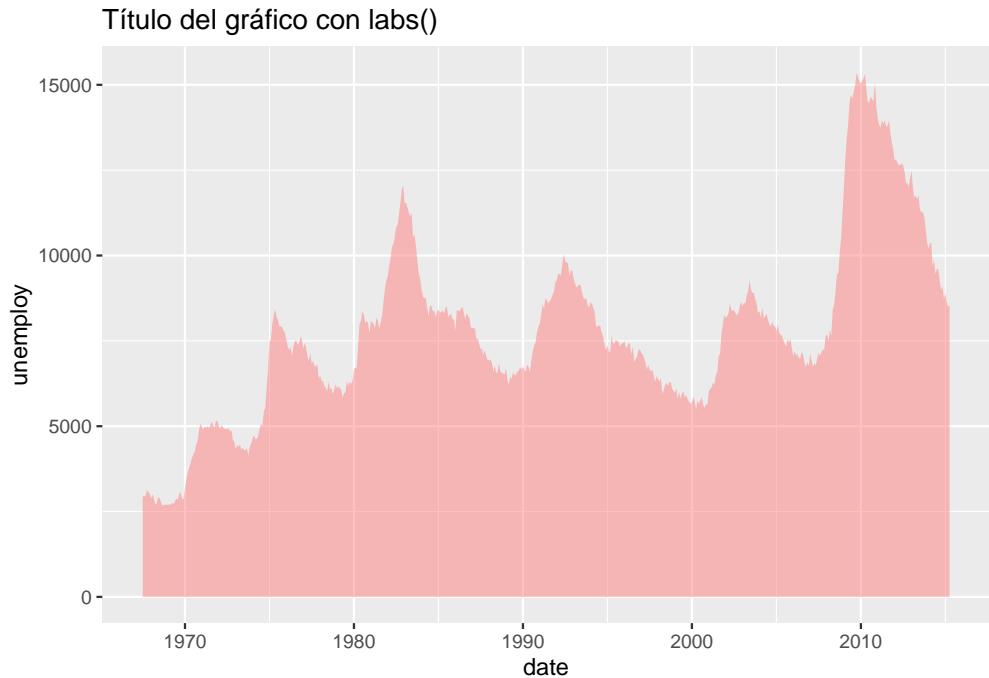


1.1.2 OPCIÓN 2. USAR `labs()`

También se puede usar la función `labs()` para añadir el título.

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  labs(title = "Título del gráfico con labs()")
```



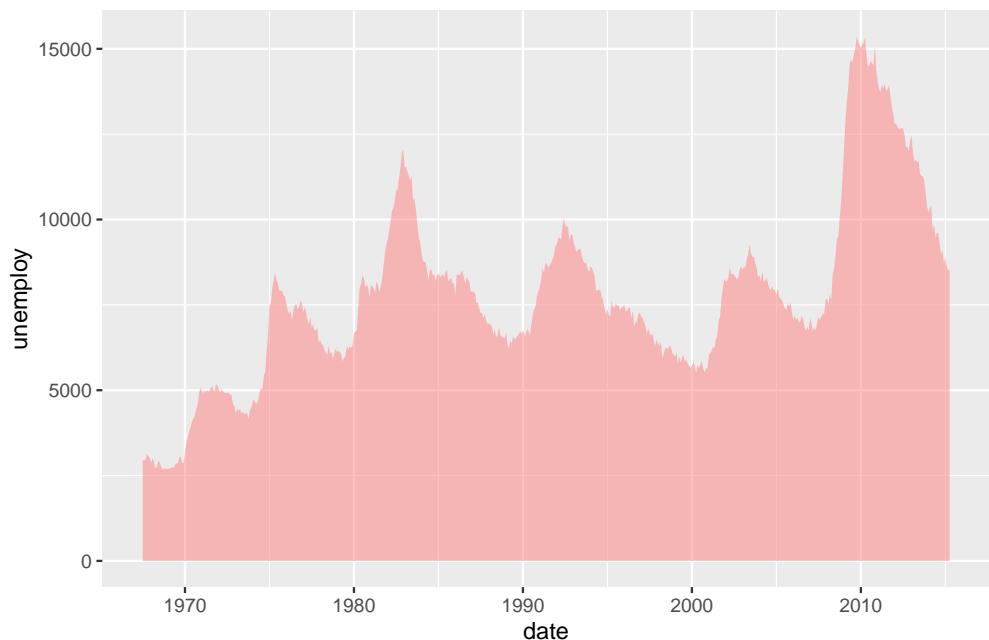
1.1.3 POSICIÓN DEL TÍTULO

La posición del título se puede establecer con respecto a todo el gráfico en lugar de solo el panel con el componente `plot.title.position` de la función `theme()`. El valor por defecto es “**panel**”. *Esto también se aplica al subtítulo.*

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  labs(title = "Título en el margen del gráfico") +
  theme(plot.title.position = "plot")
```

Título en el margen del gráfico



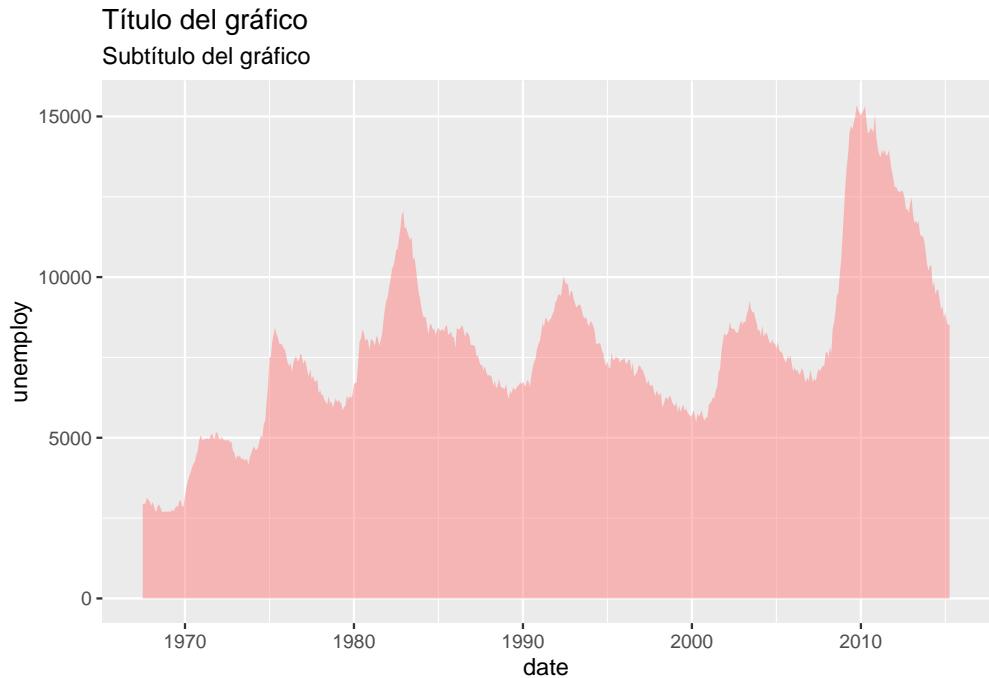
1.2 SUBTÍTULO

Se puede agregar un subtítulo de la misma manera que se añade el título, pero usando el argumento `subtitle`.

1.2.1 OPCIÓN 1. USAR `ggttitle()`

```
# install.packages(ggplot2)
library(ggplot2)

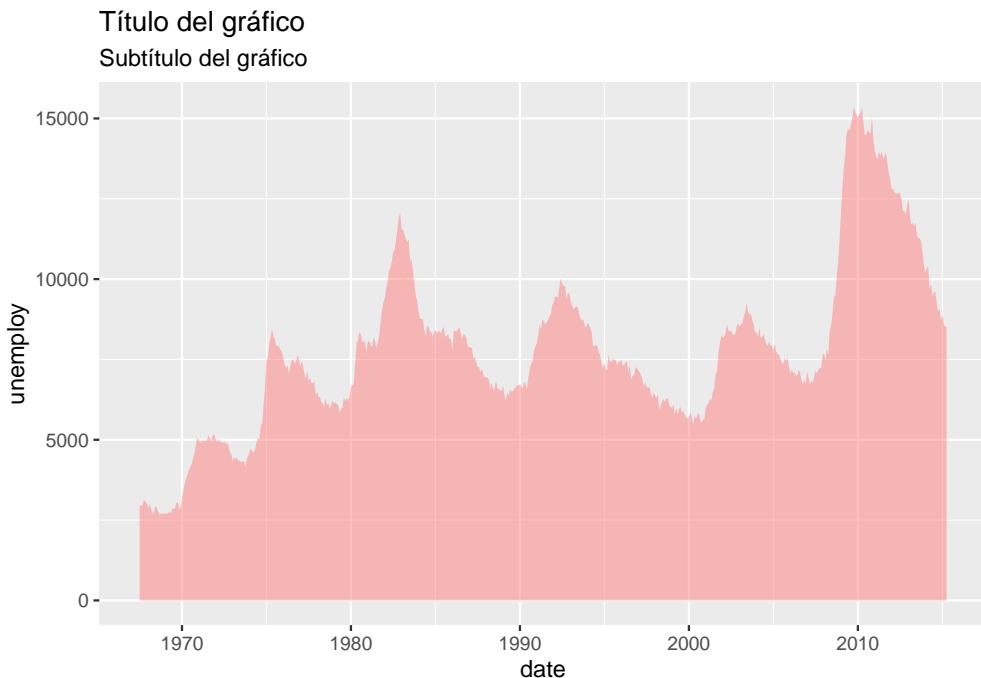
ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  ggttitle("Título del gráfico",
           subtitle = "Subtítulo del gráfico")
```



1.2.2 OPCIÓN 2. USAR labs()

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  labs(title = "Título del gráfico",
       subtitle = "Subtítulo del gráfico")
```

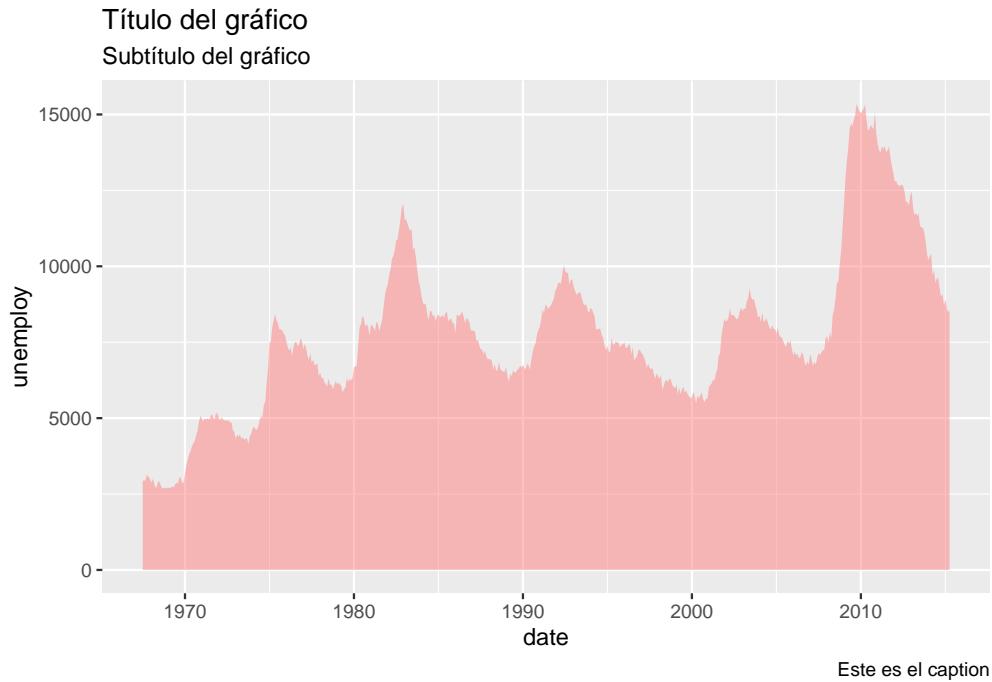


1.3 CAPTION

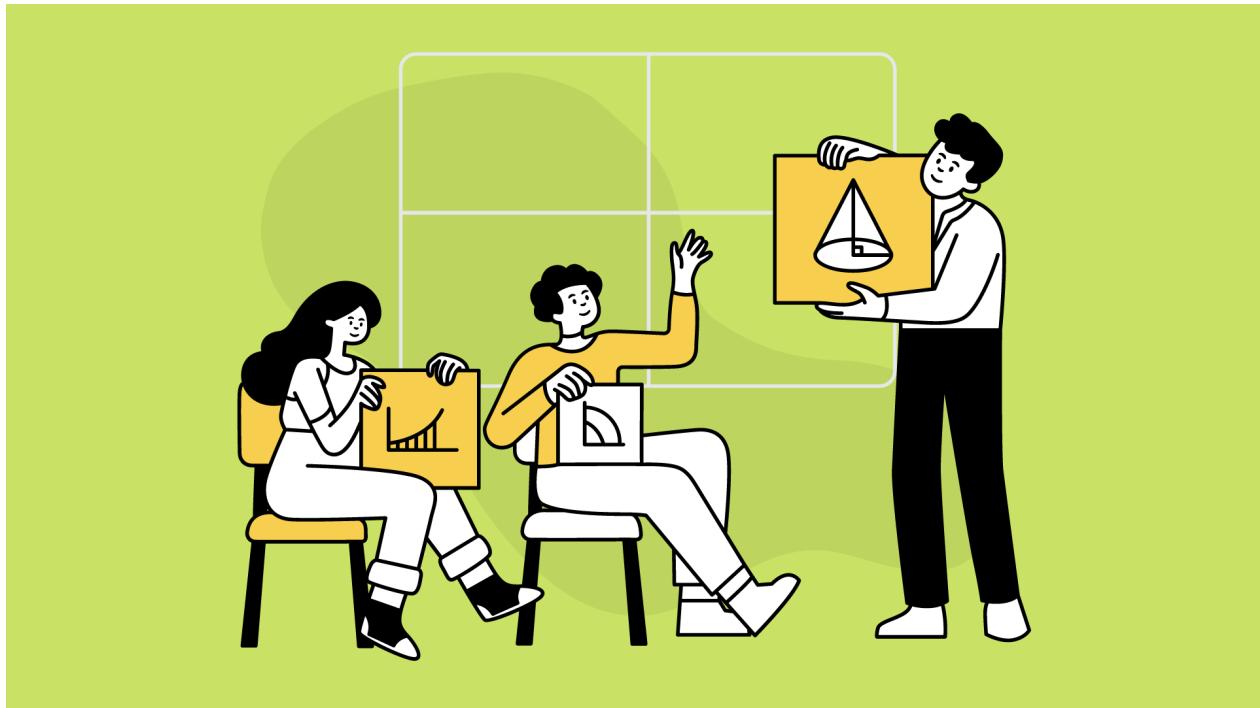
Un caption se puede usar para describir la figura. Se puede agregar con el argumento `caption` de la función `labs`.

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(1, 0.5, 0.5, alpha = 0.5)) +
  labs(title = "Título del gráfico",
       subtitle = "Subtítulo del gráfico",
       caption = "Este es el caption")
```



CAPÍTULO 2: COLOR DE FONDO EN ggplot2



2.1 GRÁFICO POR DEFECTO

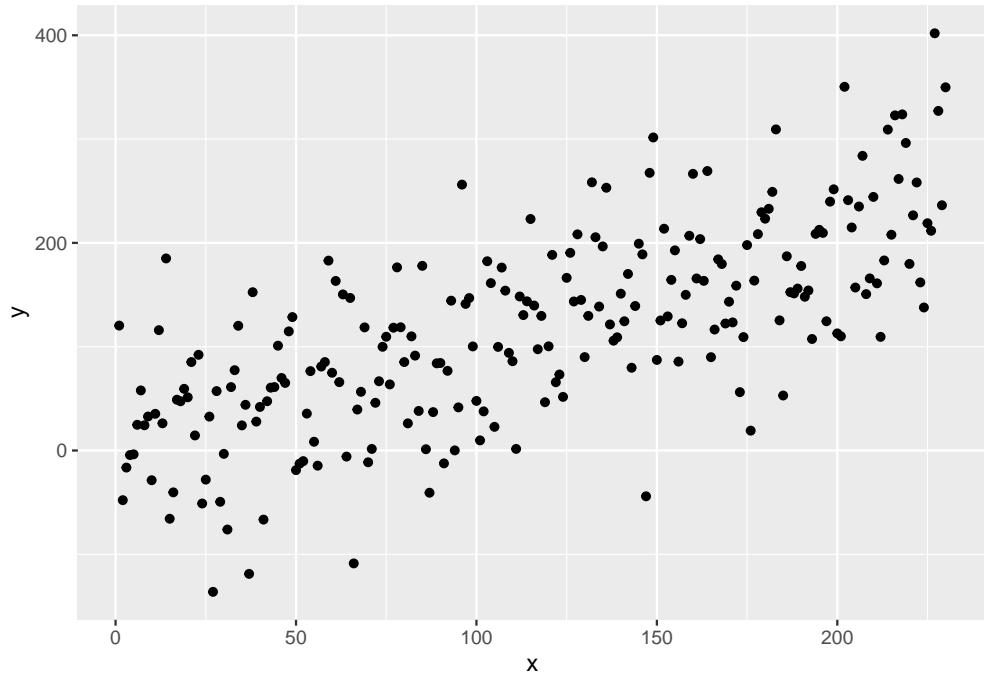
Por defecto, los gráficos de ggplot2 tienen un panel gris y un fondo blanco.

```
library(ggplot2)

# Datos de muestra
set.seed(321)
x = 1:230
y = x + rnorm(230, sd = 70)
```

```
df = data.frame(x = x, y = y)

# Gráfico
ggplot(data = df, aes(x = x, y = y)) +
  geom_point()
```

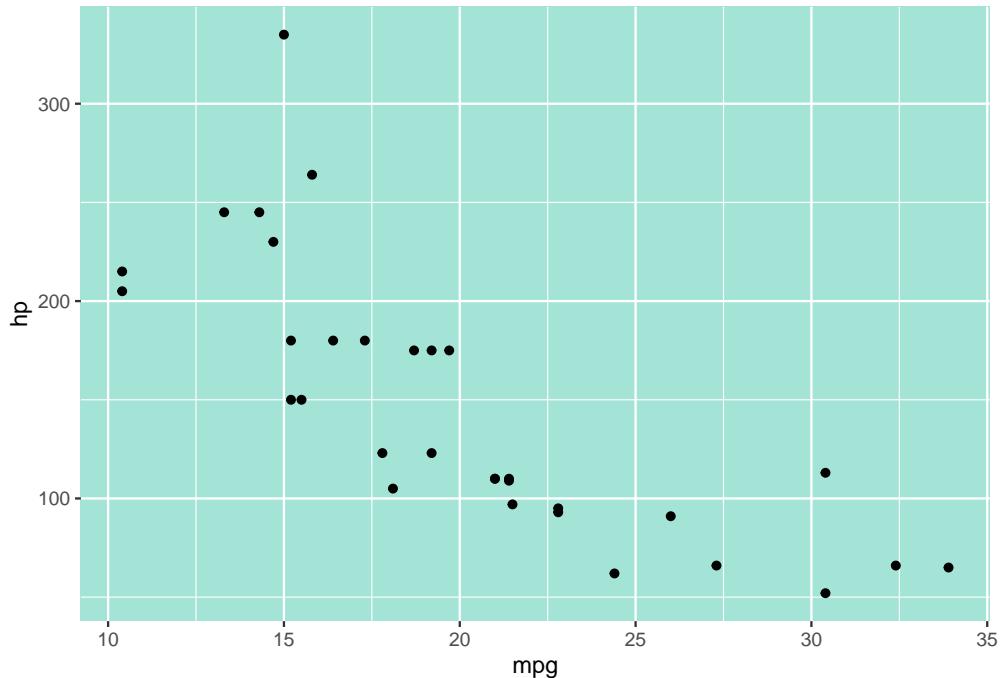


2.2 COLOR DEL PANEL

Se puede cambiar el color de fondo del panel estableciendo un `element_rect` en el componente ‘`panel.background`’ de la función `theme()` de la siguiente manera.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = mpg, y = hp)) +
  geom_point() +
  theme(panel.background = element_rect(fill = "#A3E4D7"))
```



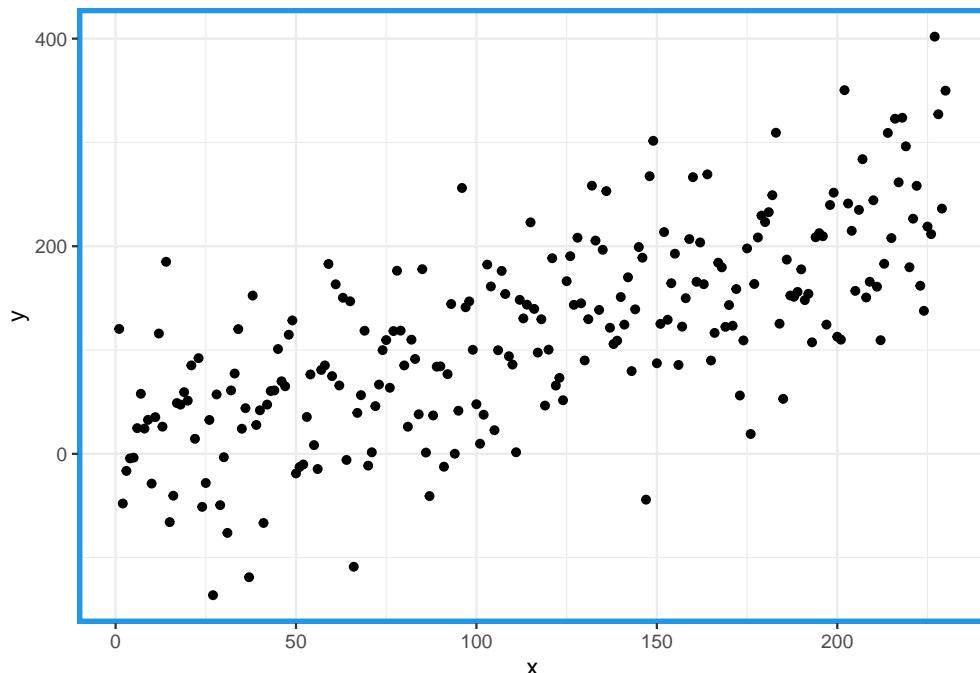
2.3 COLOR DEL BORDE DE PANEL

2.3.1 OPCIÓN 1

El componente `panel.border` de la función ‘`theme()`’ controla el color y el ancho del borde del panel con los argumentos `color` y `size`. Sin embargo, se tendrá que establecer `fill = "transparent"` para evitar ocultar los datos.

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_bw() +
  theme(panel.border = element_rect(fill = "transparent",
                                    # Necesario para agregar el borde
                                    color = 4,
                                    # Color del borde
                                    size = 2))
```



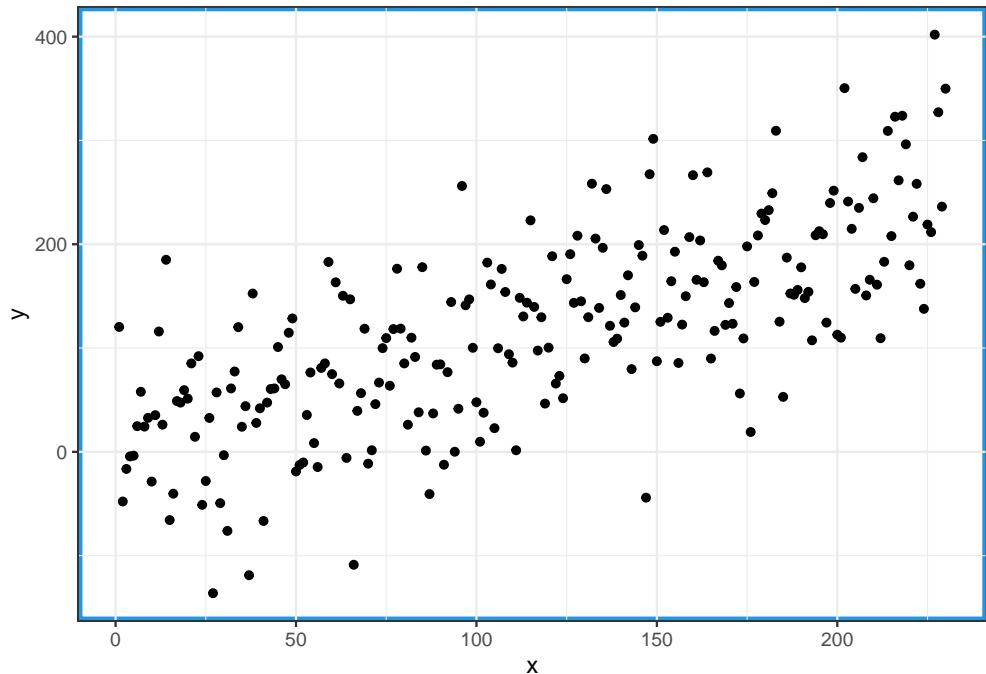
```
# Ancho del borde
```

2.3.2 OPCIÓN 2

También se puede establecer un `element_rect` para el componente `panel.background` y modificar el color del borde con el argumento `color`. Sin embargo **esto no es lo recomendable, ya que no se sobrescribe el color del borde actual**. Puede uno comprobarlo con `theme_bw` (observando que el borde negro está detrás del azul).

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_bw() +
  theme(panel.background = element_rect(color = 4, # Color del borde
                                         size = 2)) # Ancho del borde
```

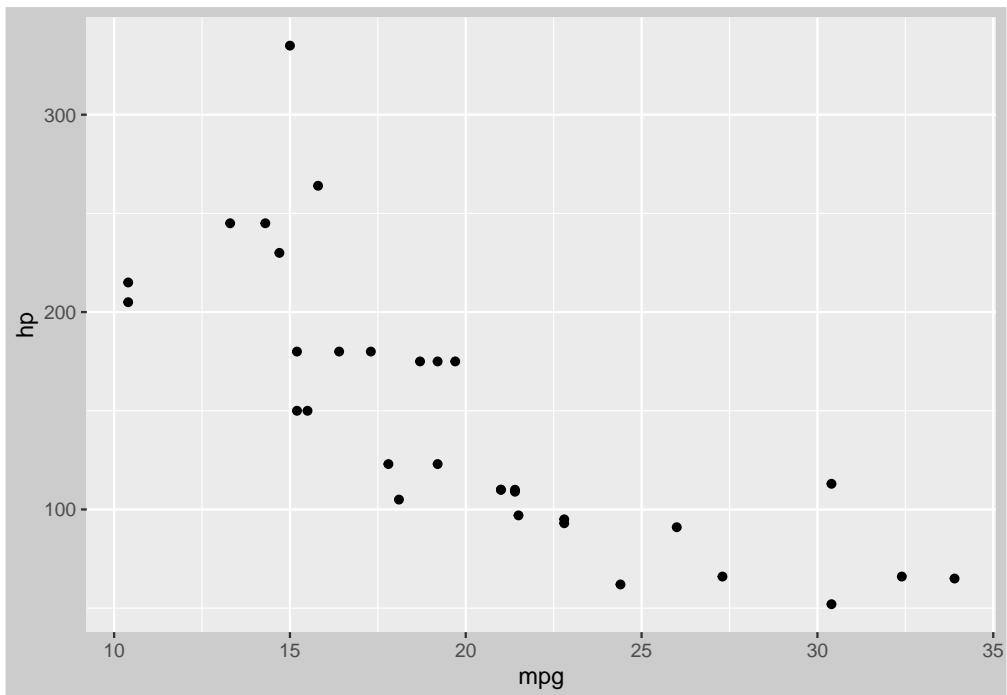


2.4 COLOR DE FONDO

El componente `plot.background` de la función `theme()` permite modificar el color de fondo de la figura. Establece el color dentro del argumento `fill` de `element_rect()`.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = mpg, y = hp)) +
  geom_point() +
  theme(plot.background = element_rect(fill = "gray80")) # Color de
```



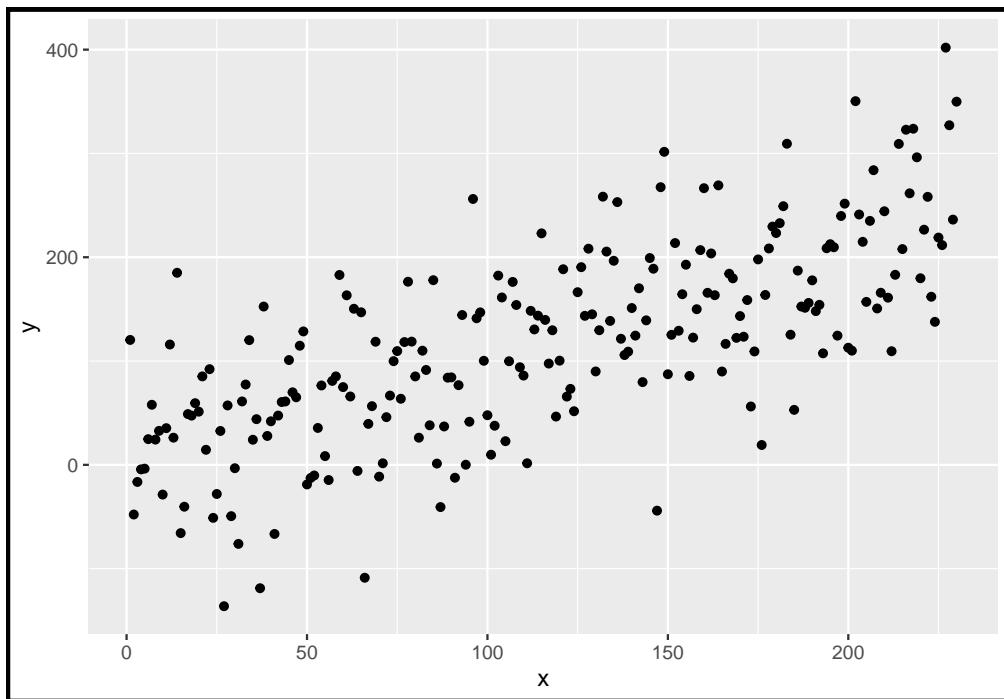
```
# fondo de la figura
```

2.5 COLOR DEL BORDE DEL GRÁFICO

También se puede establecer un color para el borde de toda la figura. Para ello se pasa un `element_rect()` al componente `plot.background` de la función `theme()` y modifica el color y el ancho del borde con los argumentos `color` y `size`, respectivamente.

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme(plot.background = element_rect(color = "black", # Color
                                         size = 1.7))      # Ancho
```



2.6 CAMBIANDO LOS COLORES CON TEMAS

Vale la pena mencionar que existen demasiados temas personalizados de `ggplot2` disponibles dentro del mismo paquete⁵ que proporcionan distintos colores de fondo.

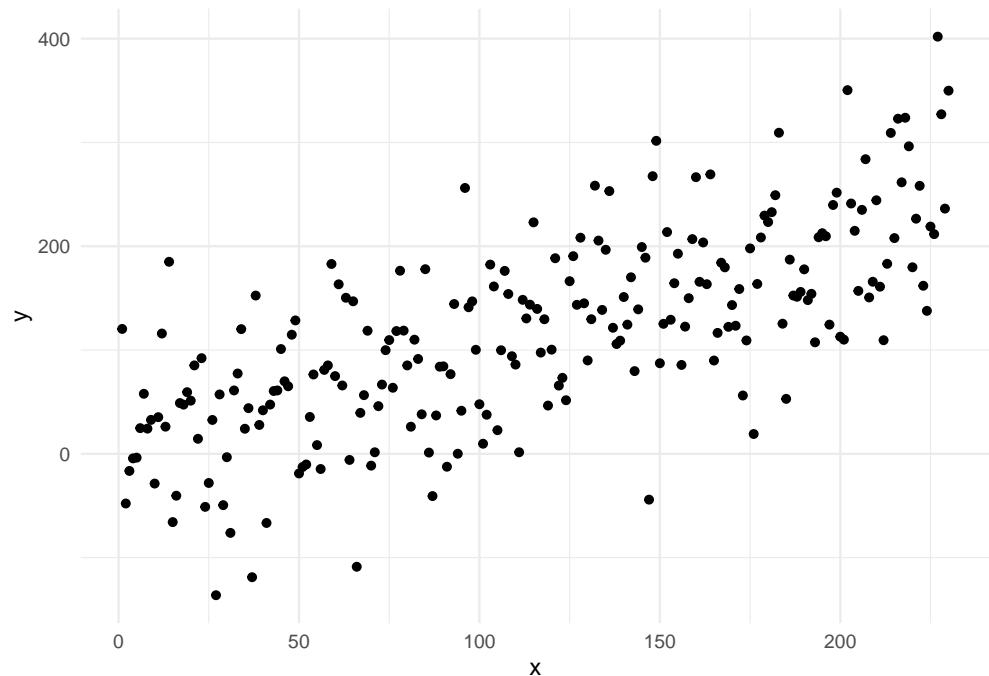
En este ejemplo hemos seleccionado el tema `theme_minimal()`, que es uno de los temas preferidos de `ggplot2`.⁶

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_minimal()
```

⁵Los temas se encuentran al dar click en las líneas de código de lectura de la palabra `ggplot2`.

⁶Hay que tener en cuenta que el tema por defecto es `theme_grey()` o `theme_gray()`.

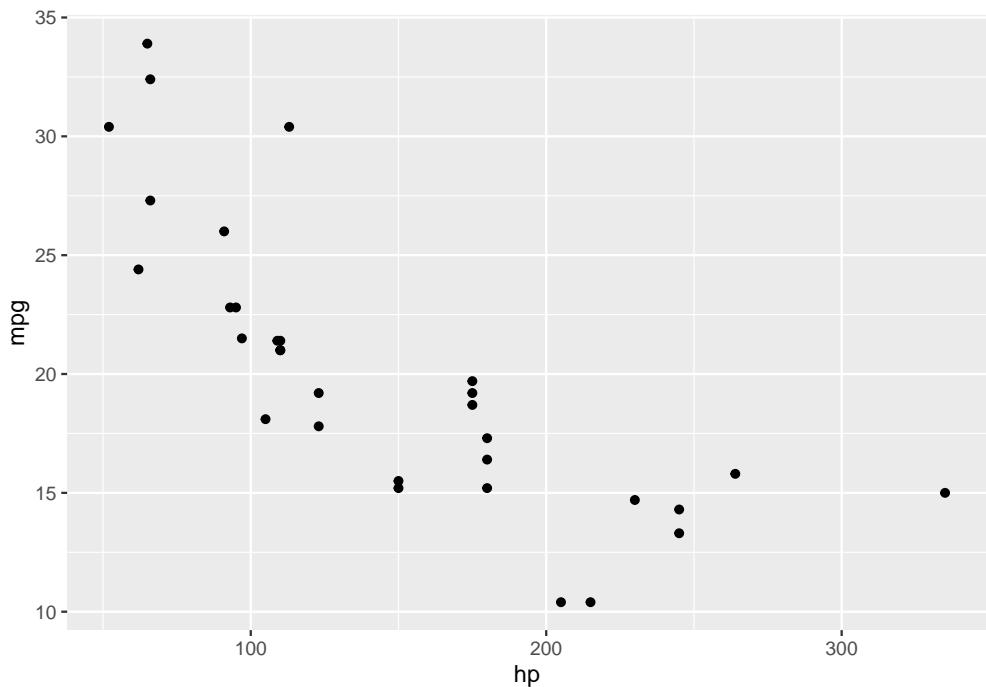


CAPÍTULO 3: PERSONALIZACIÓN DEL GRID EN `ggplot2`



Por defecto, `ggplot2` crea un grid principal (major) y otro secundario (minor) blanco como el de la siguiente figura.

```
library(ggplot2)
ggplot(data = mtcars, aes(x = hp, y = mpg))+
  geom_point()
```

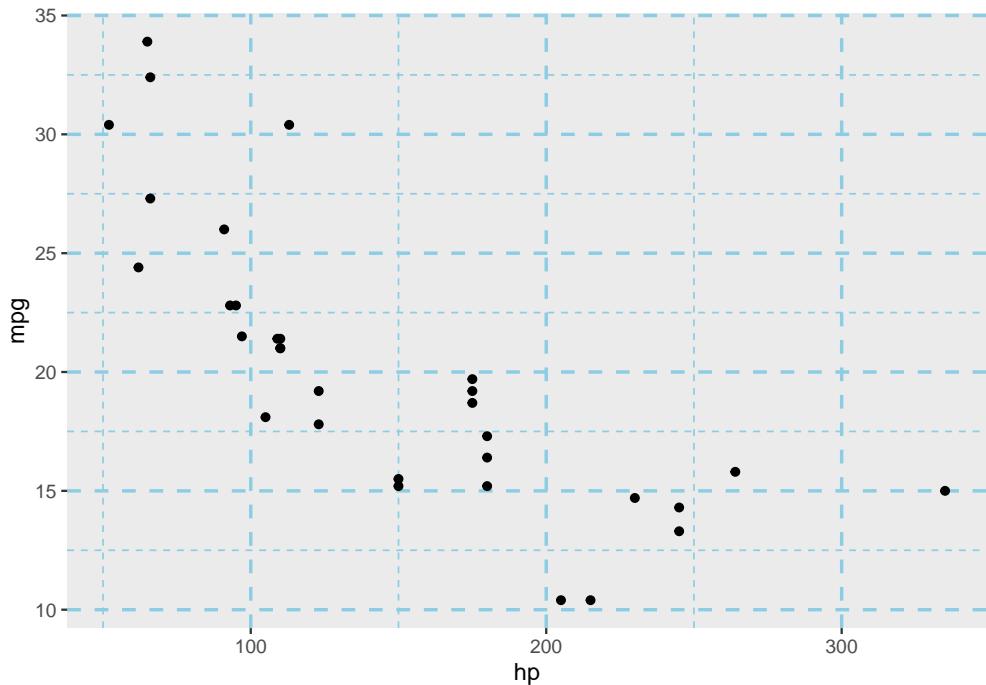


3.1 PERSONALIZACIÓN DE LA REJILLA

La estética de la rejilla se puede personalizar con el componente `panel.grid` de la función `theme()`. Personaliza el color, el ancho o el tipo de línea con los argumentos de la función `element_line()`.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid = element_line(color = "#8ccde3",
                                   size = 0.75,
                                   linetype = 2))
```



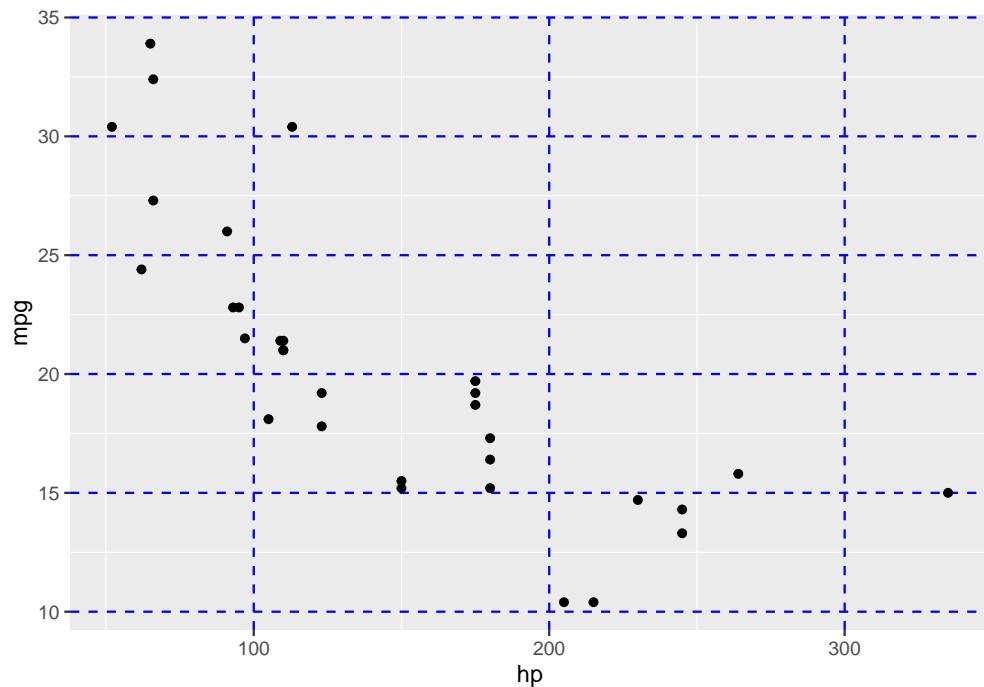
También se puede personalizar los grids principal y secundario de manera individual, tal y como se muestra en las siguientes secciones.

3.2 MAJOR GRID

El componente `panel.grid.major` permite personalizar el grid principal (major grid) del panel.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major = element_line(color = "blue",
                                         size = 0.5,
                                         linetype = 2))
```

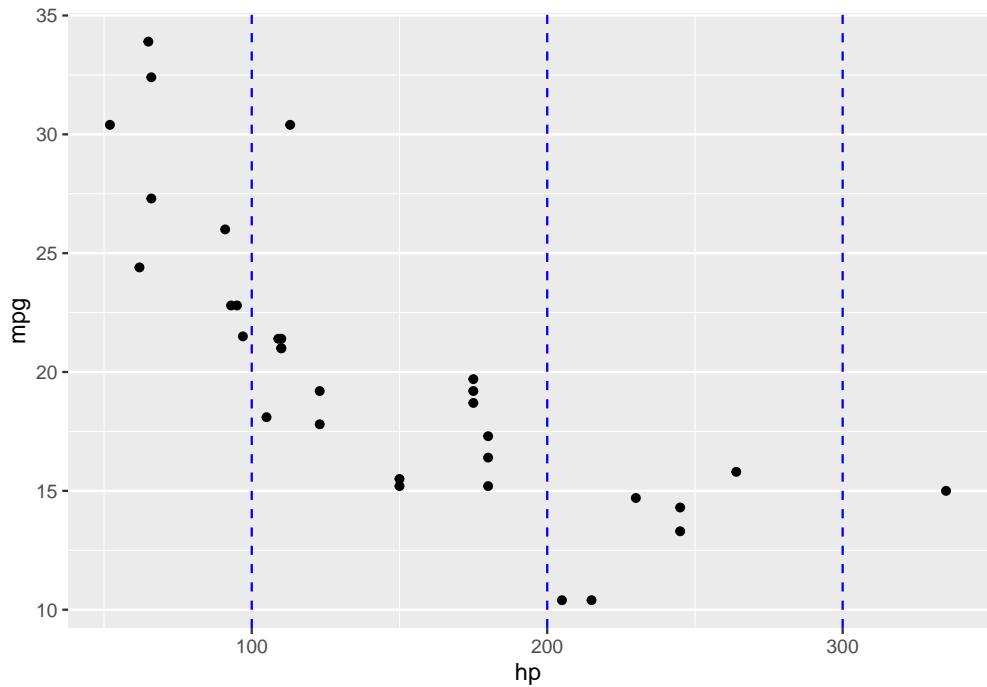


Agregando `.x` o `.y` al componente anterior se podrá personalizar las líneas verticales u horizontales del grid principal.

3.2.1 LÍNEAS VERTICALES DEL GRID PRINCIPAL

```
library(ggplot2)

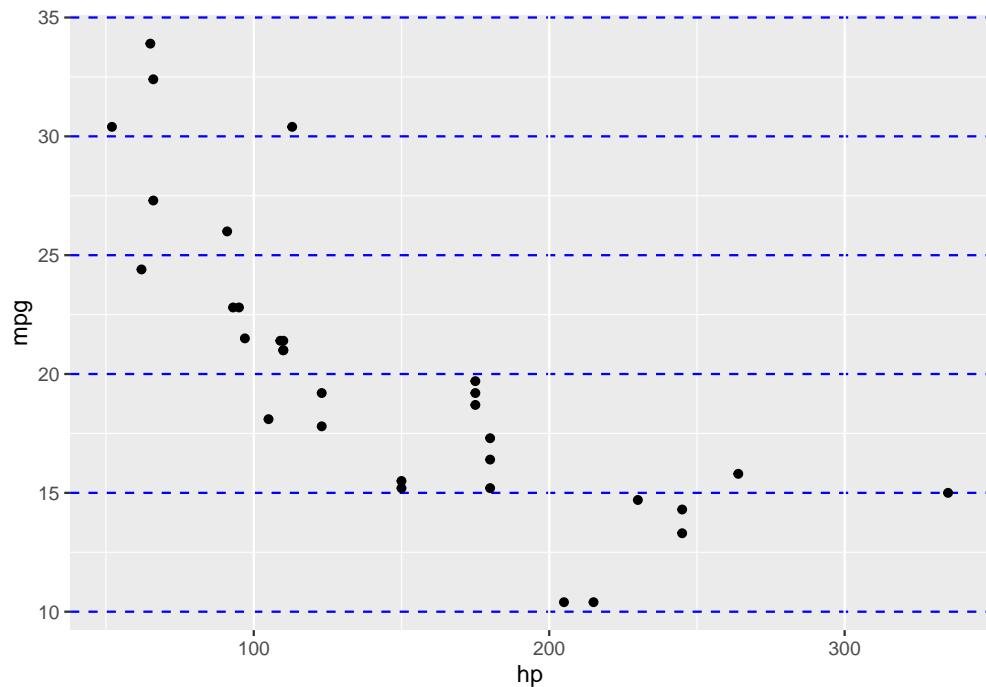
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major.x = element_line(color = "blue",
                                            size = 0.5,
                                            linetype = 2))
```



3.2.2 LÍNEAS HORIZONTALES DEL GRID PRINCIPAL

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major.y = element_line(color = "blue",
                                            size = 0.5,
                                            linetype = 2))
```

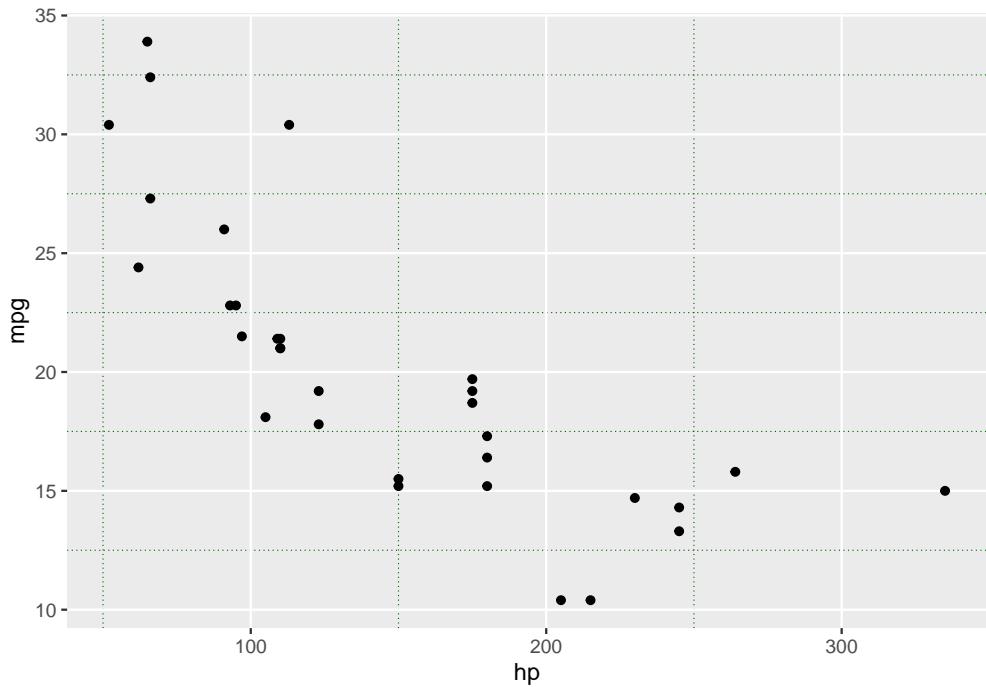


3.3 MINOR GRID

`panel.grid.minor` permite personalizar el grid secundario (o minor grid) del panel.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor = element_line(color = "darkgreen",
                                         size = 0.25,
                                         linetype = 3))
```

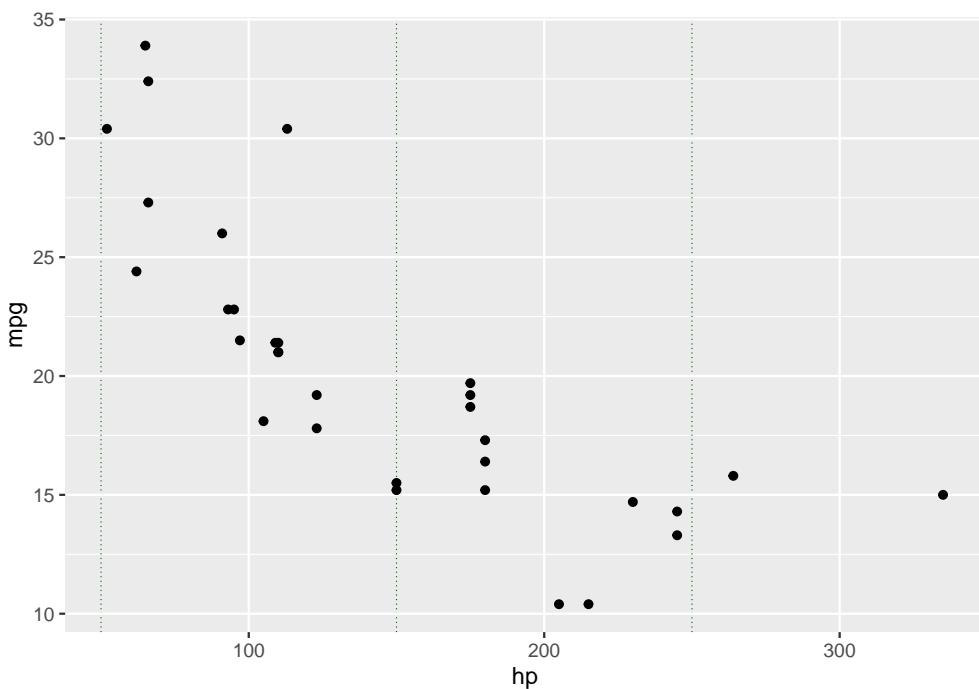


Igual que en la sección anterior, se puede agregar .x o .y al elemento.

3.3.1 LÍNEAS VERTICALES DEL GRID SECUNDARIO

```
library(ggplot2)

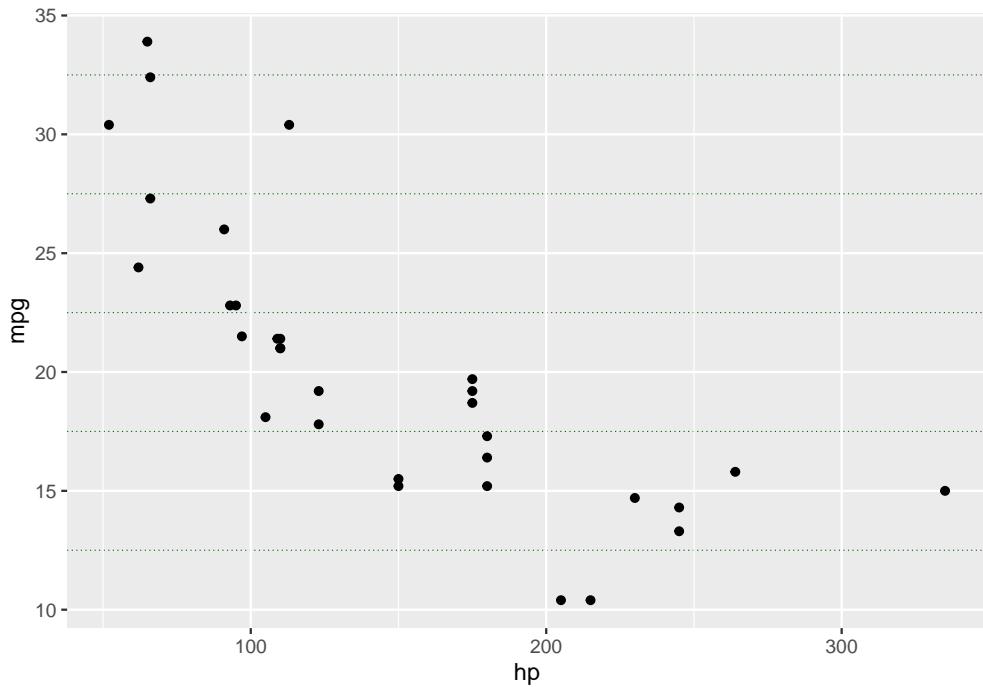
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor.x = element_line(color = "darkgreen",
                                            size = 0.25,
                                            linetype = 3))
```



3.3.2 LÍNEAS HORIZONTALES DEL GRID SECUNDARIO

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor.y = element_line(color = "darkgreen",
                                            size = 0.25,
                                            linetype = 3))
```



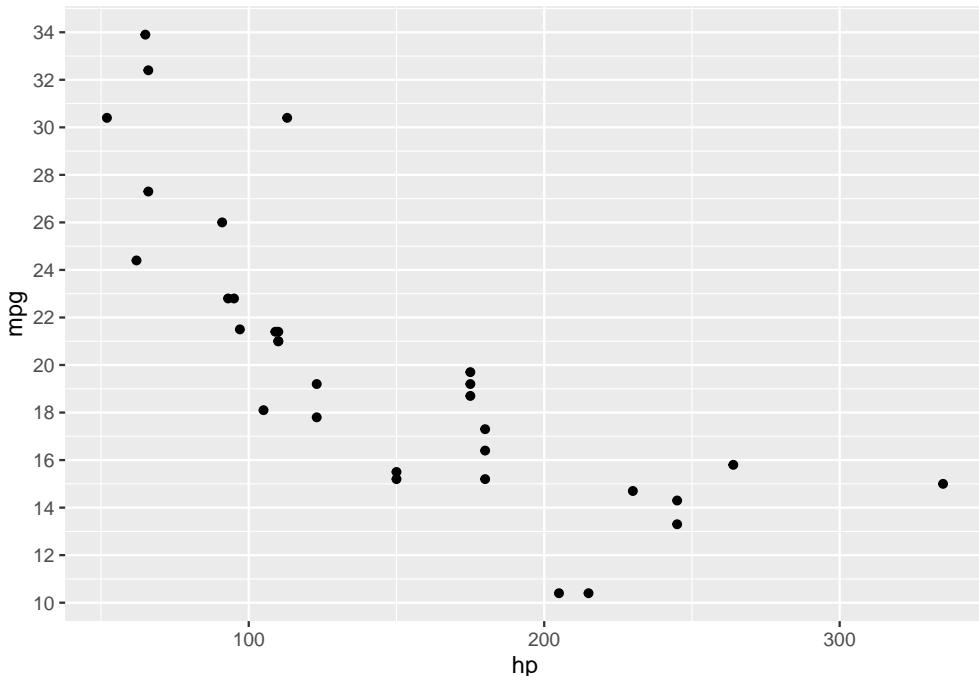
3.4 NÚMERO DE LÍNEAS

Los cortes del grid se pueden personalizar en `ggplot2` para cada eje con el argumento `breaks` de la función `scale_(x|y)_continuous()` o `scale_(x|y)_discrete()`, dependiendo si la variable del eje (`x|y`) es continua o discreta.

En este ejemplo se personalizarán los intervalos del eje Y con la función `scale_y_continuous()`.

```
library(ggplot2)

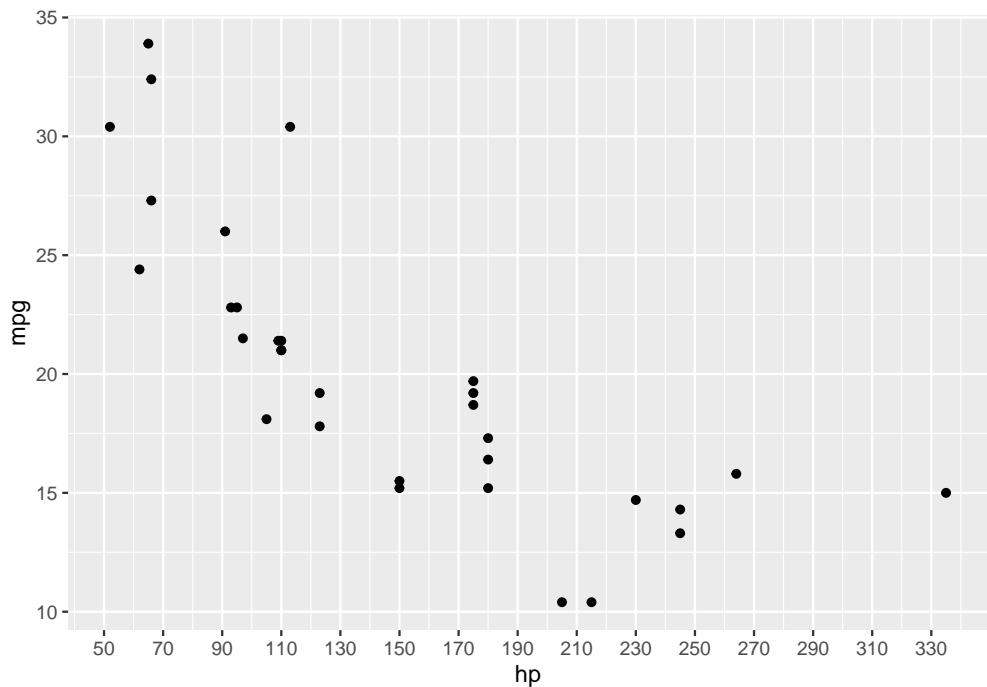
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_y_continuous(breaks = seq(10, 35, by = 2))
```



Como el eje X también es continuo, se puede usar la función `scale_x_continuous()` para personalizar los intervalos del grid del eje X.

```
library(ggplot2)

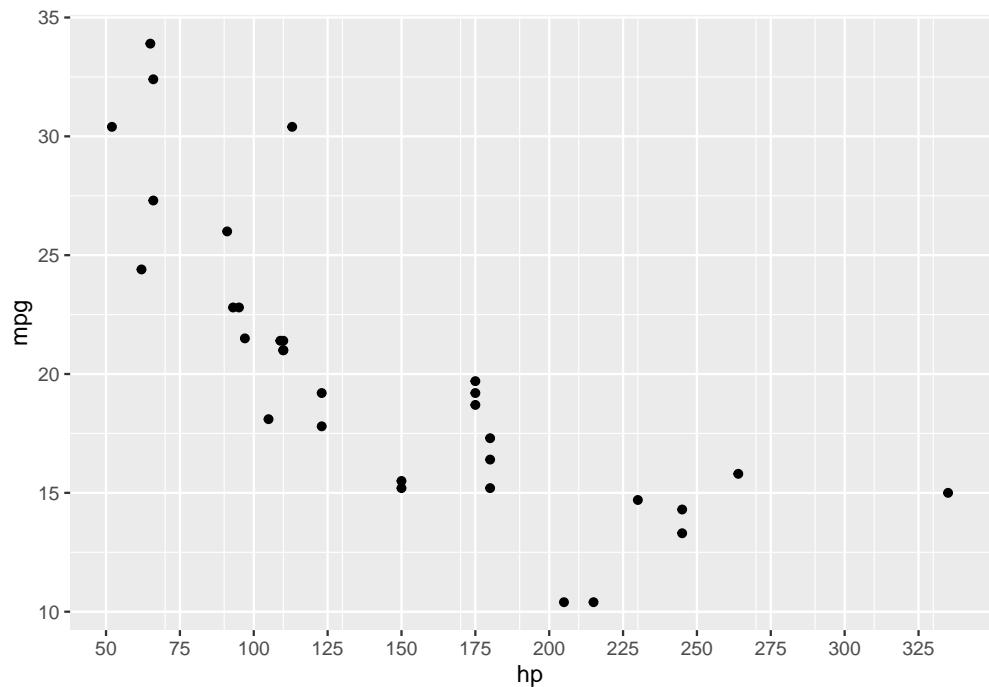
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by = 20))
```



3.4.1 ESTABLECER EL NÚMERO DE LÍNEAS DEL GRID SECUNDARIO CON EL ARGUMENTO `minor_breaks`

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by = 25),
                     minor_breaks = seq(50, 350, 20))
```

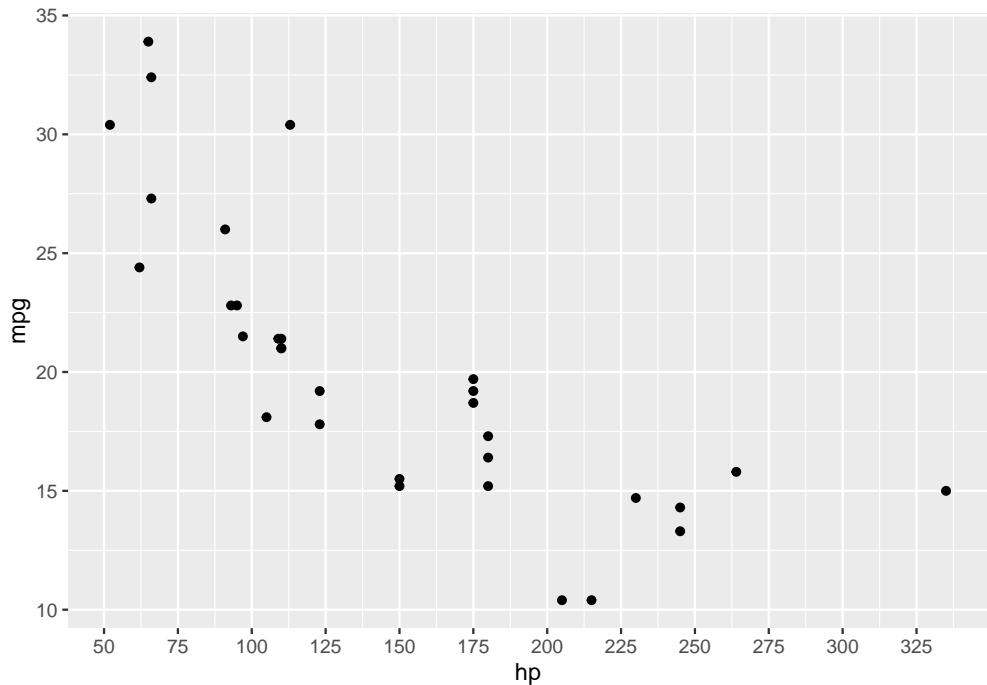


3.4.2 ESTABLECER EL NÚMERO DE LÍNEAS DEL GRID PRINCIPAL CON EL ARGUMENTO `n.breaks`⁷

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(n.breaks = 15)
```

⁷El algoritmo detrás de la generación de cortes en el eje para crear el grid principal, se puede elegir un número diferente al que se especifique para asegurar que las etiquetas de los ejes se lean correctamente.

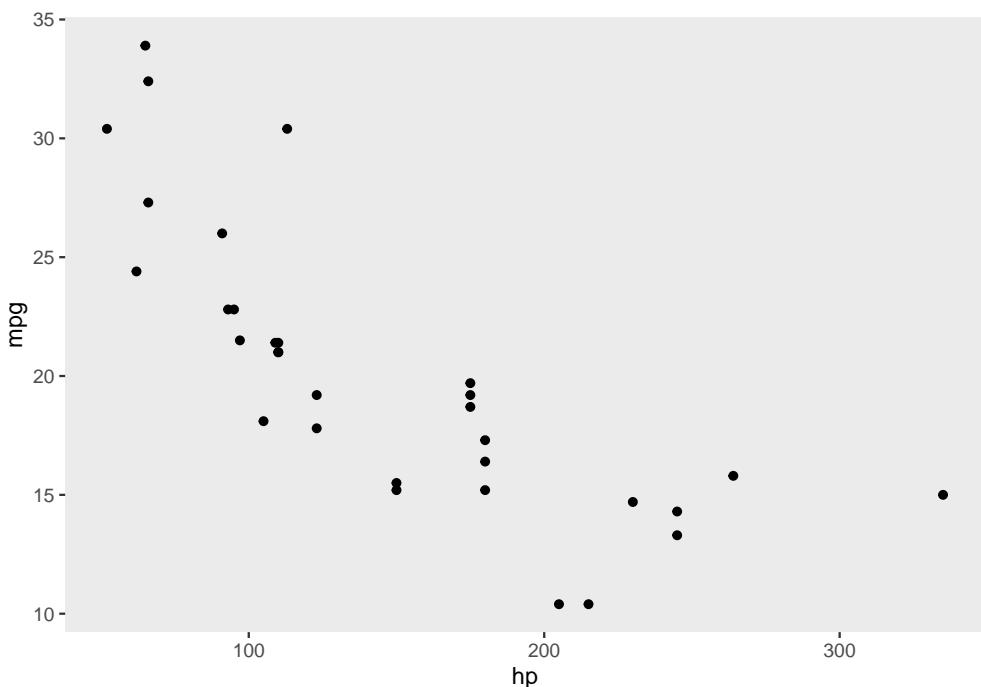


3.5 ELIMINAR EL GRID

De la misma manera que se puede personalizar cada `grid(panel.grid, panel.major, panel.major.x, panel.major.y, ..., panel.minor, panel.minor.x, panel.minor.y)` también se puede eliminar usando el argumento `element_blank()` en lugar de `element_line("")`.

```
library(ggplot2)

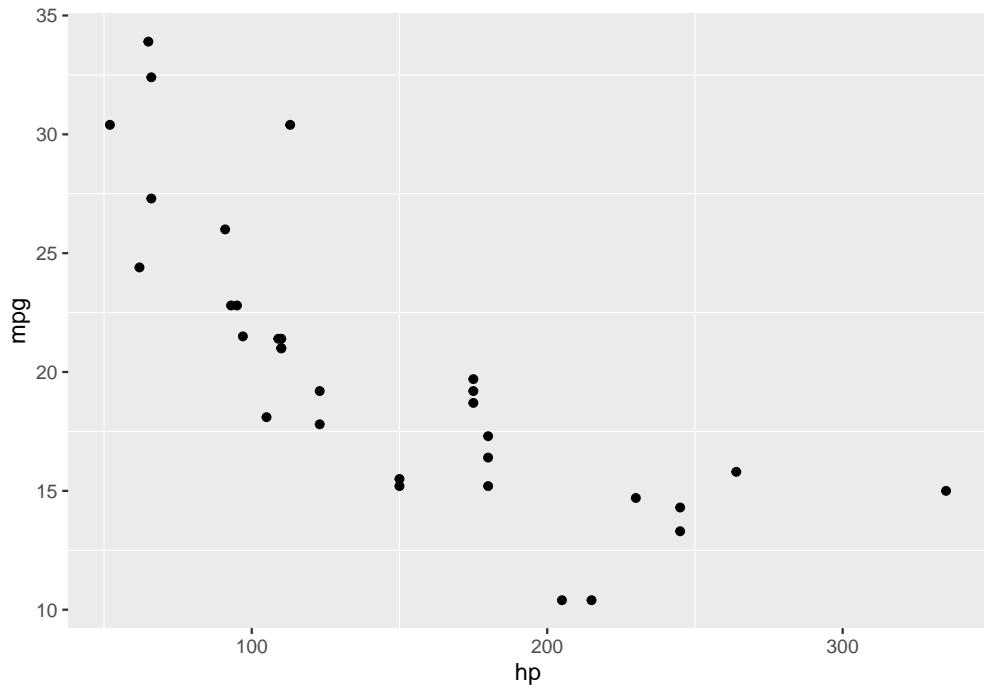
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid = element_blank())
```



3.5.1 ELIMINAR EL GRID PRINCIPAL

```
library(ggplot2)

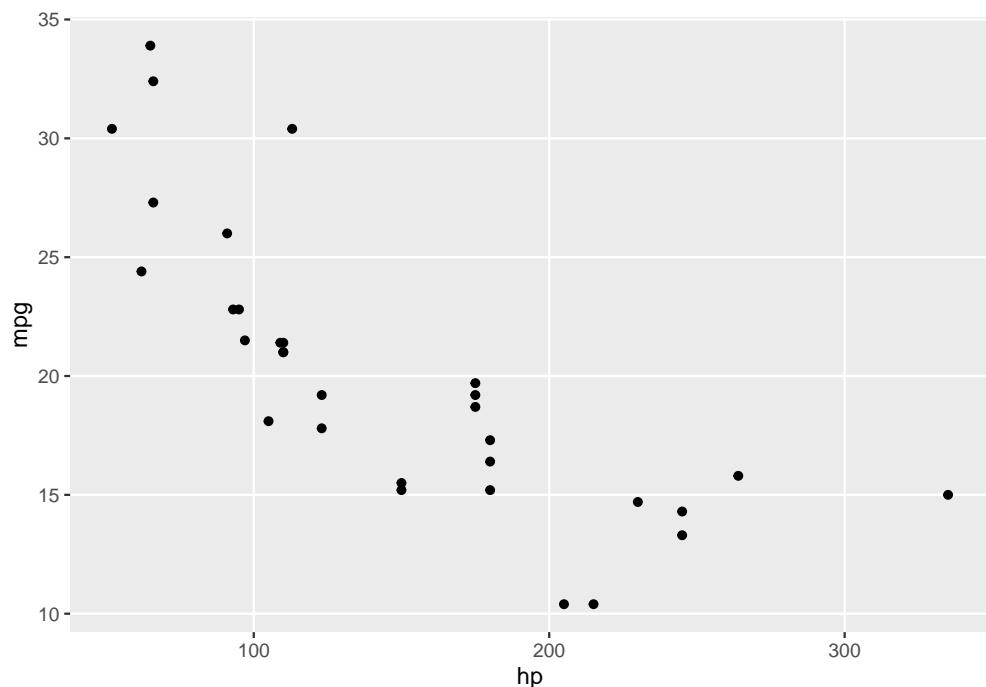
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major = element_blank())
```



3.5.2 ELIMINAR EL GRID SECUNDARIO

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor = element_blank())
```



CAPÍTULO 4: ANOTACIONES DE TEXTO EN `ggplot2`



Las funciones `geom_text()` y `geom_label()` permiten añadir textos o etiquetas, respectivamente, a los gráficos creados con `ggplot2`. Se pueden añadir anotaciones a ciertas coordenadas de puntos o etiquetar las observaciones.

4.1 AGREGANDO TEXTOS CON `geom_text`

```
# install.packages("ggplot2")
library(ggplot2)

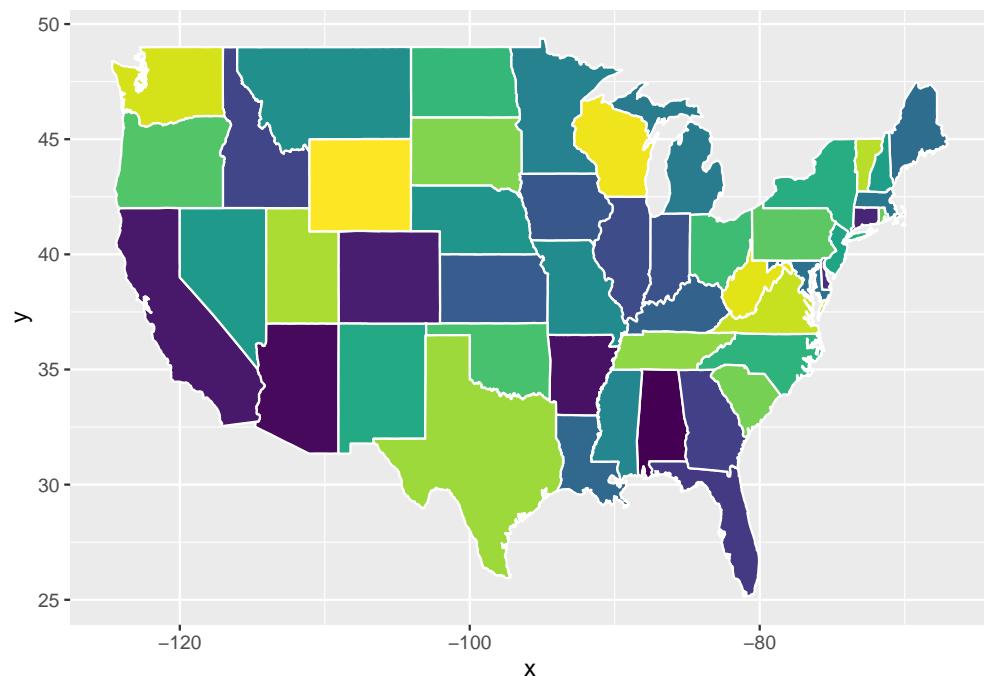
# install.packages("maps")
library(maps)
```

```
# install.packages("viridis")
library(viridis)

df = data.frame(x = state.center$x, y = state.center$y,
                 state = state.name)

p = ggplot(df, aes(x = x, y = y)) +
    geom_polygon(data = map_data("state"),
                  color = "white",
                  aes(x = long, y = lat,
                      fill = map_data("state")$region,
                      group = group)) +
    guides(fill = FALSE) +
    scale_fill_viridis(discrete = TRUE)

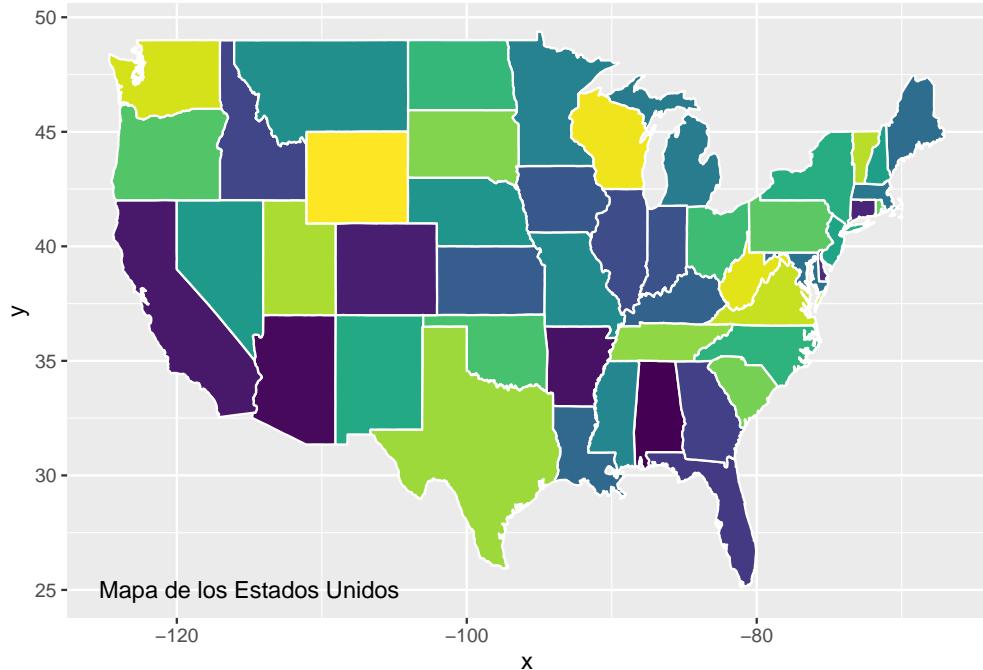
p
```



4.1.1 AÑADIENDO UNA ANOTACIÓN DE TEXTO⁸

⁸Se establece `stat = "unique"`, ya que en otro caso las etiquetas se dibujarán para cada punto del data frame, una encima de otra.

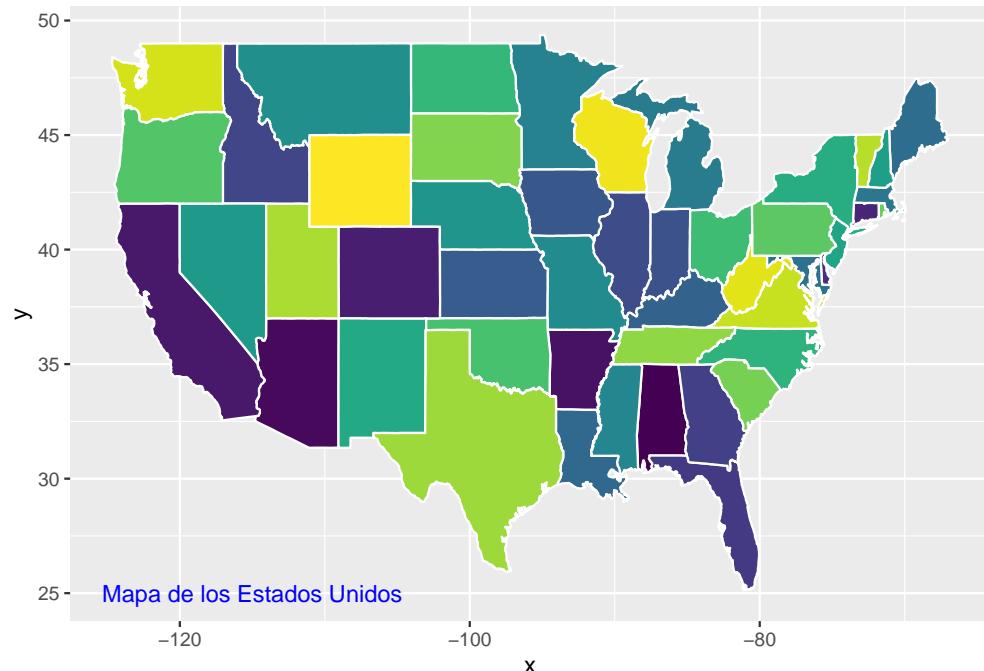
```
p +  
  geom_text(aes(x = -115, y = 25,  
                label = "Mapa de los Estados Unidos"),  
            stat = "unique")
```



4.1.2 PERSONALIZAR LAS ANOTACIONES

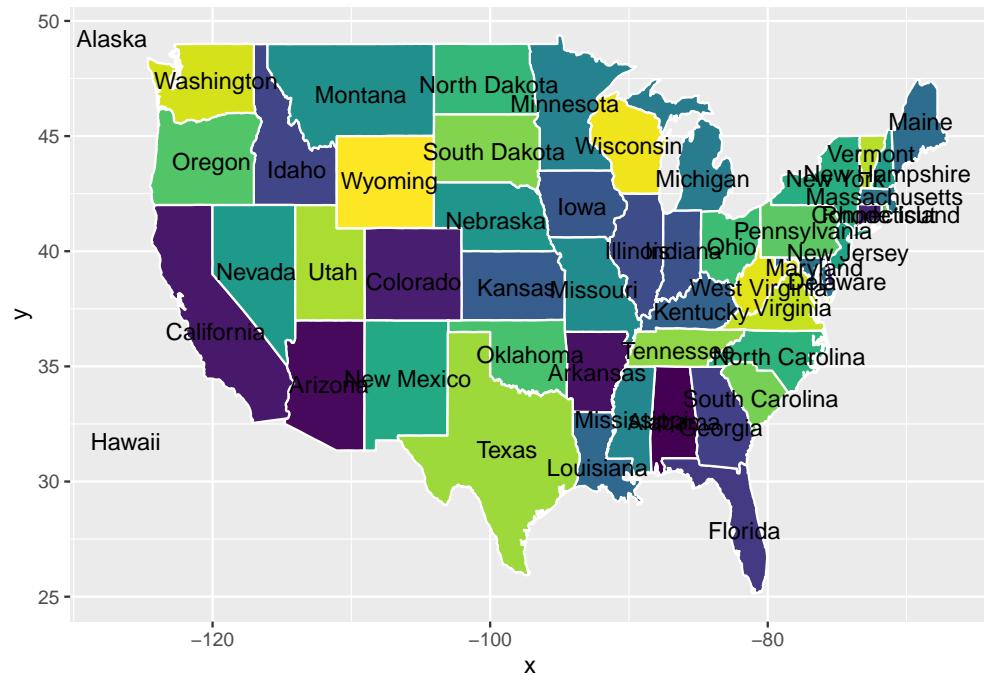
Existen varios argumentos que se pueden utilizar para personalizar, como el color y el tamaño del texto.

```
p +  
  geom_text(aes(x = -115, y = 25,  
                label = "Mapa de los Estados Unidos"),  
            stat = "unique",  
            size = 4, color = "blue")
```



Si el conjunto de datos contiene una variable que representa grupos o etiquetas, se puede pasar dentro del argumento `label`.

```
p +
  geom_text(aes(label = state))
```

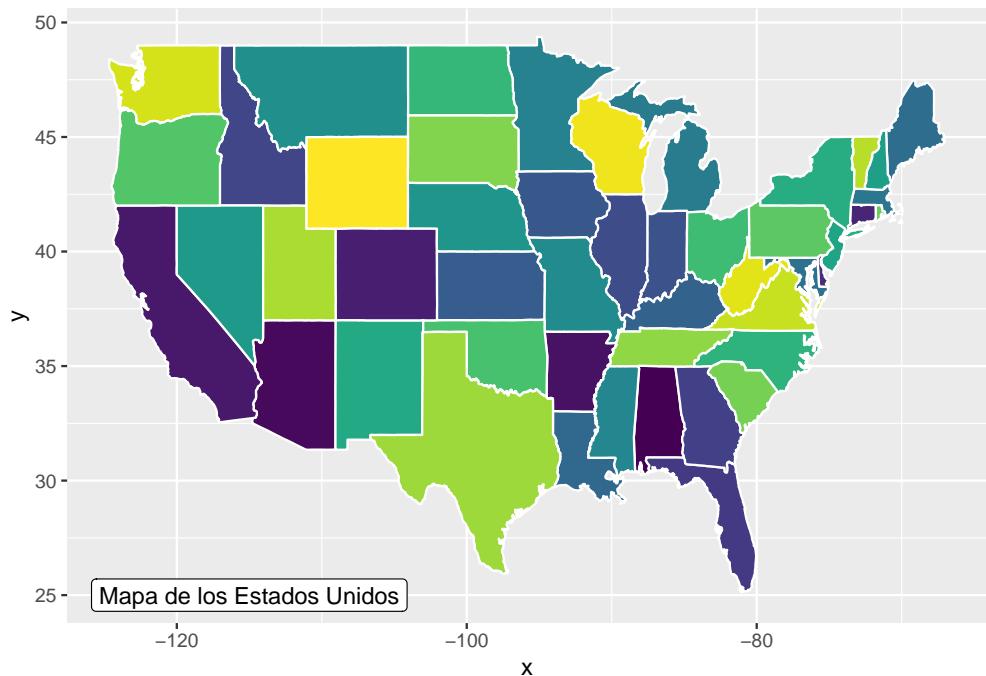


4.2 AGREGANDO ETIQUETAS CON `geom_label()`

Si se prefiere agregar etiquetas en lugar de textos, es preferible usar la función `geom_label()`. La función se comporta de la misma manera que la anterior, pero tiene un color de fondo, haciendo los textos más fáciles de leer.

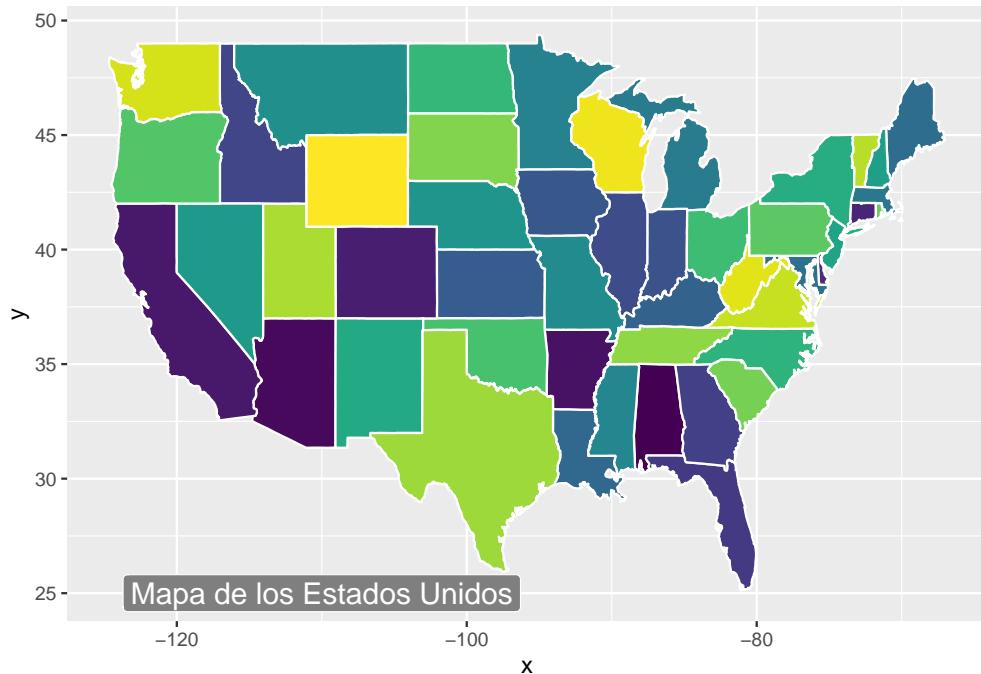
4.2.1 ETIQUETAS

```
p +
  geom_label(aes(x = -115, y = 25,
                  label = "Mapa de los Estados Unidos"),
             stat = "unique")
```



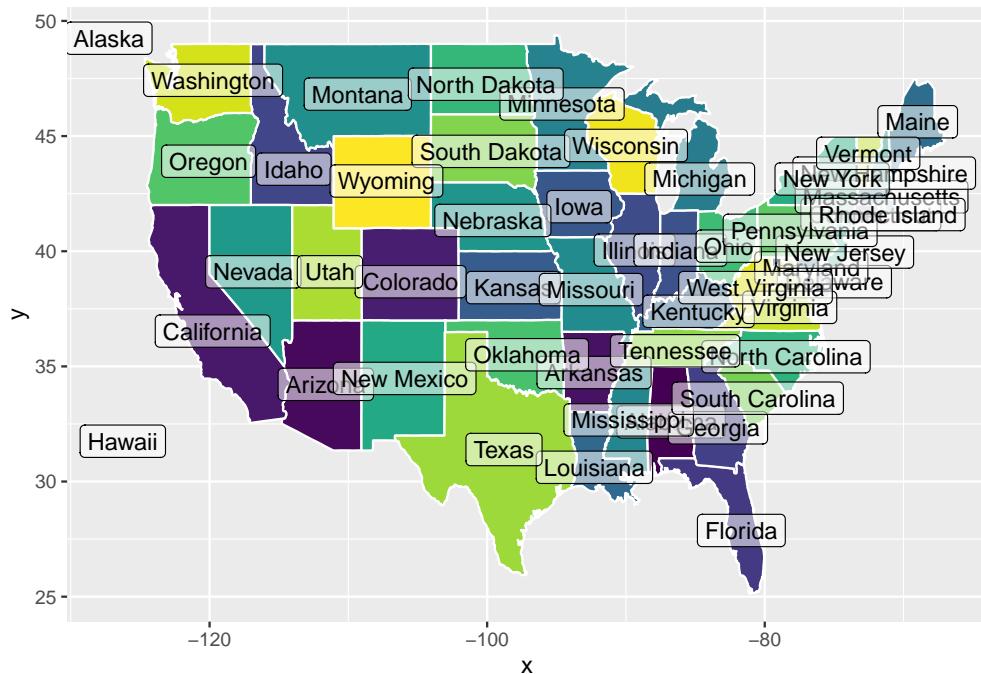
4.2.2 PERSONALIZAR LAS ETIQUETAS

```
p +
  geom_label(aes(x = -110, y = 25,
                  label = "Mapa de los Estados Unidos"),
             stat = "unique",
             size = 5, color = "white", fill = "gray50")
```



4.2.3 ETIQUETAR OBSERVACIONES

```
p +
  geom_label(aes(label = state, alpha = 0.6)) +
  theme(legend.position = "none")
```



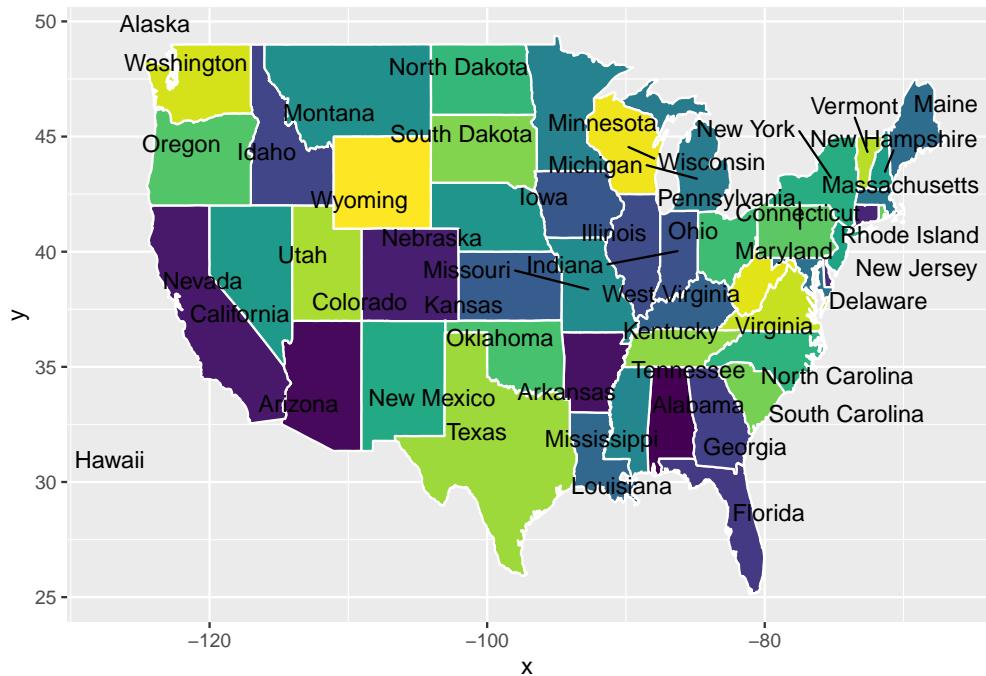
4.3 EVITAR SOLAPAMIENTOS CON LA LIBRERÍA ggrepel

Los textos y las etiquetas se colocan en las coordenadas que se seleccionen, pero se pueden solapar. El paquete `ggrepel` proporciona las funciones `geom_text_repel()` y `geom_label_repel()`, que hacen que las etiquetas se repelen tanto como sea posible.

4.3.1 EL USO DE LA FUNCIÓN `geom_text_repel()`

```
# install.packages("ggrepel")
library(ggrepel)

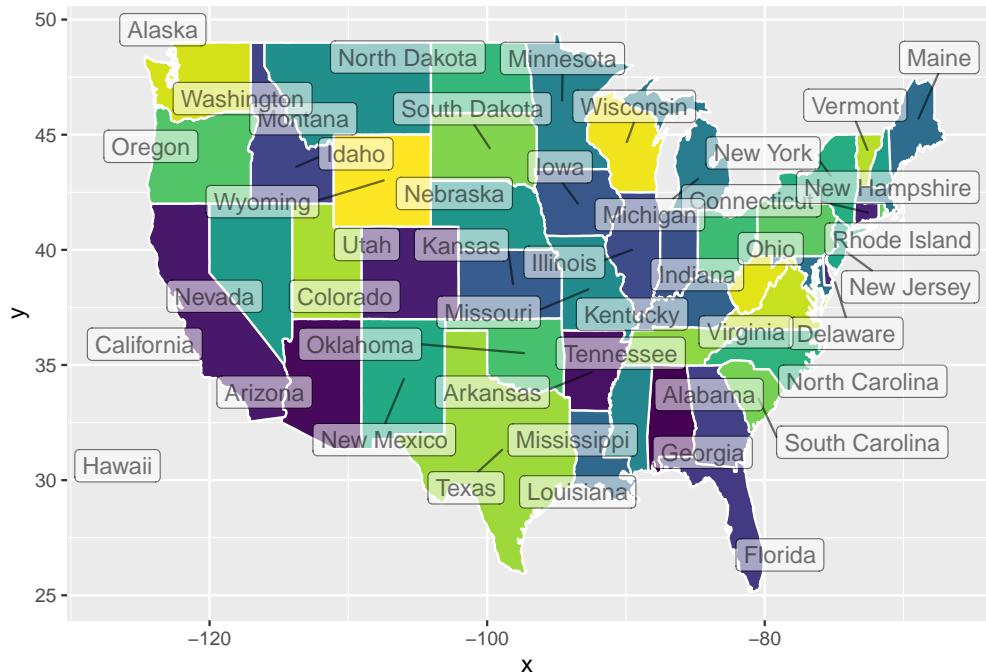
p +
  geom_text_repel(aes(label = state))
```



4.3.2 EL USO DE LA FUNCIÓN `geom_label_repel()`

```
# install.packages("ggrepel")
library(ggrepel)
```

```
p +  
  geom_label_repel(aes(label = state, alpha = 0.6)) +  
  theme(legend.position = "none")
```



Se pueden personalizar los colores, las fuentes y otros argumentos de la misma manera que con `geom_text()` o `geom_label()`.⁹

4.4 USO DE MARKDOWN Y HTML CON ggtext

Si se quiere personalizar las anotaciones completamente, se debe de usar la función `geom_richtext()` del paquete `ggttext`, ya que permite añadir formato markdown o HTML a las anotaciones de texto.

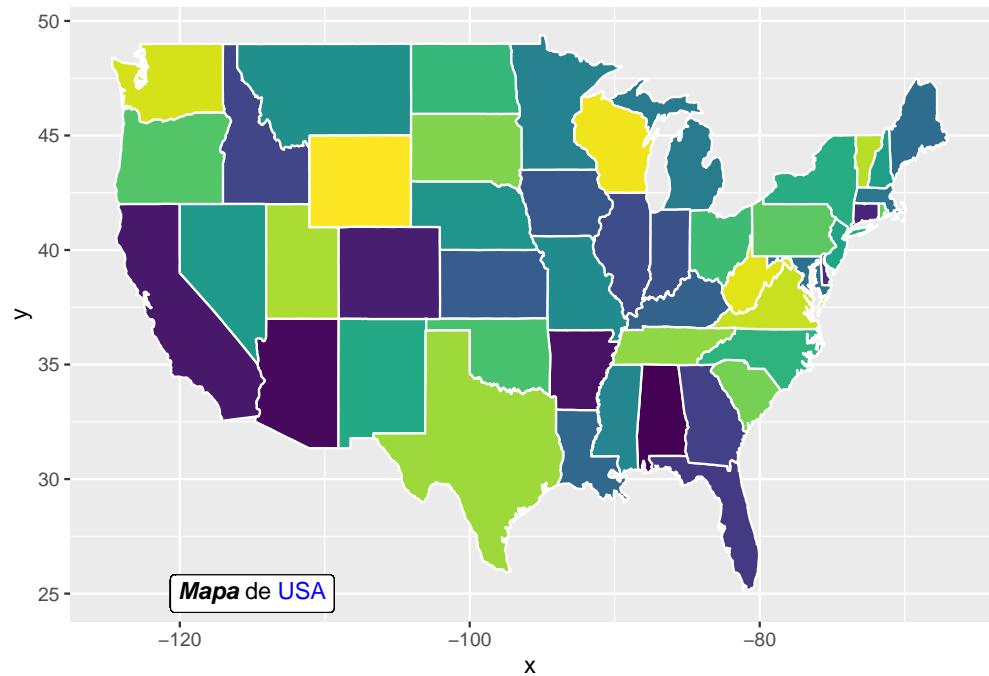
4.4.1 TEXTO ENRIQUECIDO

```
# install.packages("ggtext")
library(ggtext)

lab = "***Mapa*** de <span style = 'color:blue'>USA</span>"
```

⁹Mira más ejemplos del paquete para distintos casos posibles al acceder al dar [click aquí](#).

```
p +
  geom_richtext(aes(x = -115, y = 25,
                     label = lab))
```

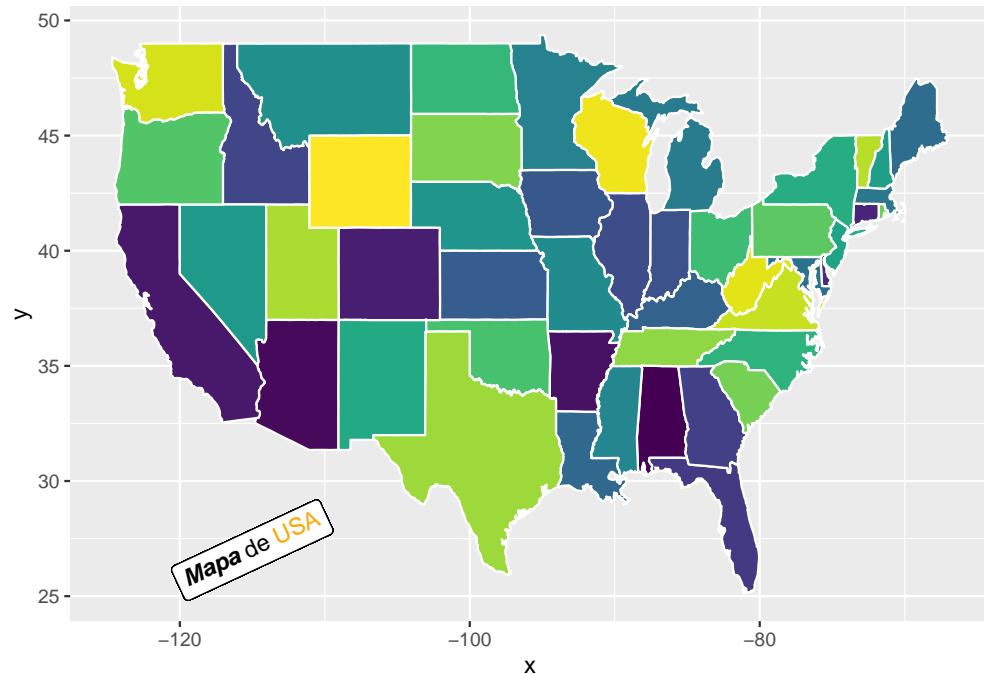


4.4.2 ROTAR EL TEXTO

La función permite incluso rotar el texto, algo que no es posible con las funciones de `ggplot2`.

```
# install.packages("ggtext")
library(ggtext)

lab = "***Mapa*** de <span style = 'color:orange'>USA</span>"
p +
  geom_richtext(aes(x = -115, y = 27,
                     label = lab),
                angle = 25)
```



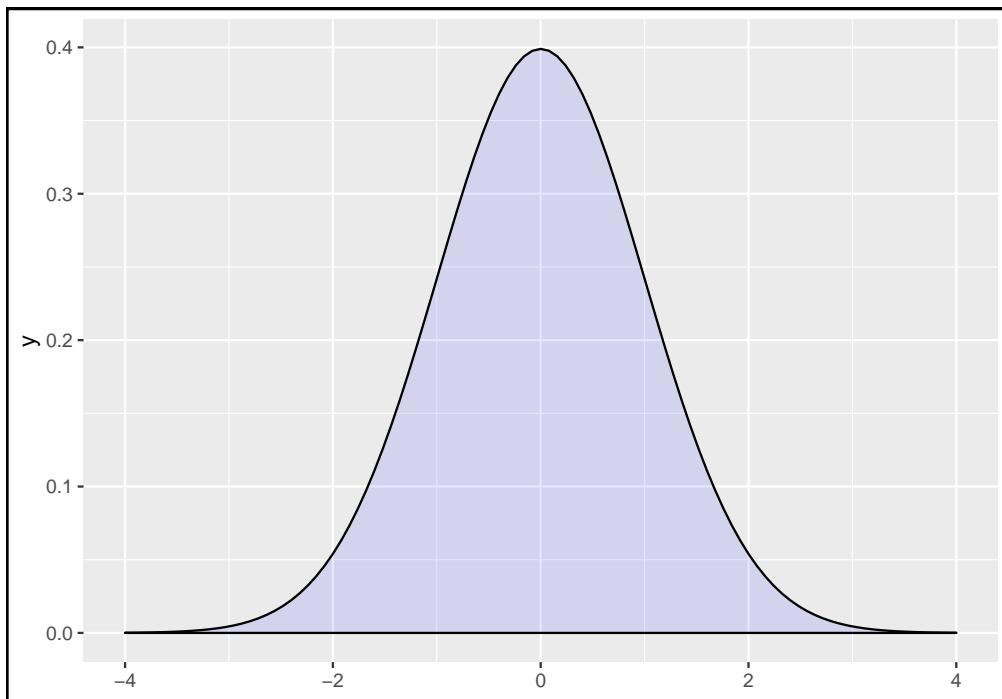
CAPÍTULO 5: MÁRGENES EN ggplot2



Los márgenes de los diagramas hechos con ggplot2 se ajustarán de manera automática a las nuevas capas. En este ejemplo se agregará un borde negro al rededor del gráfico que muestra los cambios en los márgenes.

```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
               xlim = c(-4, 4),
               fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                        size = 1))
```

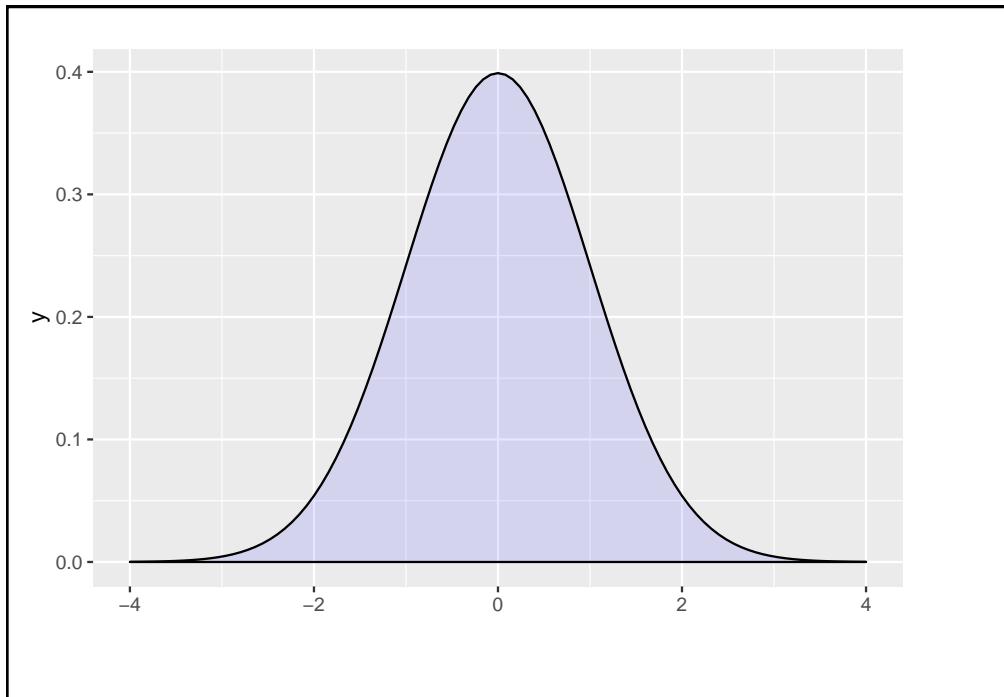


5.1 INCREMENTAR LOS MÁRGENES

Para modificar los márgenes del diagrama, solamente se debe de pasar la función `margin()` dentro del componente `plot.margin` de la función `theme()`.

```
library(ggplot2)

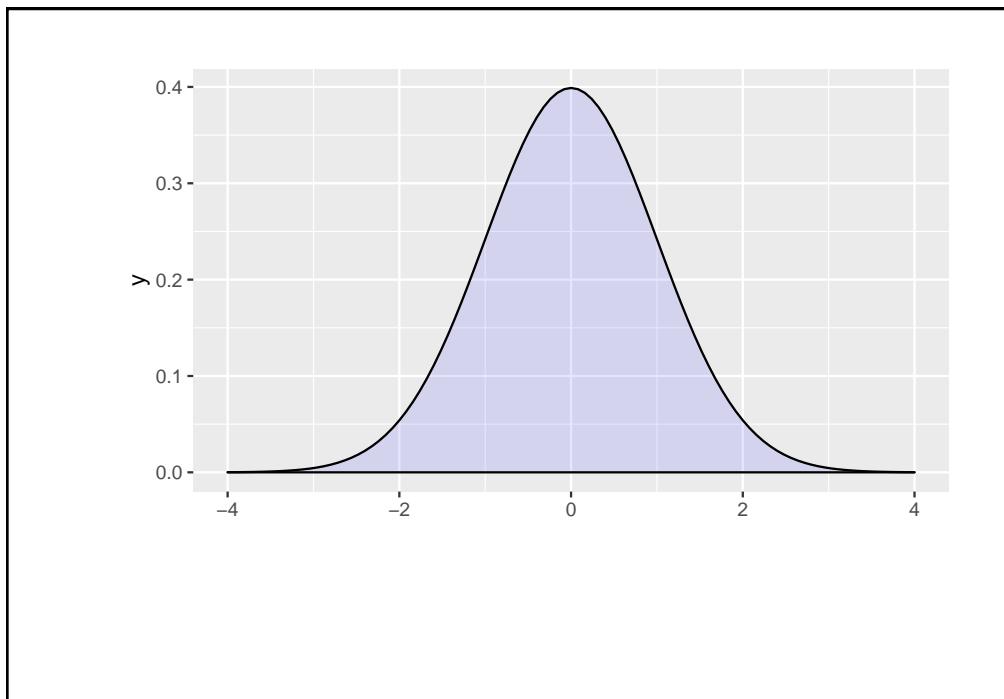
ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                         size = 1),
        plot.margin = margin(t = 20, # Margen superior
                           r = 50, # Margen derecho
                           b = 40, # Margen inferior
                           l = 10)) # Margen izquierdo
```



Los márgenes se miden con puntos ("pt"), pero se puede usar otra unidad de medida en el argumento `unit`, como centímetros. Solamente se escribe `?unit` para ver todas las posibles medidas.

```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
    xlim = c(-4, 4),
    fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
    size = 1),
    plot.margin = margin(t = 1, # Margen superior
      r = 1, # Margen derecho
      b = 3, # Margen inferior
      l = 2, # Margen izquierdo
      unit = "cm"))
```



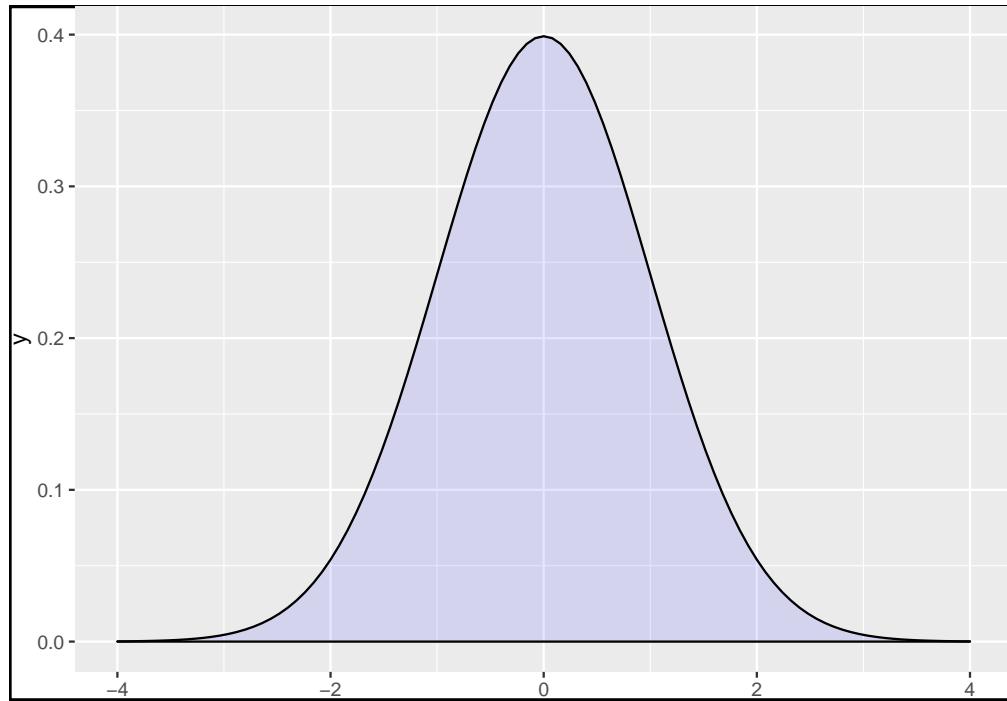
5.2 ELIMINAR LOS MÁRGENES

Para eliminar los márgenes, se establece todos los valores a 0. Hay que tener en cuenta que todavía quedará espacio para que entren todos los elementos del gráfico. Se pueden pasar valores negativos para reducir los márgenes.¹⁰

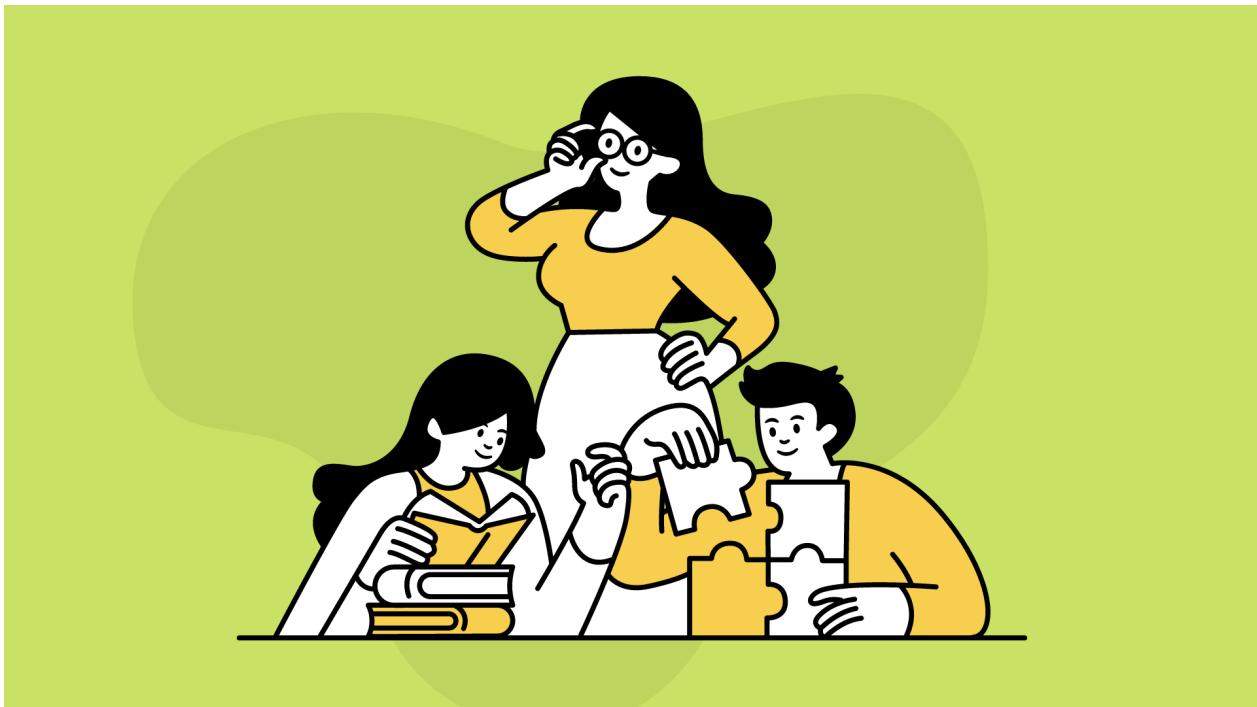
```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                         size = 1),
        plot.margin = margin(t = 0, # Margen superior
                           r = 0, # Margen derecho
                           b = 0, # Margen inferior
                           l = 0)) # Margen izquierdo
```

¹⁰Se puede recordar el orden de los argumento de la función `margin(t, r, b, l)`, recordando la palabra **trabalenguas**.



CAPÍTULO 6: LEYENDAS EN ggplot2



6.1 DATOS DE MUESTRA

Dentro de esta sección del manual, lo que se va a utilizar es el siguiente conjunto de datos que se categorizan en dos grupos.

```
# Datos
set.seed(3)
df = data.frame(x = c(rnorm(300, -3, 1.5),
                      rnorm(300, 0, 1)),
                 grupo = c(rep("A", 300),
                           rep("B", 300)))
```

6.2 AGREGANDO UNA LEYENDA

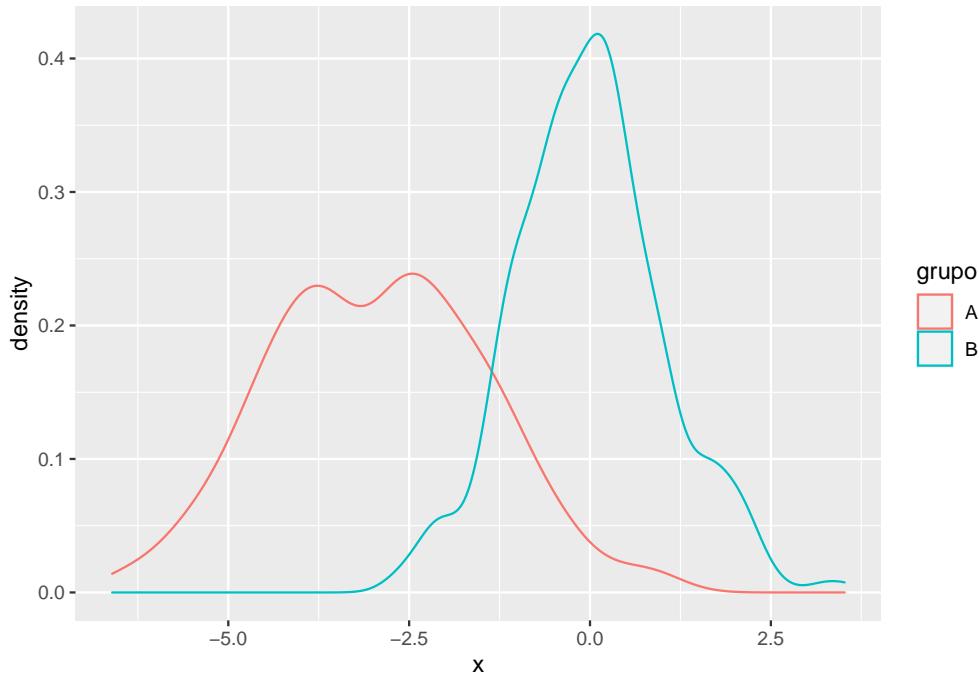
Si se quiere agregar una leyenda a un gráfico hecho con la librería `ggplot2` se tendrá que pasar una variable categórica (o numérica) a `color`, `fill`, `shape` o `alpha` dentro de la función `aes()`. Dependiendo de cuál argumento se utilice y del caso particular de cada salida.

6.2.1 COLOR

Pasando la variable categórica a `color` dentro de la función `aes()`, los datos se dibujarán con base a los grupos y se mostrará una leyenda de manera automática.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, color = grupo)) +
  geom_density(alpha = 0)
```

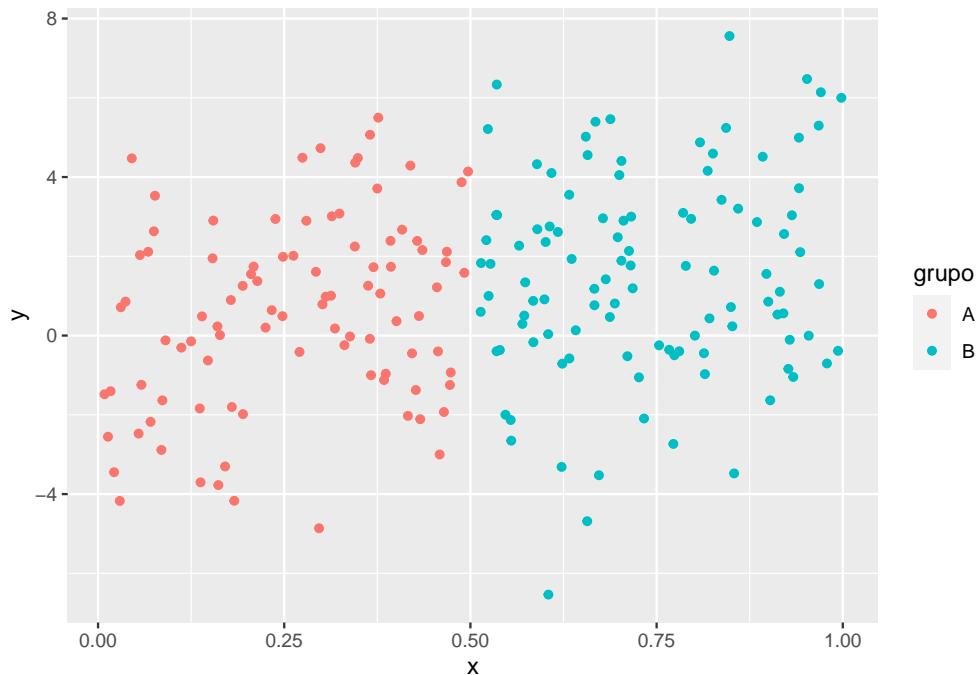


Hay que tener en cuenta que este es el argumento que se debería de usar si se quiere crear un gráfico que no admite un color de fondo, como un diagrama de correlación. Se puede usar una variable categórica o numérica.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
# Datos
x = runif(200)
y = 3 * x ^ 2 + rnorm(length(x), sd = 2.5)
grupo = ifelse(x < 0.5, "A", "B")
df2 = data.frame(x = x, y = y, grupo = grupo)

ggplot(df2, aes(x = x, y = y, color = grupo)) +
  geom_point()
```

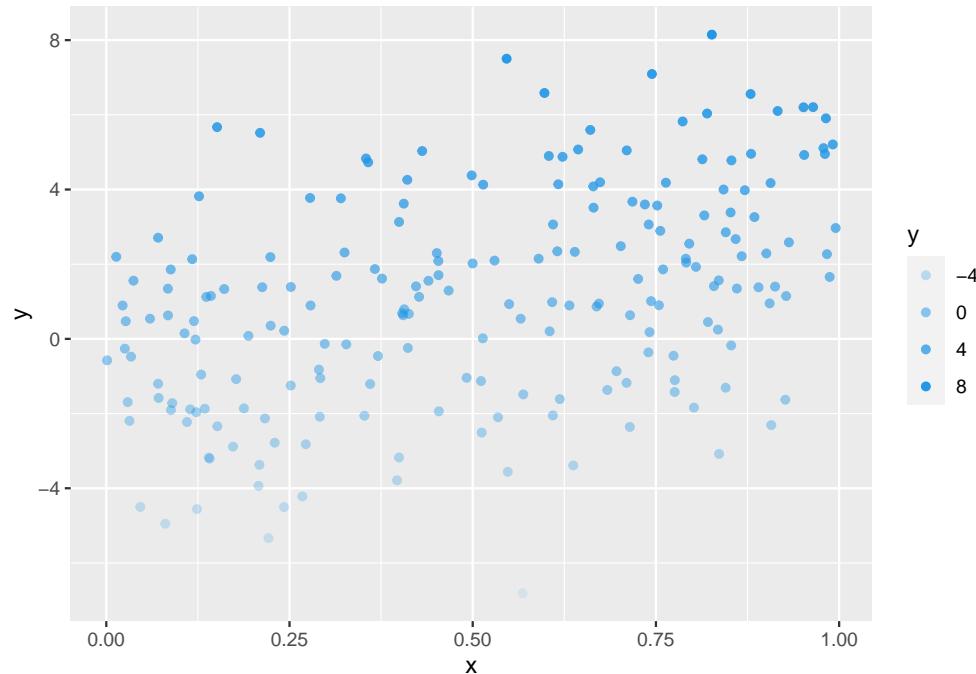


6.2.2 ALPA (TRANSPARENCIA)

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
x = runif(200)
y = 4 * x ^ 2 + rnorm(length(x), sd = 2.5)
group = ifelse(x < 0.5, "A", "B")
df2 = data.frame(x = x, y = y)

ggplot(df2, aes(x = x, y = y, alpha = y)) +
  geom_point(colour = 4)
```

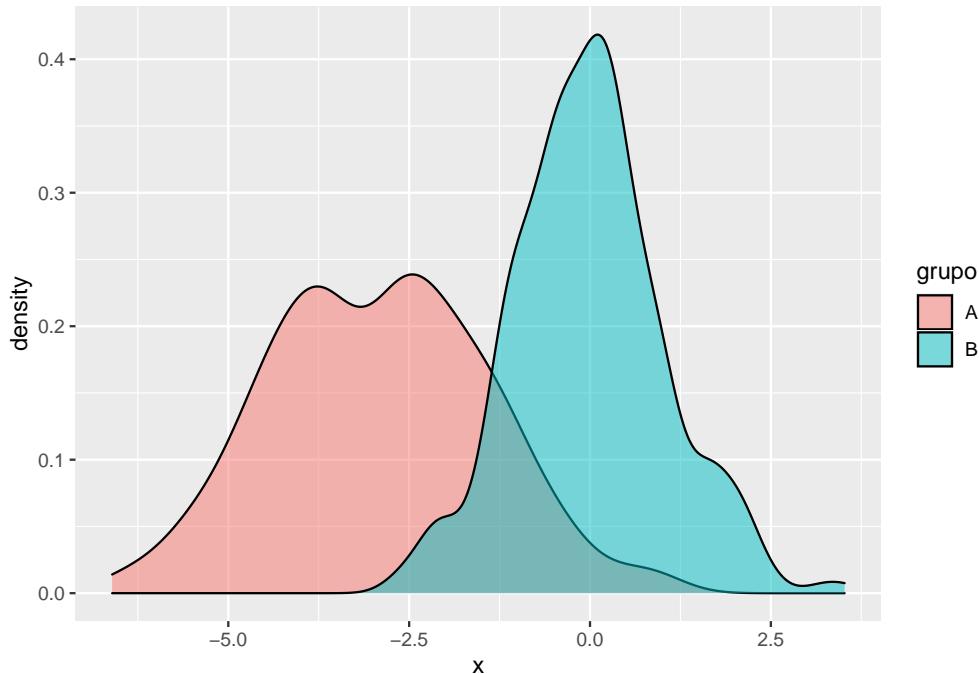


6.2.3 FILL

Si el gráfico que se está creando permite agregar un color de relleno, se puede usar el argumento `fill` dentro de la función `aes()`, de modo que las cajas de la leyenda se colorearán.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5, col = "black")
```



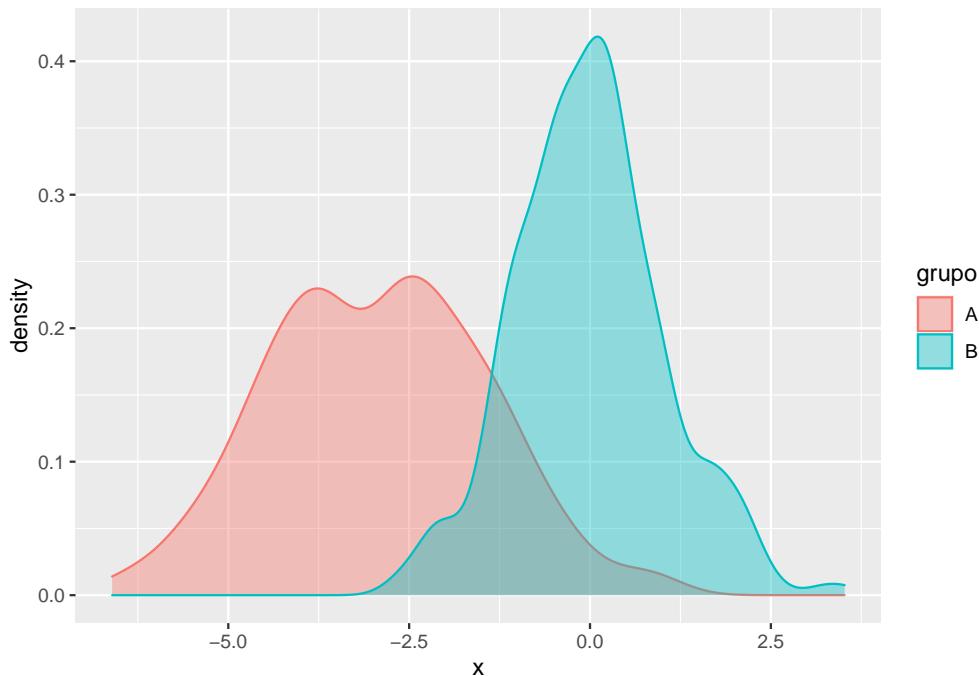
6.2.4 COLOR Y FILL A LA VEZ

En el escenario anterior también se puede usar ambos argumentos (`fill` y `color`), de forma que la leyenda se coloreará con el color correspondiente y el borde también tendrá el color que corresponde a cada grupo.¹¹

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo, color = grupo)) +
  geom_density(alpha = 0.4)
```

¹¹Si se desea cambiar los colores de fondo o de borde del gráfico (y, por lo tanto los de la leyenda), es mejor revisar otros tutoriales, como el de [gráficos de densidad por grupo](#). Tema que se puede encontrar al dar click en las letras negritas.



6.3 CAMBIAR O ELIMINAR EL TÍTULO DE LA LEYENDA

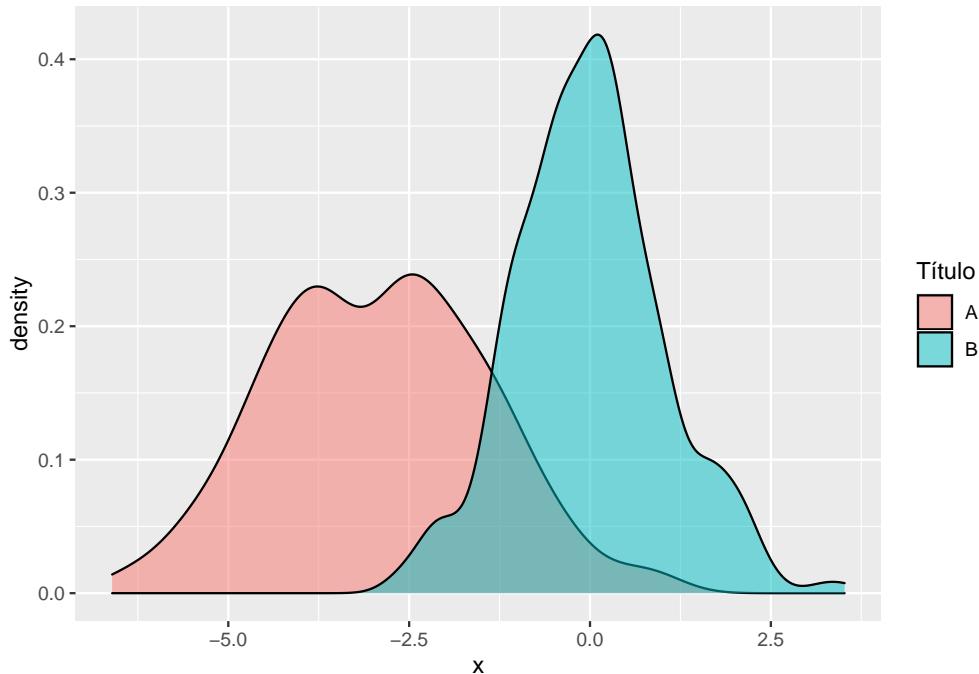
Existen varias formas de cambiar el título de la leyenda de un diagrama. Hay que tener en cuenta que la opción elegida dependerá del tipo de diagrama y las preferencias.

6.3.1 OPCIÓN 1

La primera opción es utilizar la función `guides()` y pasar la función `guides_legend()` a `fill` o `color`, dependiendo del diagrama.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  guides(fill = guide_legend(title = "Título"))
```

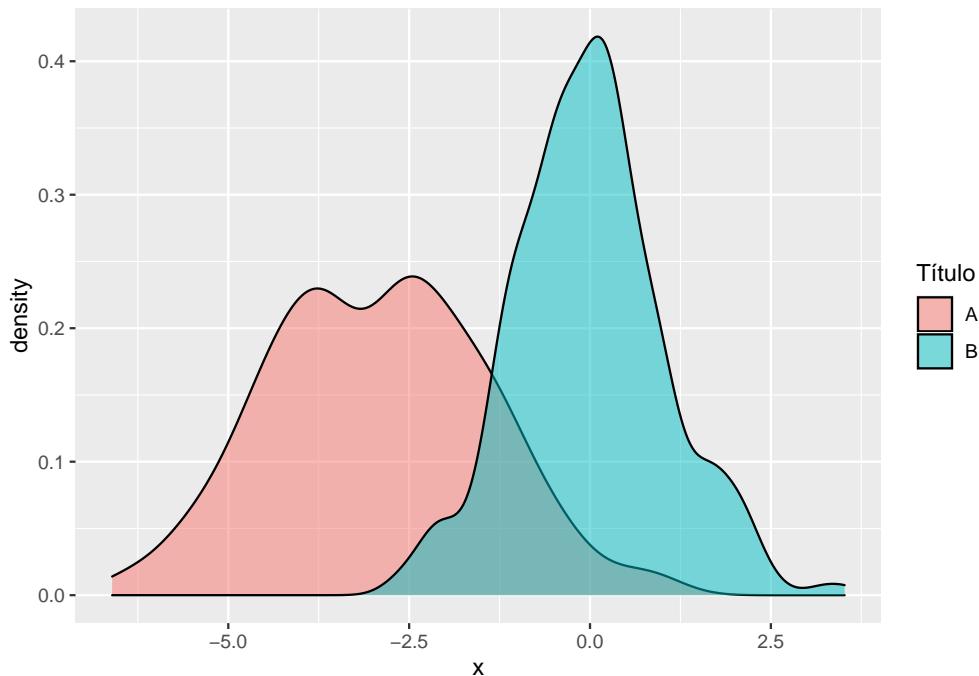


6.3.2 OPCIÓN 2

La segunda opción es pasar el nuevo título al argumento `fill` o `color` de la función `labs()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  labs(fill = "Título")
```

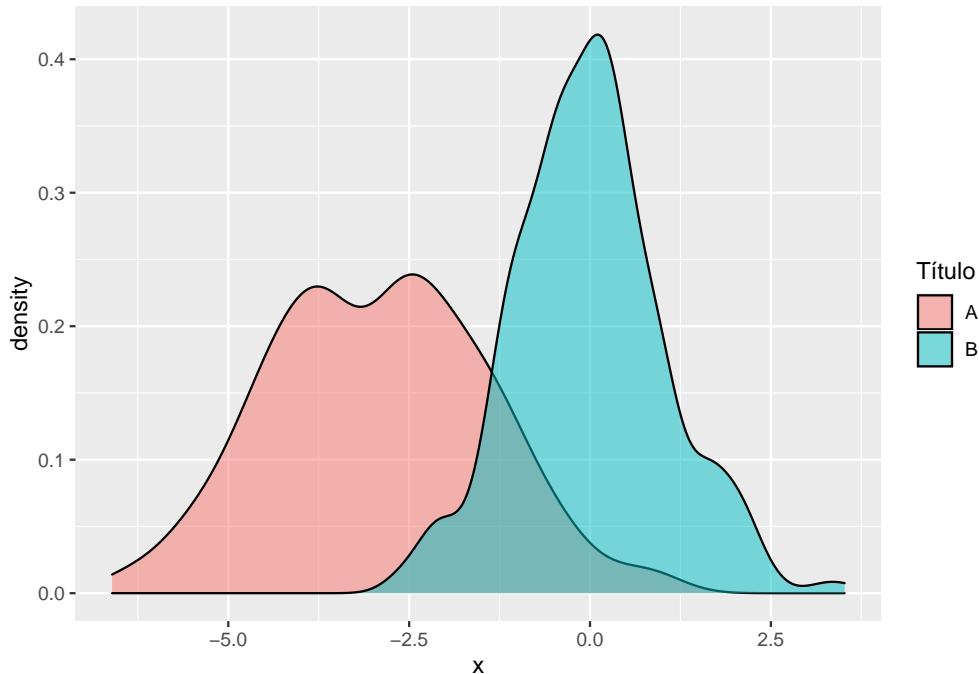


6.3.3 OPCIÓN 3

La tercera opción es pasar el nuevo título al argumento `name` de la correspondiente función de la forma `scale_x_y`, como por ejemplo `scale_fill_discrete()` o `scale_color_discrete()`, si se están usando en el diagrama.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  scale_fill_discrete(name = "Título")
```

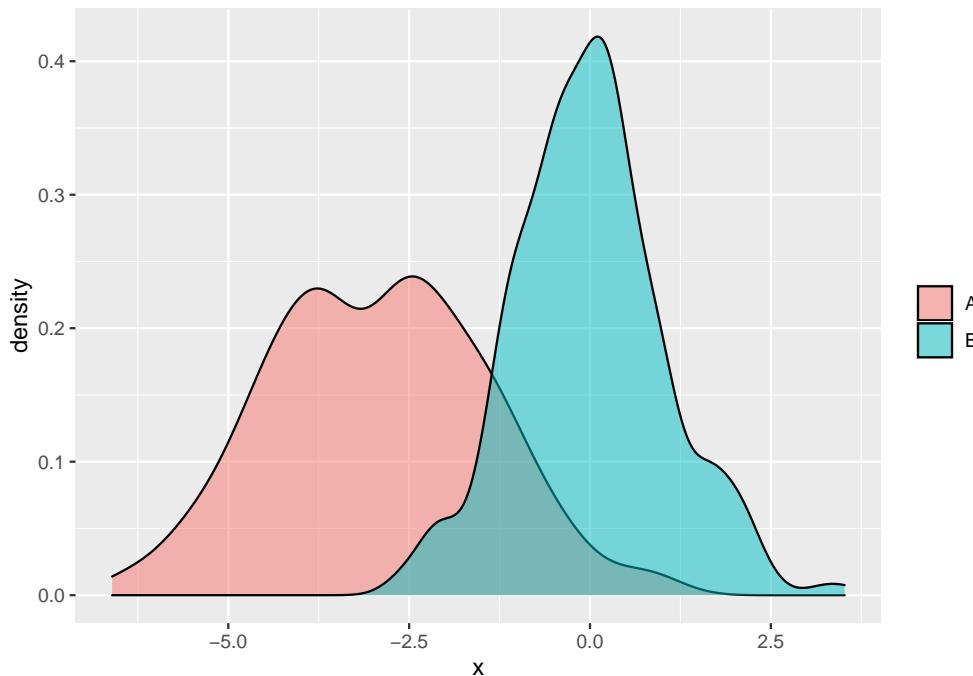


6.3.4 ELIMINAR EL TÍTULO

Por último, si se prefiere eliminar el título de la leyenda, tan solo se pasa la función `element_blank()` al argumento `legend.title` de la función `theme()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.title = element_blank())
```



6.4 CAMBIAR O REORDENAR LAS ETIQUETAS DE LA LEYENDA

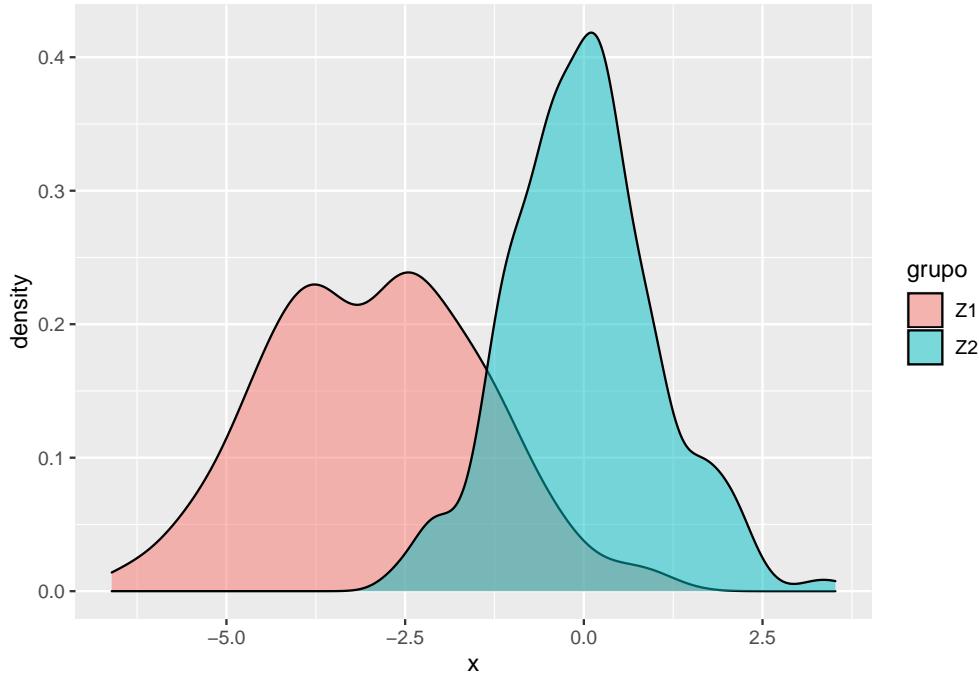
Si se quiere cambiar los nombres de los grupos de la leyenda, se puede personalizar la columna del data frame que representa los grupos.

Otra opción es pasar las nuevas etiquetas al argumento `labels` de las funciones `scale_color_hue` o `scale_fill_hue` para modificar solo las etiquetas o utilizar las funciones `scale_color_discrete` o `scale_fill_discrete` si solo se quiere y se desea cambiar los colores.

6.4.1 NUEVAS ETIQUETAS DE LOS GRUPOS DE LA LEYENDA

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  scale_fill_hue(labels = c("Z1", "Z2"))
```



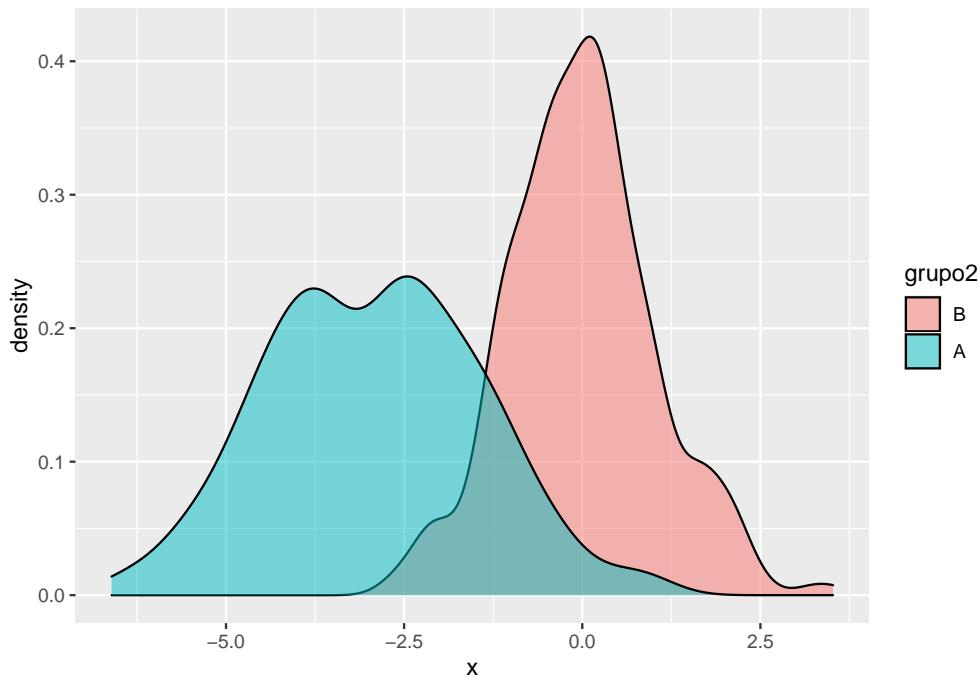
6.4.2 REORDENAR LAS ETIQUETAS

En caso de necesitar reordenar las etiquetas de la leyenda, se tendrá que ordenar la variable de tipo factor. En este ejemplo se ha creado una nueva variable con el nuevo orden.

```
# install.packages("ggplot2")
library(ggplot2)

# Crea una nueva variable con otros niveles
df$grupo2 = factor(df$grupo,
                    levels = c("B", "A"))

ggplot(df, aes(x = x, fill = grupo2)) +
  geom_density(alpha = 0.5)
```



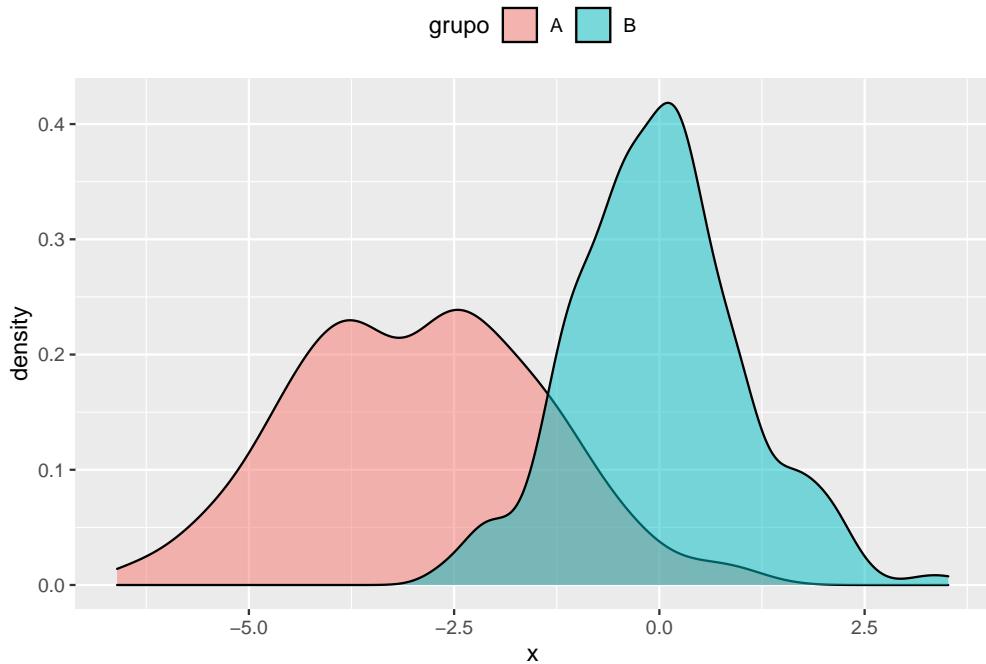
6.5 CAMBIAR LA POSICIÓN DE LA LEYENDA

Por defecto, la leyenda se generará automáticamente a la derecha del gráfico de `ggplot2`. Sin embargo, haciendo uso del argumento `legend.position` de la función `theme()`, se puede modificar su posición. Los posibles valores son: "right" (por defecto), "top", "left", "bottom" y "none".

6.5.1 ARRIBA ("top")

```
# install.packages("ggplot2")
library(ggplot2)

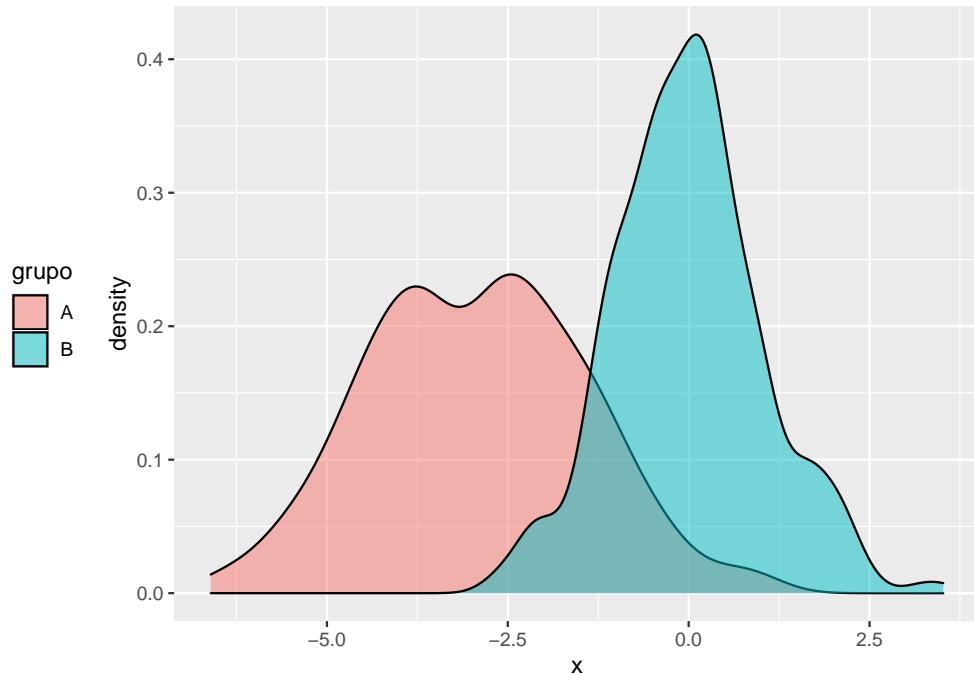
ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "top")
```



6.5.2 IZQUIERDA ("left")

```
# install.packages("ggplot2")
library(ggplot2)

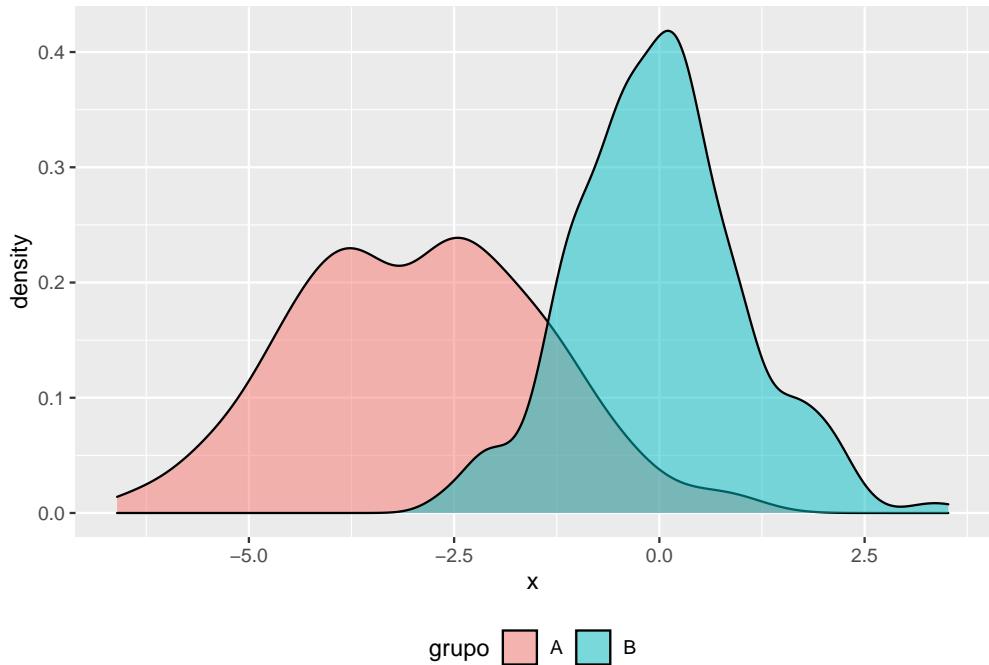
ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "left")
```



6.5.3 DEBAJO ("bottom")

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "bottom")
```



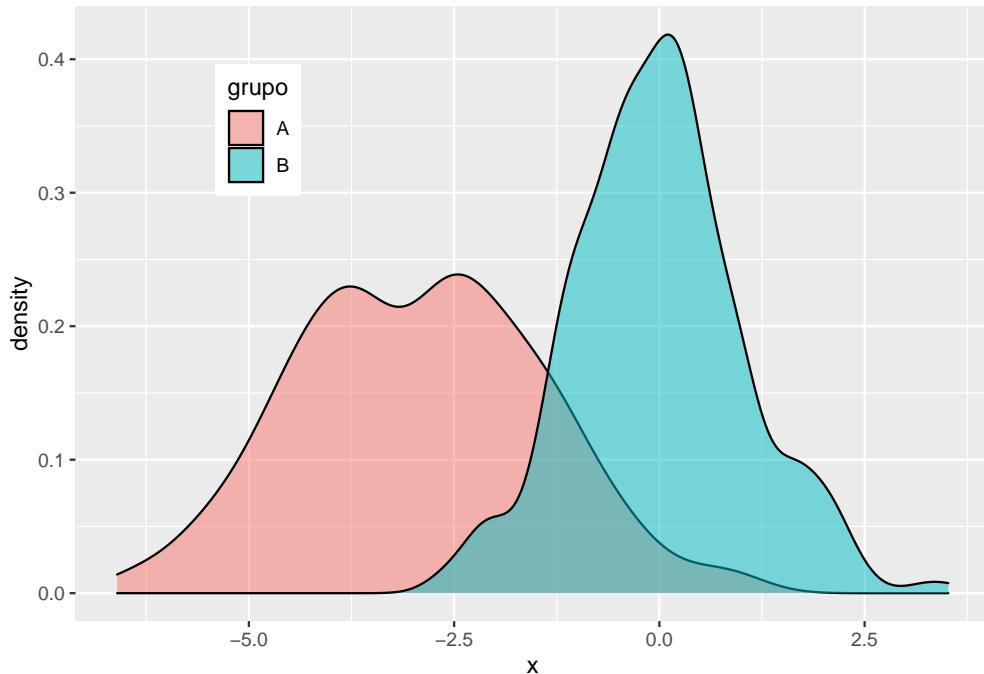
6.5.4 POSICIÓN PERSONALIZADA

Hay que tener en cuenta que incluso se puede establecer una posición personalizada para la leyenda de `ggplot2` y **ponerla dentro del diagrama**. Se tendrá que establecer las **posiciones entre 0 y 1** relativas a los ejes con el argumento `legend.position` de la función `theme()`. Además, se puede cambiar el color de fondo de la leyenda con `legend.background` en caso de ser necesario.¹²

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = c(0.2, 0.8),
        legend.background = element_rect(fill = "white"))
```

¹²De ser necesario se puede revisar tambien los argumentos `legend.justification`, `legend.direction` y `legend.box.just` para una personalización más avanzada.



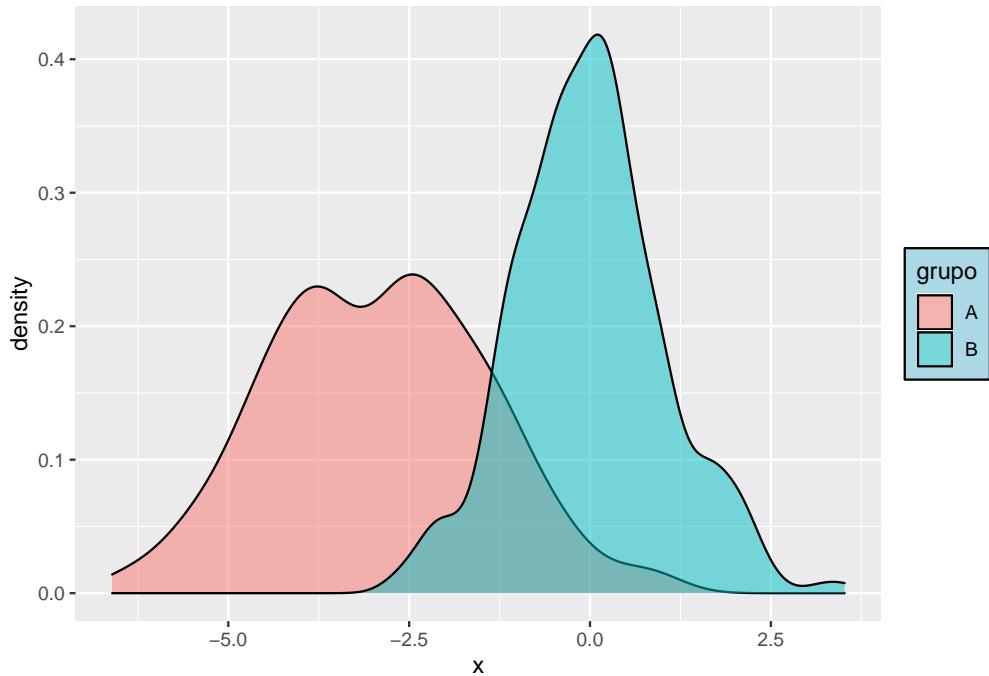
6.6 PERSONALIZACIÓN DE LA LEYENDA

Existen numerosos argumentos que permiten personalizar las leyendas en ggplot2. Se tendrán que usar los argumentos de la función `theme()` que comienza con `legend.` como `legend.title`, `legend.background`, `legend.margin`, entre otros más.

6.6.1 PERSONALIZAR EL COLOR DE FONDO Y DE BORDE DE LA LEYENDA

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.background = element_rect(fill = "lightblue", # Fondo
                                         colour = 1))      # Borde
```

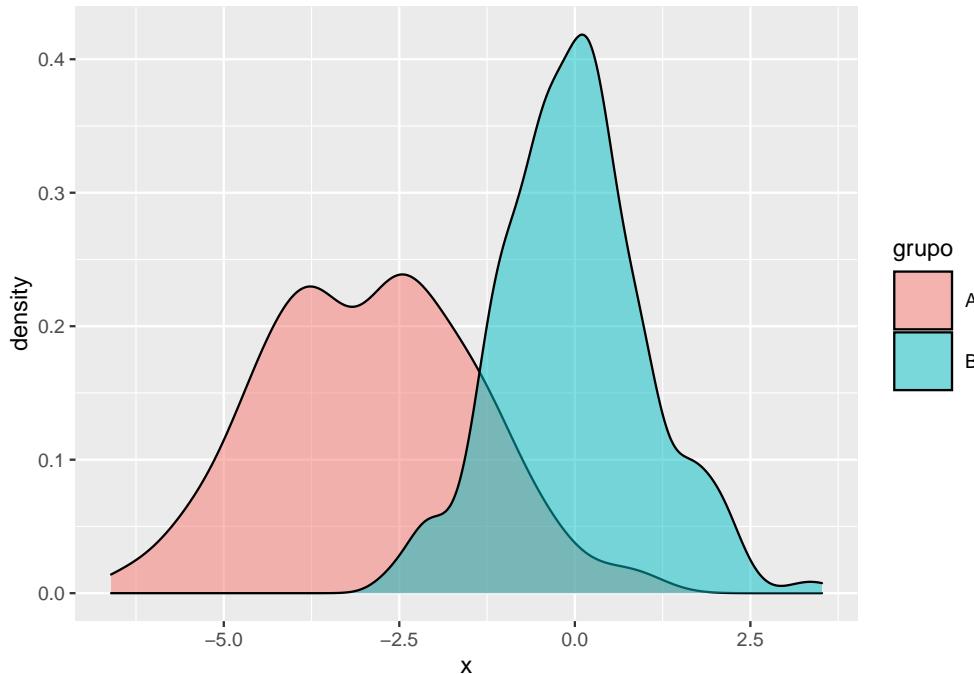


6.6.2 PERSONALIZAR LOS KEYS DE LA LEYENDA

En este ejemplo se va a incrementar el tamaño de las cajas, pero si se tiene, por ejemplo, un gráfico de dispersión, se puede cambiar el tamaño de los símbolos con `guides(color = guide_legend(override.aes = list(size = 4)))`, siendo 4 el nuevo tamaño.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.key.size = unit(1, units = "cm"))
```



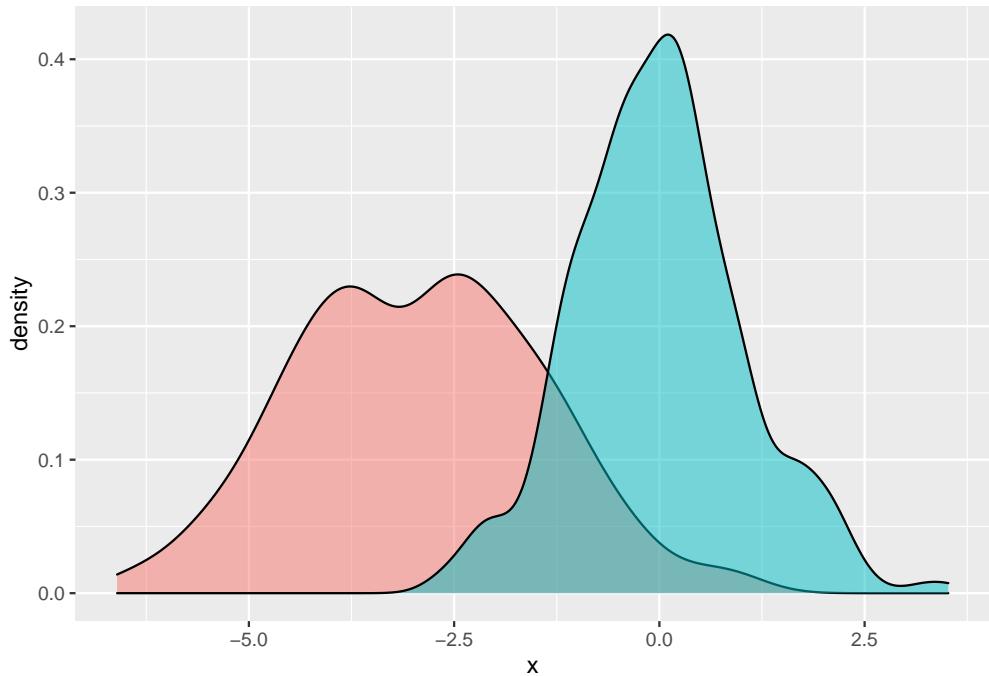
6.7 ELIMINAR LA LEYENDA

Por último, en caso de que se quiera deshacer de la leyenda, por defecto se puede establecer su posición como `none`, tal y como se muestra en el siguiente ejemplo:

6.7.1 ELIMINAR LA LEYENDA

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "none")
```

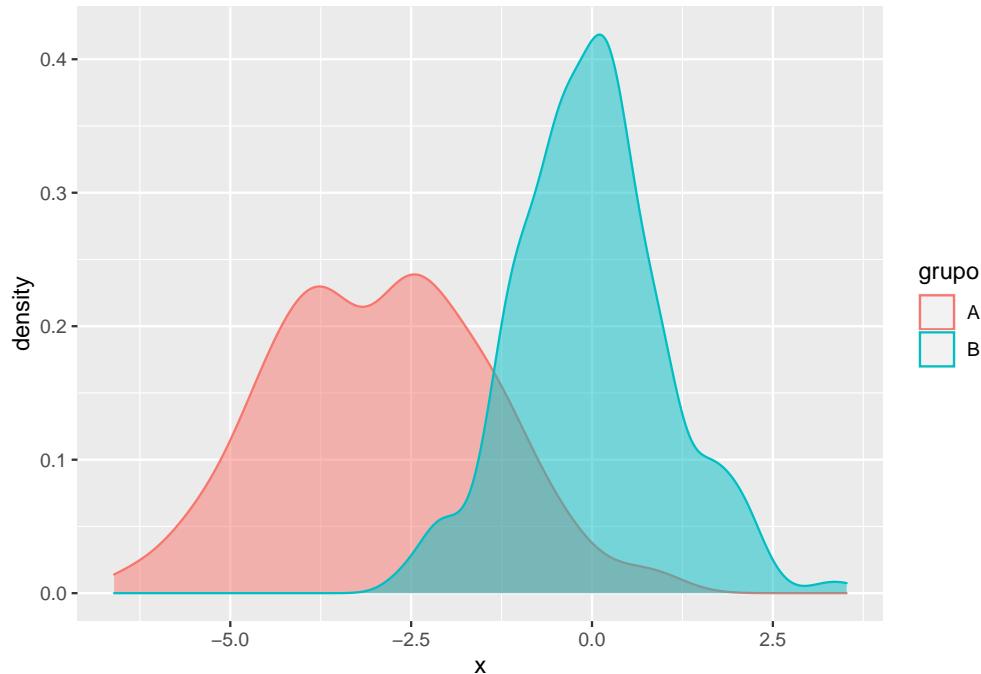


Además, en caso de que el diagrama tenga varias leyendas o que se haya mapeado una variable a vaarios argumentos, se puede deshacer de parte de la leyenda estableciendo ese argumento como "none" dentro de la función `guides()`. En el ejemplo que sigue, se está eliminando la leyenda que corresponde al `fill` del diagrama, manteniendo solamente la leyenda que corresponde a `color`.

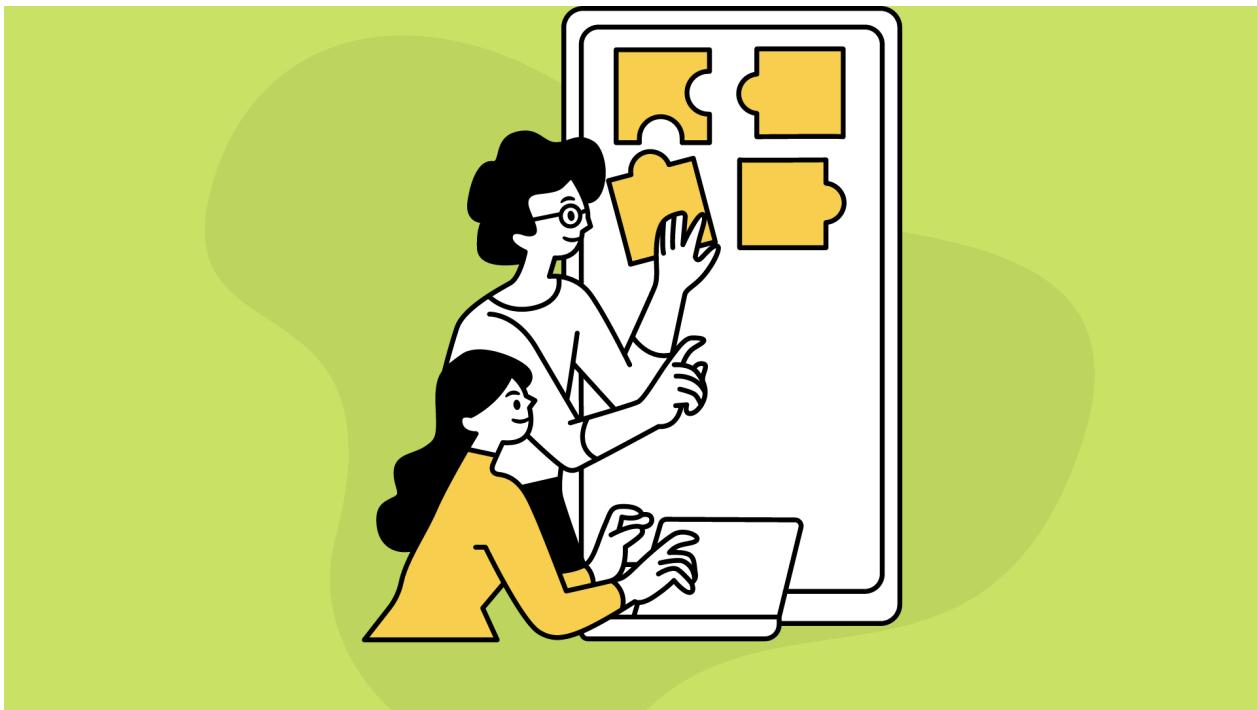
6.7.2 ELIMINAR UNA PARTE DE LA LEYENDA

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo, color = grupo)) +
  geom_density(alpha = 0.5) +
  guides(fill = "none")
```



CAPÍTULO 7: SISTEMAS DE COORDENADAS EN `ggplot2`



Los sistemas de coordenadas en `ggplot2` se pueden dividir en dos categorías: sistemas lineales (`coord_cartesian()`, `coord_flixed()`, `coord_flip()`) y no lineales (`coord_trans()`, `coord_polar()`, `coord_quickmap()`, `coord_map()`). Estos sistemas se revisarán a lo largo de este capítulo.

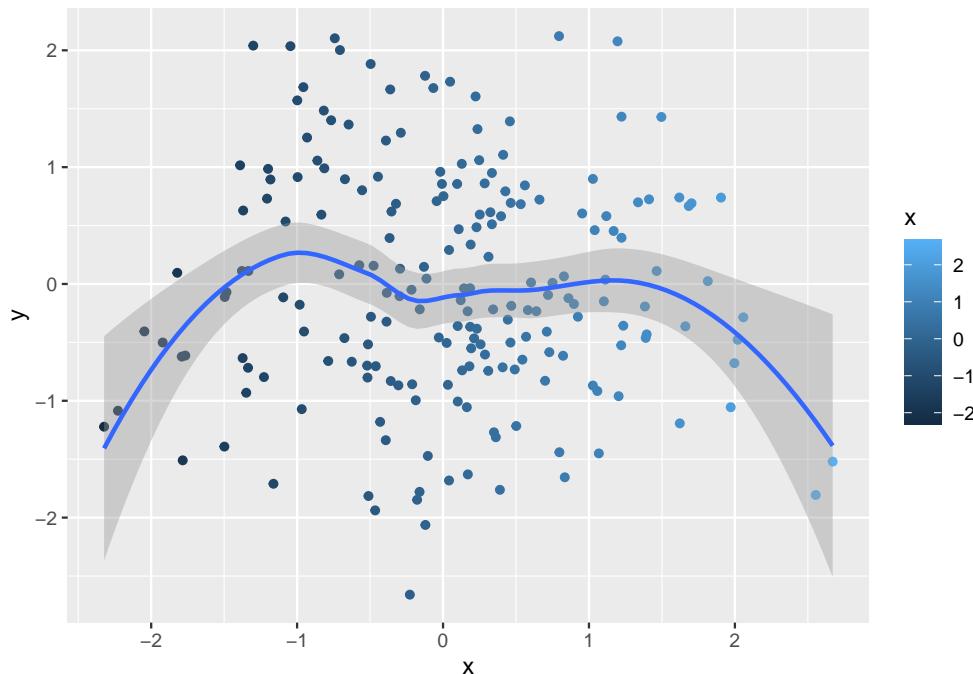
7.1 COORDENADAS CARTESIANAS CON `coord_cartesian()`

Por defecto, los diagramas de `ggplot2` tienen coordenadas cartesianas. Sin embargo, al función `coord_cartesian()` es muy útil para hacer **zoom a los gráficos**, porque si se usa `scale_x_continuous()` o `scale_y_continuous()` los datos subyacentes cambiarán y por lo tanto también lo harán las estadísticas calculadas, tal y como se muestra a continuación:

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1000)
df = data.frame(x = rnorm(200),
                 y = rnorm(200))

ggplot(df, aes(x = x, y = y, col = x)) +
  geom_point() +
  geom_smooth()
```



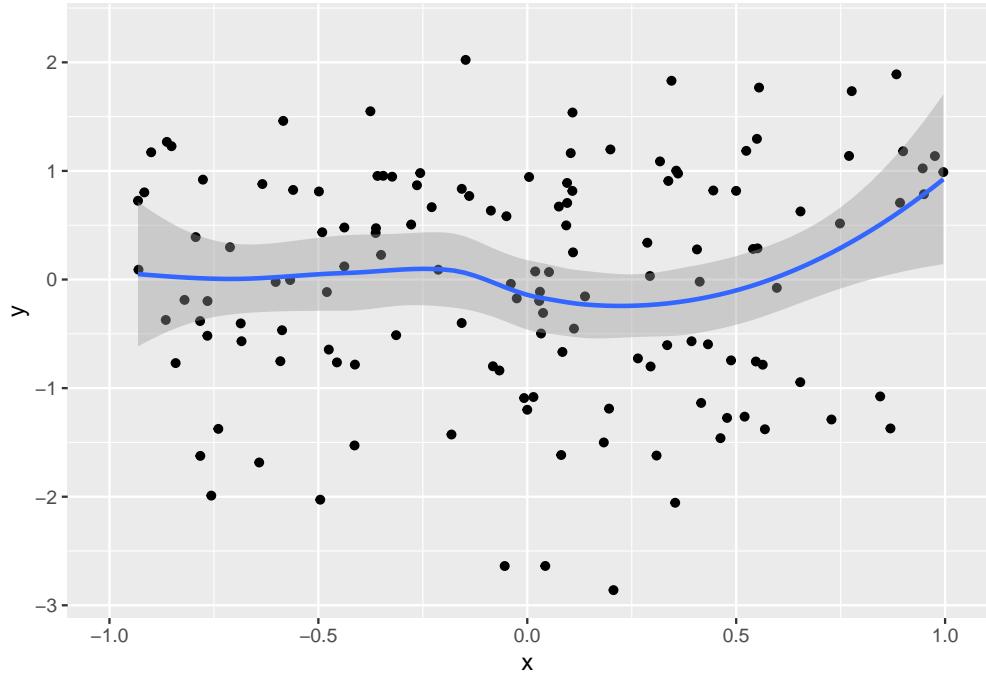
Usando la función `scale_x_continuous()` para hacer zoom, se modifican los datos y, por lo tanto, la estimación de la suavización.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(50)
df = data.frame(x = rnorm(200),
                 y = rnorm(200))

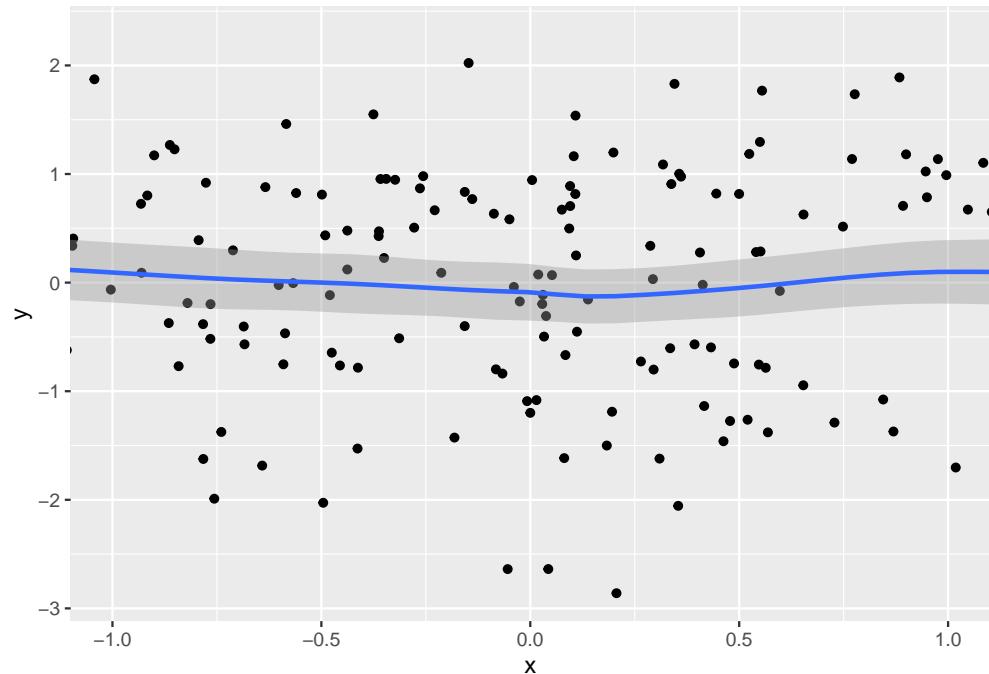
p = ggplot(df, aes(x = x, y = y)) +
```

```
geom_point() +  
geom_smooth()  
  
p + scale_x_continuous(limits = c(-1, 1))
```



Sin embargo, si se utiliza la función `coord_cartesian()` se puede establecer los límites de los ejes con `xlim` e `ylim` sin modificar las estimaciones originales, tan solo haciendo zoom.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
# Datos  
set.seed(50)  
df = data.frame(x = rnorm(200),  
                 y = rnorm(200))  
  
p = ggplot(df, aes(x = x, y = y)) +  
    geom_point() +  
    geom_smooth()  
  
p + coord_cartesian(xlim = c(-1, 1))
```



7.2 COORDENADAS FIJAS (MISMA ESCALA) CON `coord_fixed()`

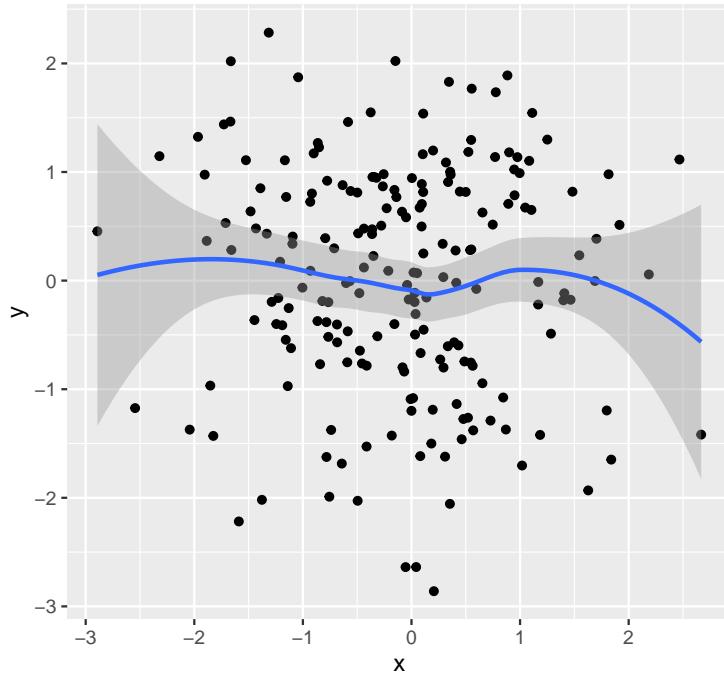
La función `coord_fixed()` es muy útil en caso de que se quiera una relación de aspecto fijo en un diagrama a pesar del tamaño del dispositivo gráfico, esto es, una unidad a lo largo del eje X será la misma unidad a lo largo del eje Y.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(50)
df = data.frame(x = rnorm(200),
                 y = rnorm(200))

p = ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()

p + coord_fixed()
```



7.3 ROTAR LOS EJES CON `coord_flip()`

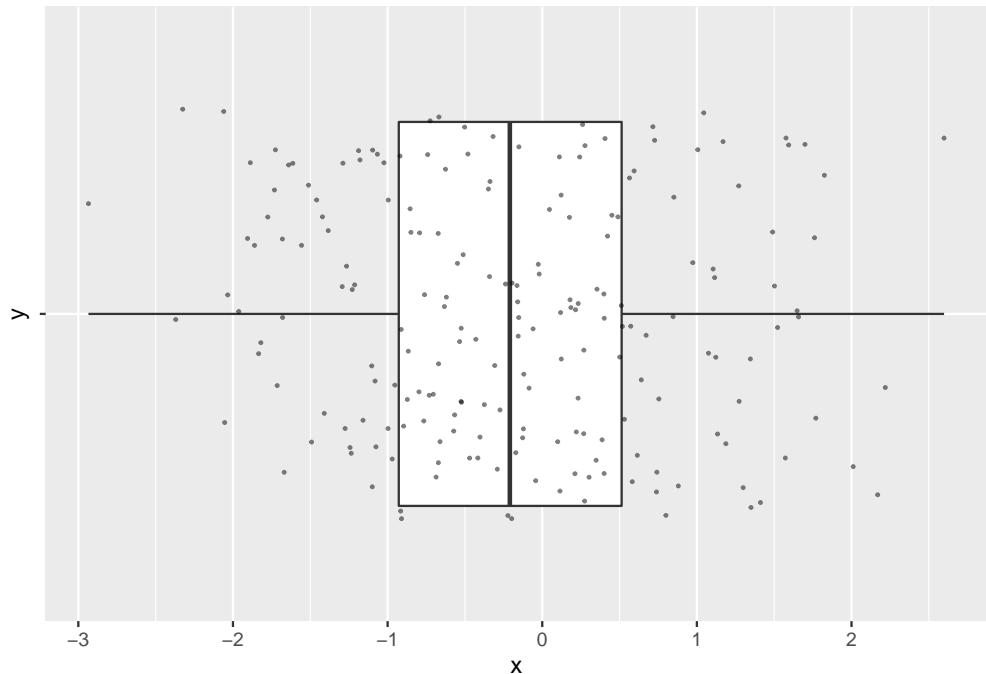
La función `coord_flip()` permite rotar los ejes en ggplot2, de modo que si se tiene un gráfico vertical se puede crear su versión horizontal y viceversa. Esto es especialmente útil en el caso de diagramas de cajas, diagramas de violín, de barras, entre otros.

7.3.1 POR DEFECTO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(30)
df = data.frame(x = rnorm(200))

ggplot(df, aes(x = x, y = "")) +
  geom_boxplot() +
  geom_jitter(size = 0.4, alpha = 0.5)
```

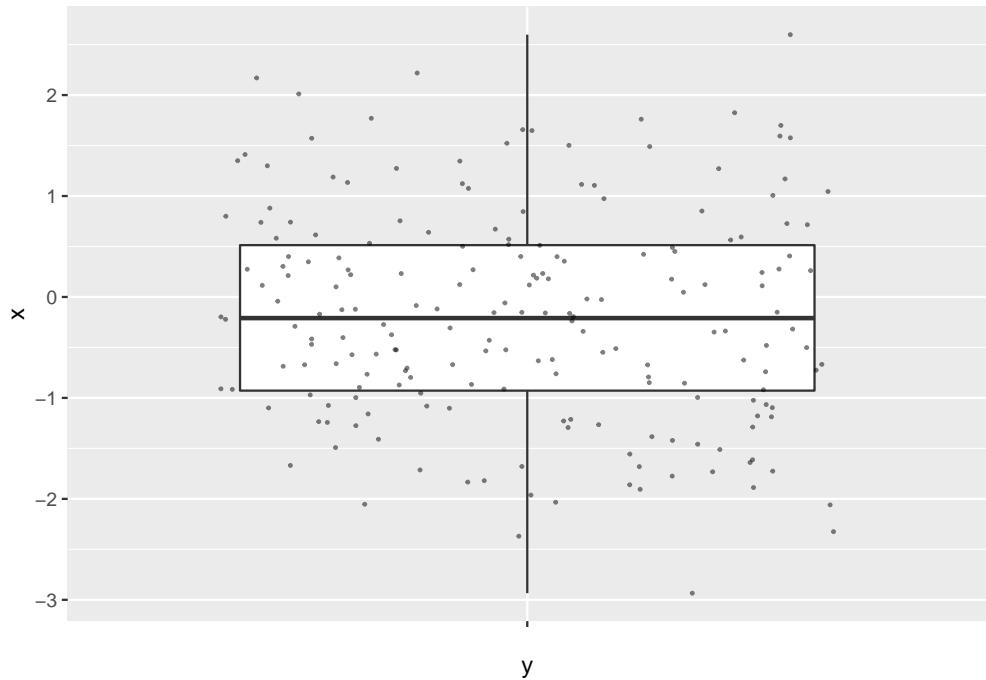


7.3.2 EJES ROTADOS

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(30)
df = data.frame(x = rnorm(200))

ggplot(df, aes(x = x, y = "")) +
  geom_boxplot() +
  geom_jitter(size = 0.4, alpha = 0.5) +
  coord_flip()
```



7.4 TRANSFORMACIONES CON `coord_trans()`

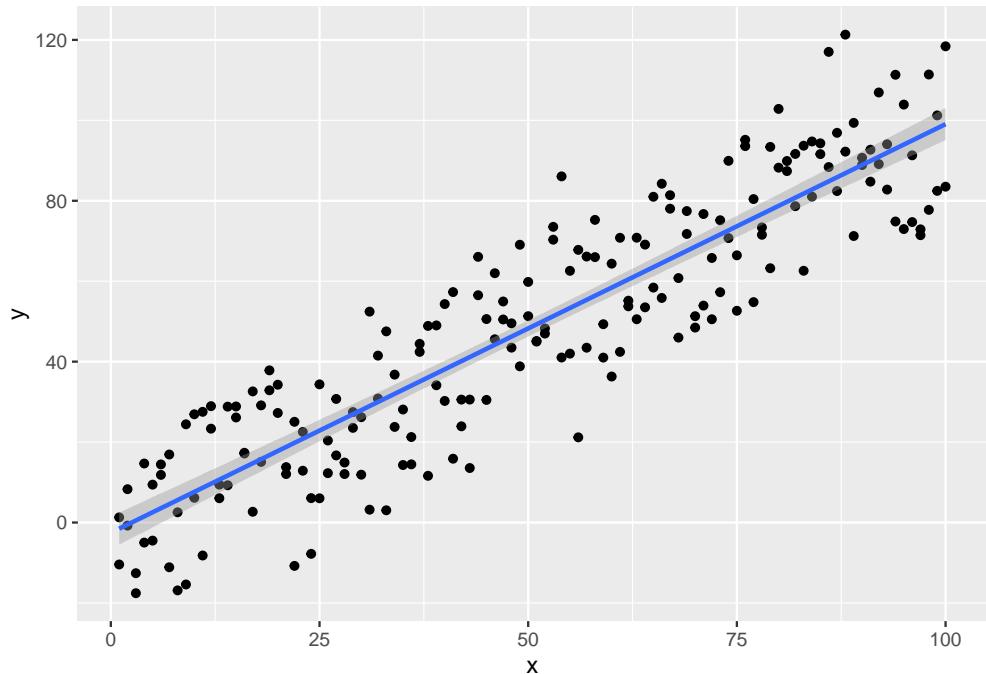
La función `coord_trans()` crea sistemas de coordenadas cartesianas transformadas. Hay que tener en cuenta que **utilizar esta función no es lo mismo que transformar a escala**, ya que cuando se usa la función `coord_trans()` la transformación ocurre después de los cálculos estadísticos, afectando a la apariencia de los geoms. Consideremos el siguiente diagrama:

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
df = data.frame(x = 1:100,
                 y = 1:100 + rnorm(200, sd = 15))

p = ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm")

p
```



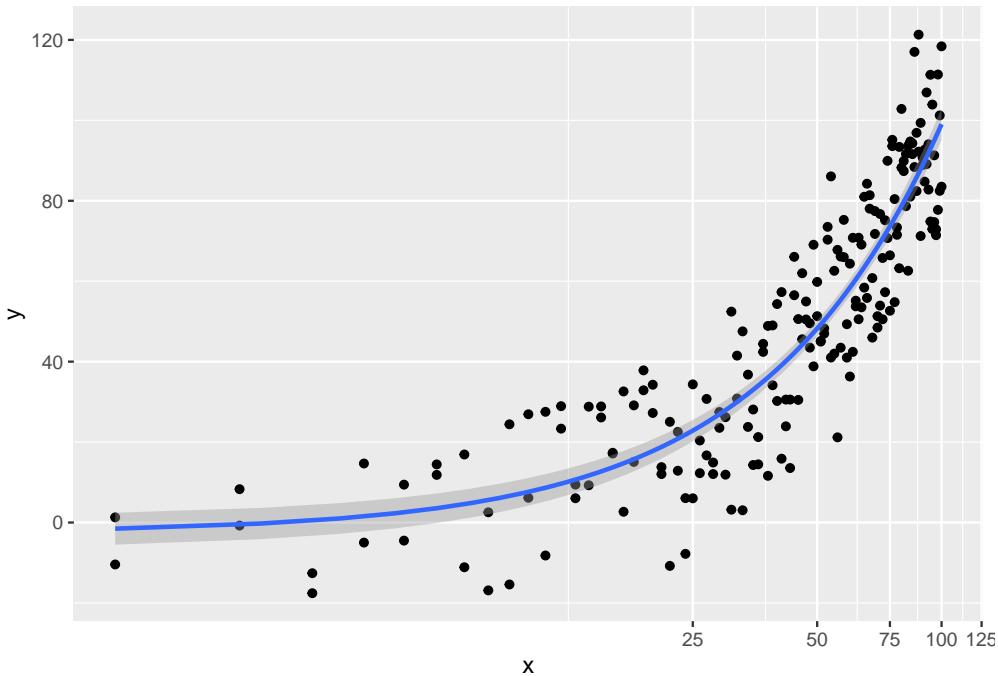
El siguiente ejemplo transforma el eje X (argumento `x`) con la función `log`, de modo que la estimación lineal se convertirá en una curva. Hay que recordar que se puede pasar cualquier función a los ejes siempre y cuando tenga sentido.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
df = data.frame(x = 1:100,
                 y = 1:100 + rnorm(200, sd = 15))

p = ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm")

p + coord_trans(x = "log")
```



7.5 COORDENADAS POLARES CON `coord_polar()`

Las coordenadas polares se pueden aplicar usando la función `coord_polar()`. Este tipo de coordenadas se usan habitualmente para los **diagramas de sectores**¹³<https://r-charts.com/es/parte-todo/diagrama-sectores-ggplot2/> (que no son más que gráficos de barras apiladas en coordenadas polares), rosa de los vientos, gráficos de radar, gráficos bullseye, entre otros más.

Hay que tener en cuenta que por defecto el ángulo se mapea a la variable `x`, pero se puede establecer el argumento `theta = "y"` para manjar el ángulo a la variable `y`. También se puede modificar la dirección del diagrama con el argumento `direction` (establece `-1` para el sentido contrario a las agujas del reloj). Para esto, se crea el siguiente ejemplo para aclarar lo anterior:

```
# install.packages("ggplot2")
library(ggplot2)

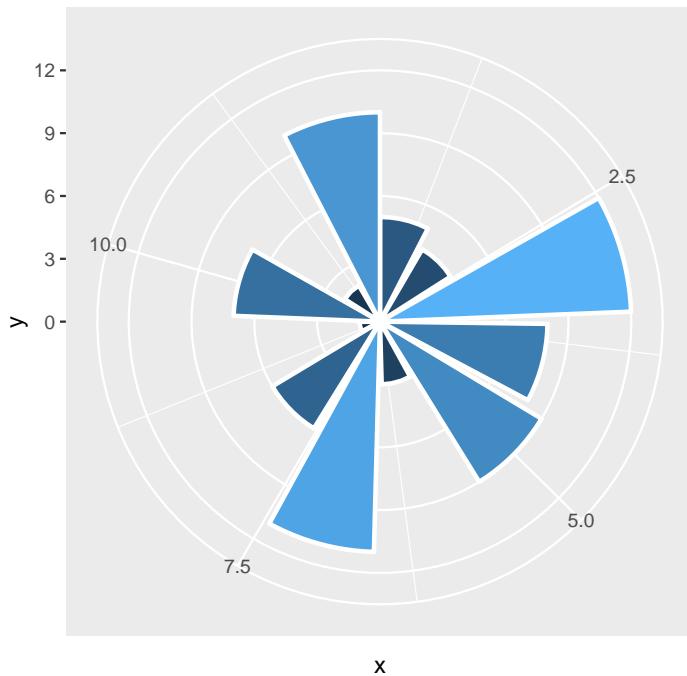
# Datos
set.seed(156)
df = data.frame(x = 1:12,
                 y = sample(1:12))

p = ggplot(df, aes(x = x, y = y, fill = y)) +
```

¹³Se puede acceder al enlace al dar click en las letras en negritas.

```
geom_bar(stat = "identity", color = "white",
lwd = 1, show.legend = FALSE)

p + coord_polar()
```

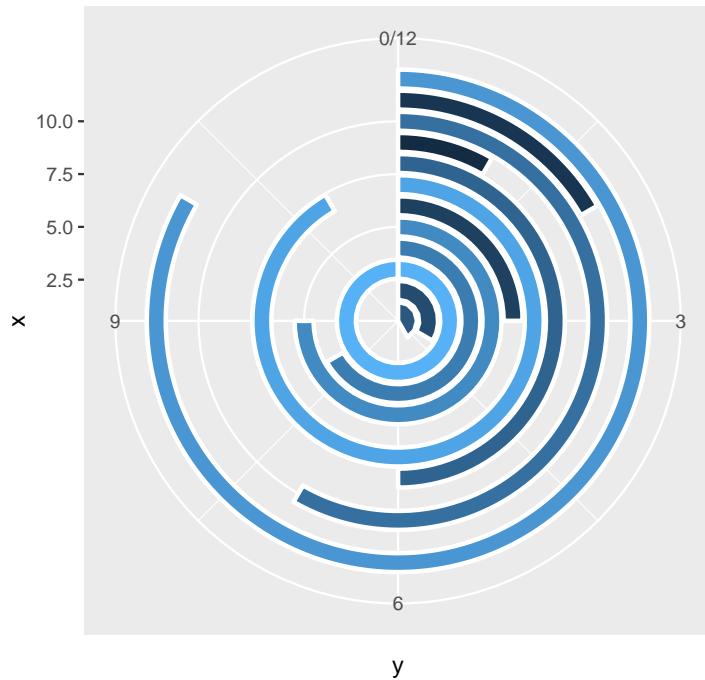


```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(156)
df = data.frame(x = 1:12,
                 y = sample(1:12))

p = ggplot(df, aes(x = x, y = y, fill = y)) +
  geom_bar(stat = "identity", color = "white",
           lwd = 1, show.legend = FALSE)

p + coord_polar(theta = "y")
```



7.6 PROYECCIONES DE MAPAS CON `coord_quickmap()` Y `coord_map()`

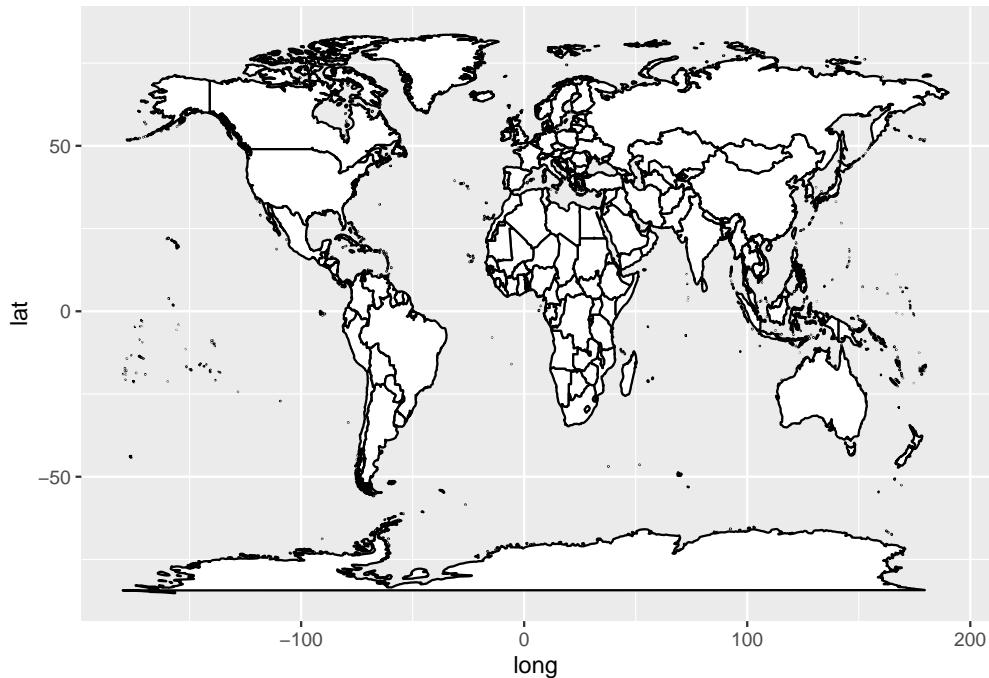
El último sistema de coordenadas está relacionada con proyecciones de mapas. Se tendrá que tener instalado el paquete `mapproj` para aplicar las proyecciones.

Por una parte, la función `coord_map()` requiere un tiempo de computación considerable, pero aproximará la proyección a lo máximo que sea posible. Hay que tener en cuenta que existen **muchas proyecciones diferentes disponibles**. Solamente se escribe `?mapproj::mapproject` para ver la lista completa.

7.6.1 MAPA POR DEFECTO

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")
library(mapproj)

ggplot(map_data("world"),
       aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)
```



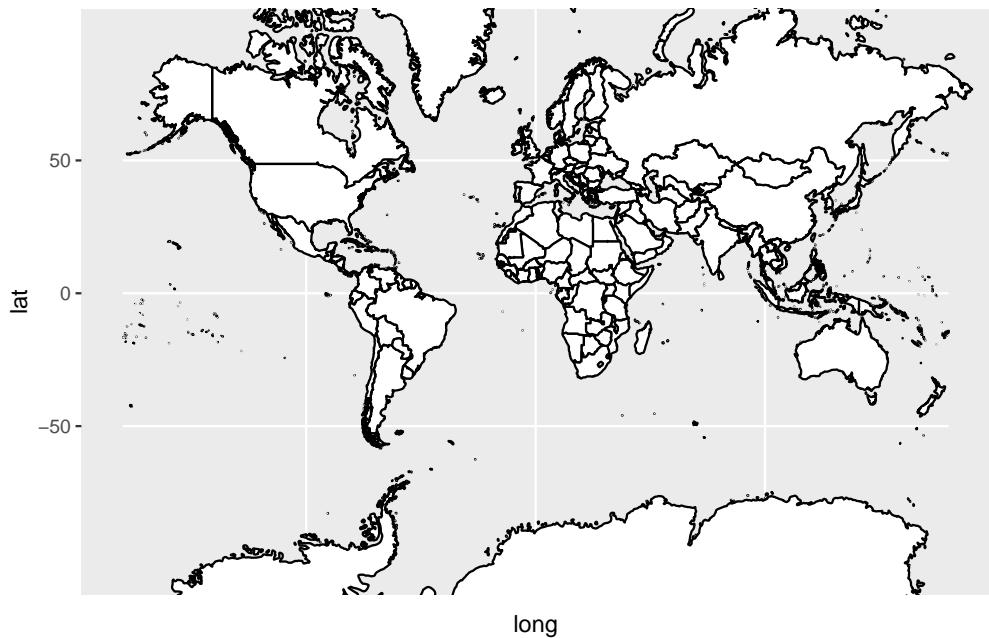
7.6.2 PROYECCIÓN MERCATOR

La proyección por defecto es la proyección Mercator. Hay que tener en cuenta que se han modificado los límites del eje X debido a que la función produce unas líneas horizontales no deseadas. Si se quiere crear un diagrama como éste, se utiliza la función `coord_sf()` del paquete `sf` como una buena opción.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")
library(mapproj)

p = ggplot(map_data("world"),
           aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)

p + coord_map(xlim = c(-180, 180))
```



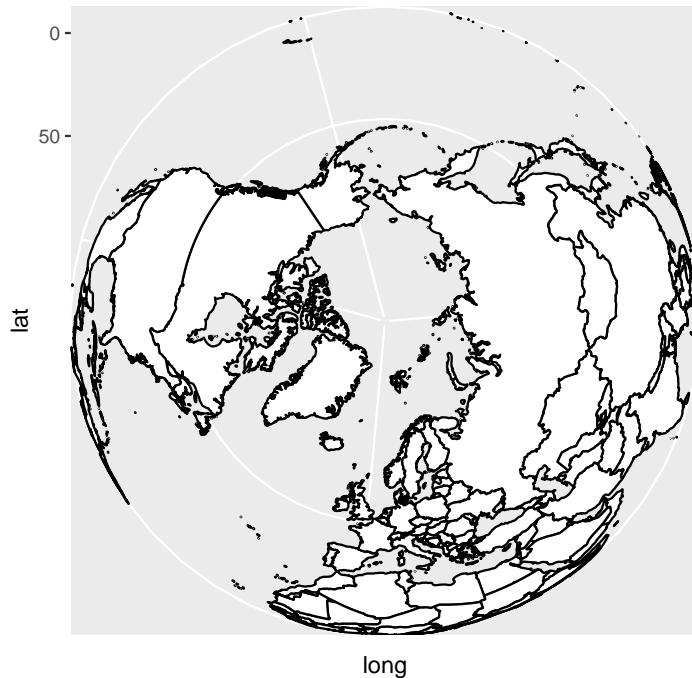
```
# p + sf::coord_sf() # Mejor opción
```

7.6.3 PROYECCIÓN ORTOGRÁFICA

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")
library(mapproj)

p = ggplot(map_data("world"),
           aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)

p + coord_map("orthographic")
```

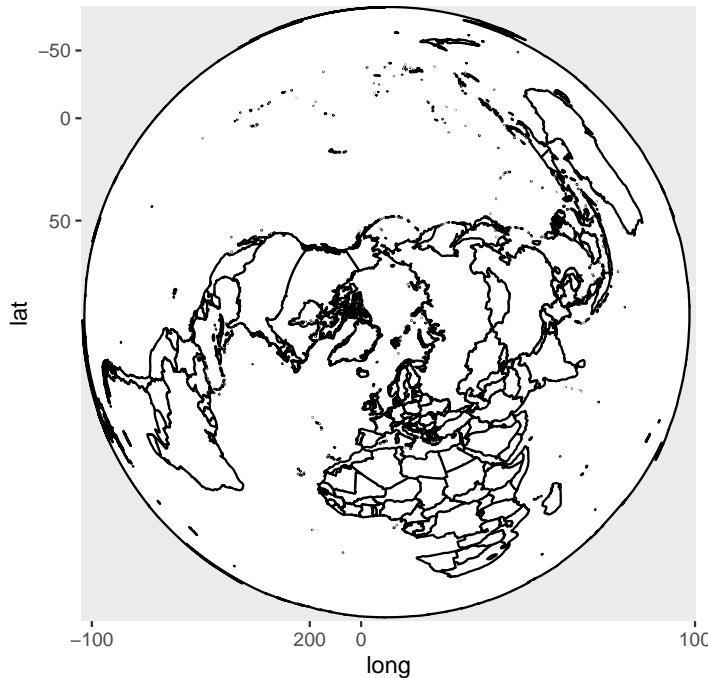


7.3.4 OJO DE PEZ

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")
library(mapproj)

p = ggplot(map_data("world"),
           aes(long, lat, group = group)) +
    geom_polygon(fill = "white", colour = 1)

p + coord_map("fisheye",
              n = 4) # Índice de refracción
```

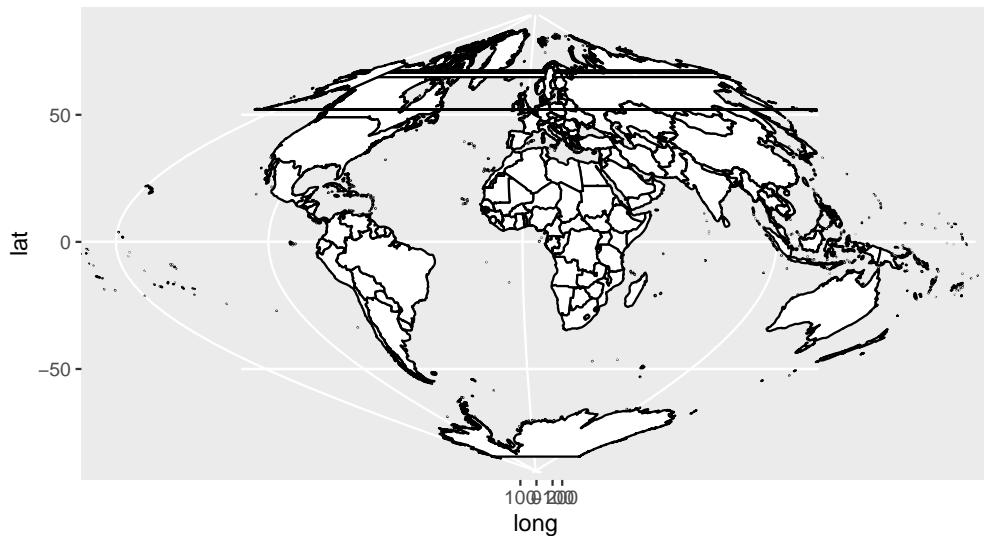


7.3.5 SINUSOIDAL

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")
library(mapproj)

p = ggplot(map_data("world"),
           aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)

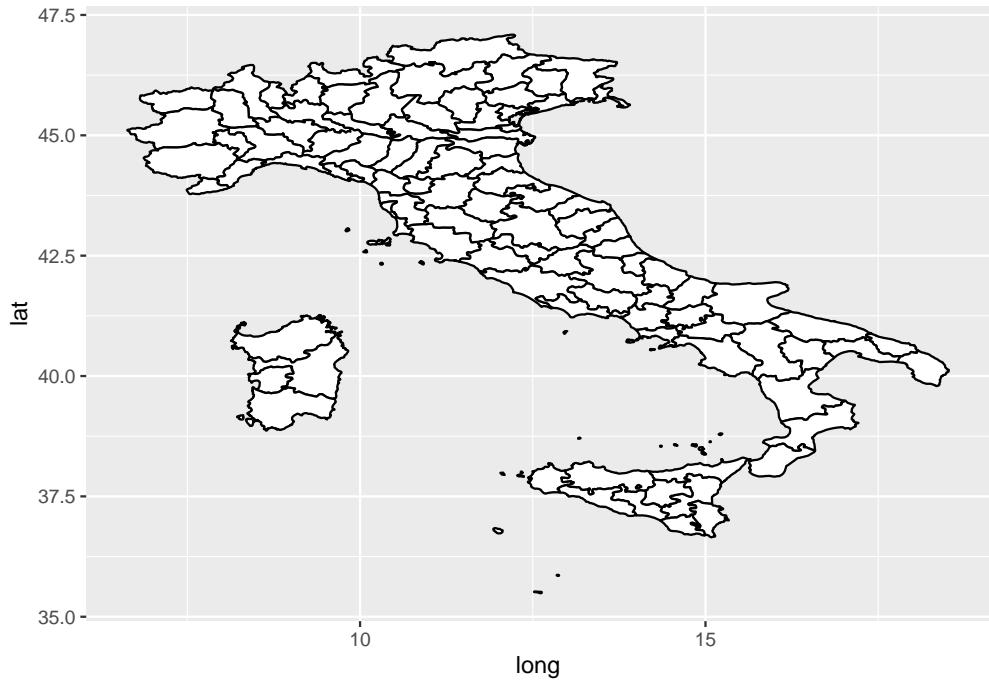
p + coord_map("sinusoidal")
```



Por otro lado, `coord_quickmap()` tambipen soluciona la proyección del mapa, tal y como se muestra a continuación.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(map_data("italy"),
       aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)
```



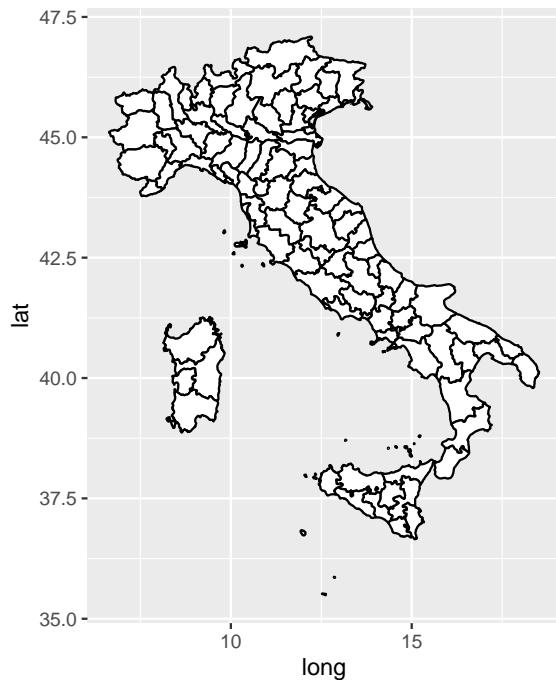
La diferencia es que **esta función es una aproximación rápida** para preservar líneas rectas que funciona mejor para áreas pequeñas cercanas al ecuador. Hay que tener en cuenta que **también se puede aplicar cualquier proyección que se desee** con el argumento `projection` de la función, con ejemplos anteriores.

7.3.6 ARREGLANDO LA PROYECCIÓN

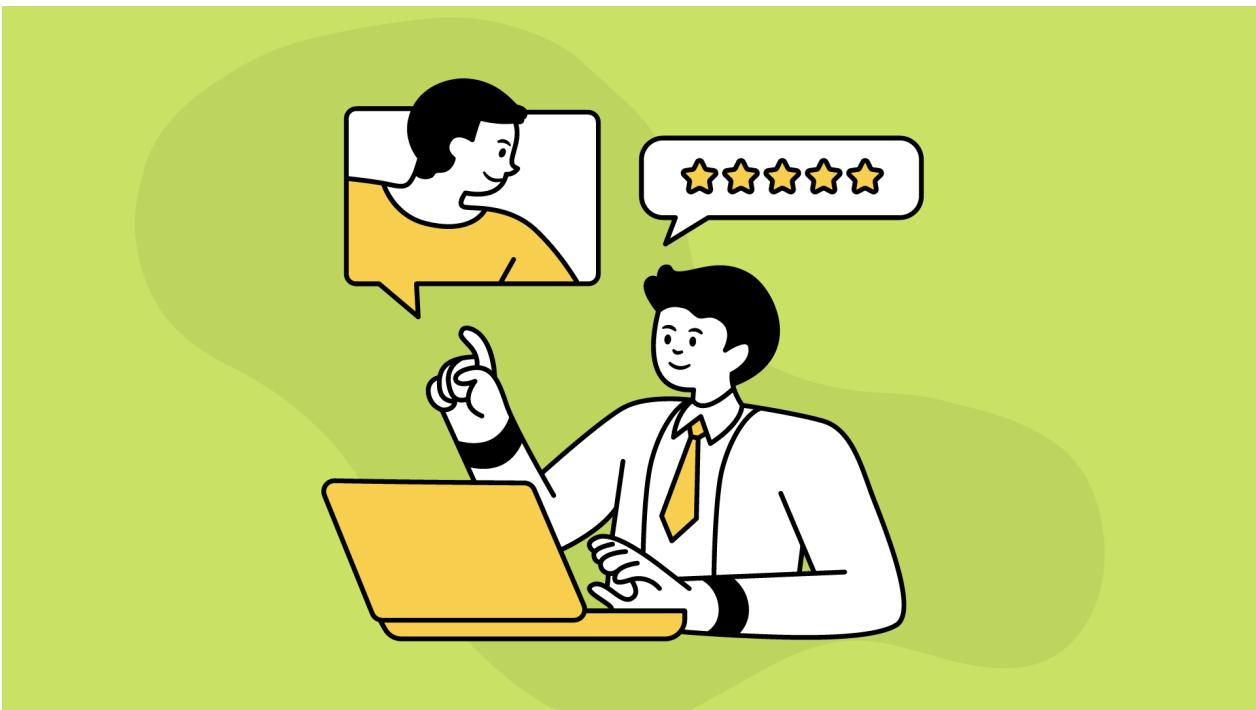
```
# install.packages("ggplot2")
library(ggplot2)

p = ggplot(map_data("italy"),
           aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = 1)

p + coord_quickmap()
```



CAPÍTULO 8: LÍNEAS DE REFERENCIA, SEGMENTOS, CURVAS Y FLECHAS EN `ggplot2`



El paquete de `ggplot2` proporciona varias funciones para agregar capas a los diagramas como líneas de referencia (funciones `geom_vline()`, `geom_hline()`, y `geom_abline()`), segmentos (`geom_segment()`), curvas (`geom_curve()`) y flechas (`arrows`).

8.1 LÍNEAS VERTICALES CON `geom_vlines()`

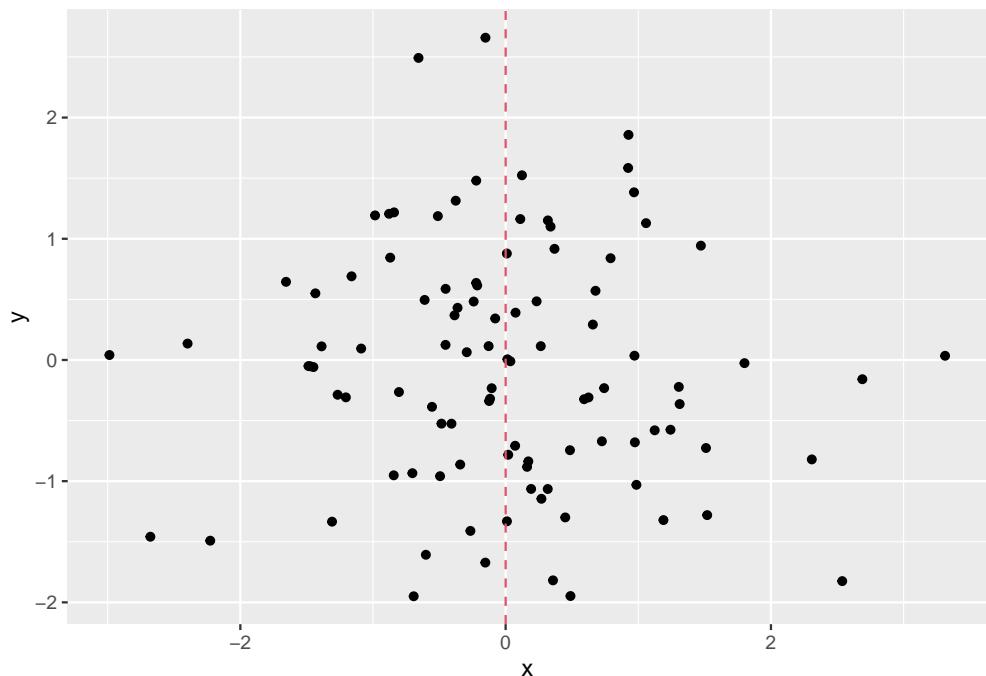
Considerando que se tiene un diagrama hecho con `ggplot2`, se puede crear una nueva capa con `geom_vline()` para **agregar unas múltiples líneas verticales**. Tan solo se tiene que pasar un único valor o un vector numérico al argumento `xintercept` de la función donde se quiere que se muestren las líneas.

Hay que tener en cuenta que se puede personalizar argumentos gráficos como `linetype` (tipo de línea), `color` o `lwd` (grosor de línea).

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_vline(xintercept = 0,
             linetype = 2,
             color = 2)
```



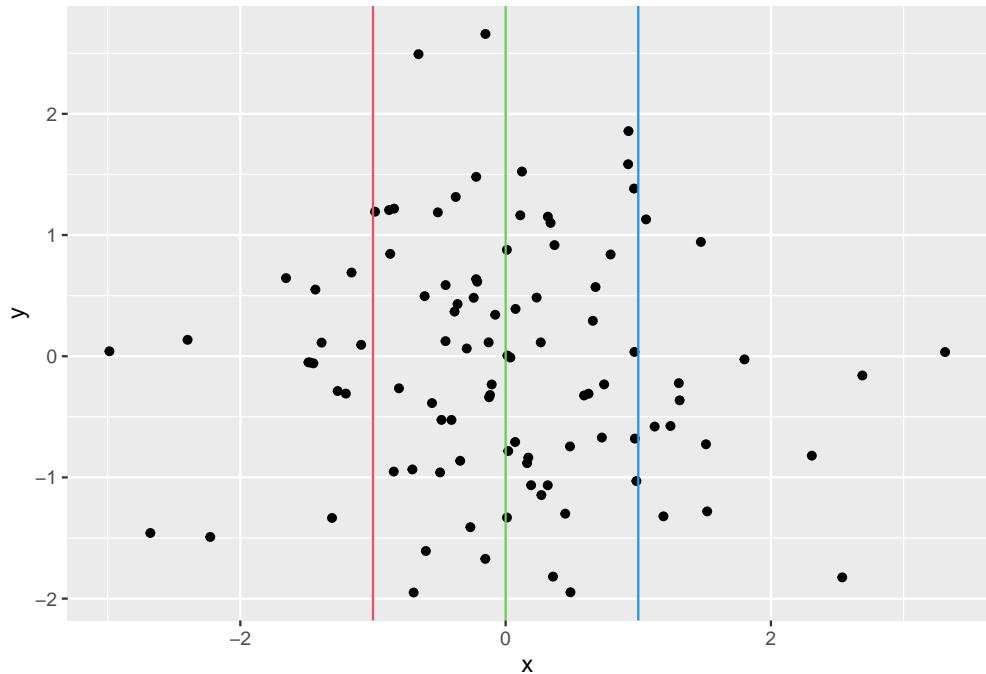
8.1.1 AGREGAR VARIAS LÍNEAS A LA VEZ

Hay que tener en cuenta que todas las funciones de este capítulo permiten hacer esto.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_vline(xintercept = -1:1,
             linetype = 1,
             color = 2:4)
```



8.2 LÍNEAS HORIZONTALES CON `geom_hline`

La función `geom_hline` se comporta de la misma manera que `geom_vline`. La única diferencia es que esta función agregará líneas horizontales, por que se tendrá que especificar el argumento `yintercept` en lugar de `xintercept`.

```
# install.packages("ggplot2")
library(ggplot2)

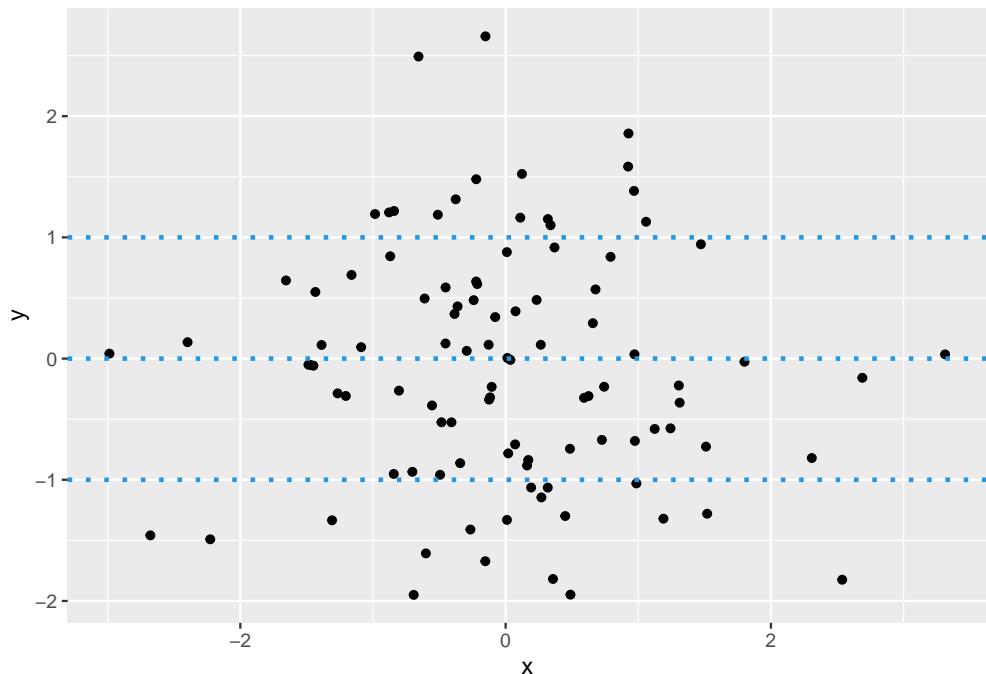
# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
```

```

y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_hline(yintercept = -1:1,
             linetype = 3,
             color = 4,
             lwd = 1)

```



8.2.1 LÍNEAS DIAGONALES CON `geom_abline()`

La función `geom_abline()` permite agregar líneas diagonales en la base a un intercepto y una pendiente. El siguiente ejemplo muestra una línea diagonal con intercepto 0 y pendiente 1.

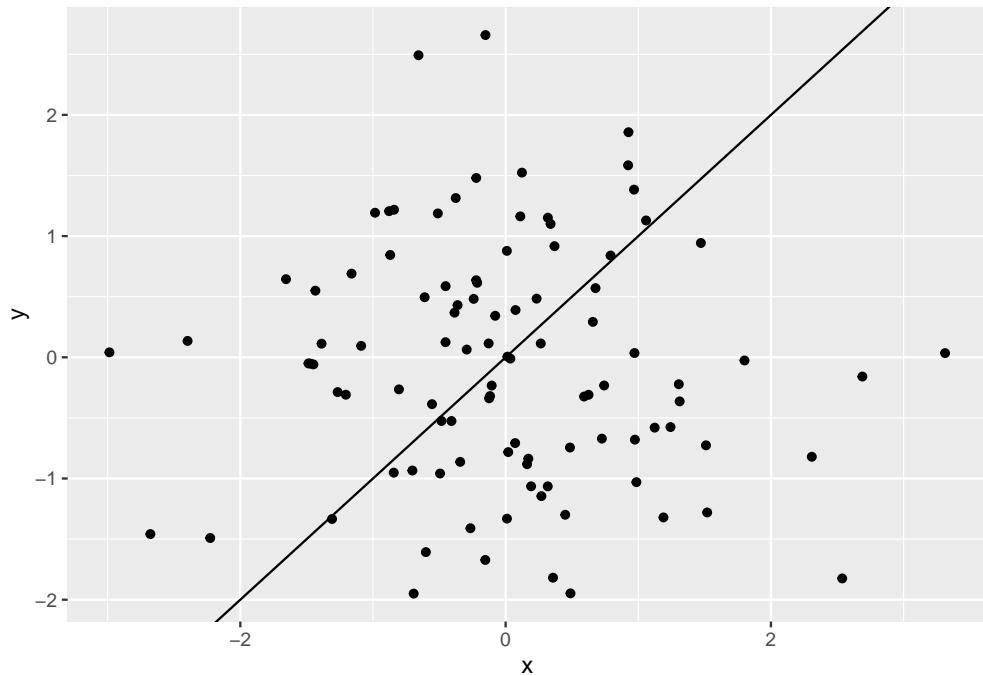
```

# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

```

```
ggplot(df, aes(x = x, y = y)) +  
  geom_point() +  
  geom_abline(intercept = 0,  
              slope = 1)
```



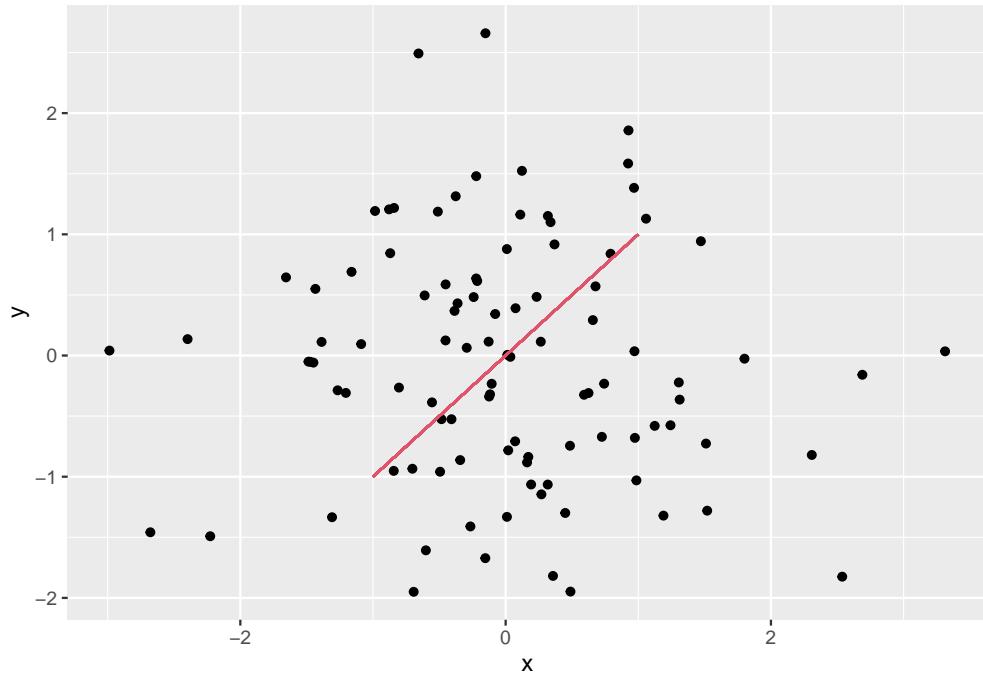
8.3 SEGMENTOS CON `geom_segment()`

Las funciones anteriores se pueden usar para agregar líneas a los diagramas, pero no se pueden crear segmentos con ellas. Para ello existe la función `geom_segment()`, que **permite especificar las coordenadas X e Y de inicio y fin del segmento deseado** con `x`, `y` (inicio) y `xend`, `yend` (final, respectivamente).

Hay que tener en cuenta que si se pasa la función `arrow` en el argumento `arrow` se puede **crear una flecha**. Se recomienda mirar los argumentos de la función `arrow` para obtener detalles adicionales de personalización.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
# Datos  
set.seed(80)  
df = data.frame(x = rnorm(100),  
                y = rnorm(100))
```

```
ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_segment(x = -1, y = -1,
               xend = 1, yend = 1,
               color = 2)
```

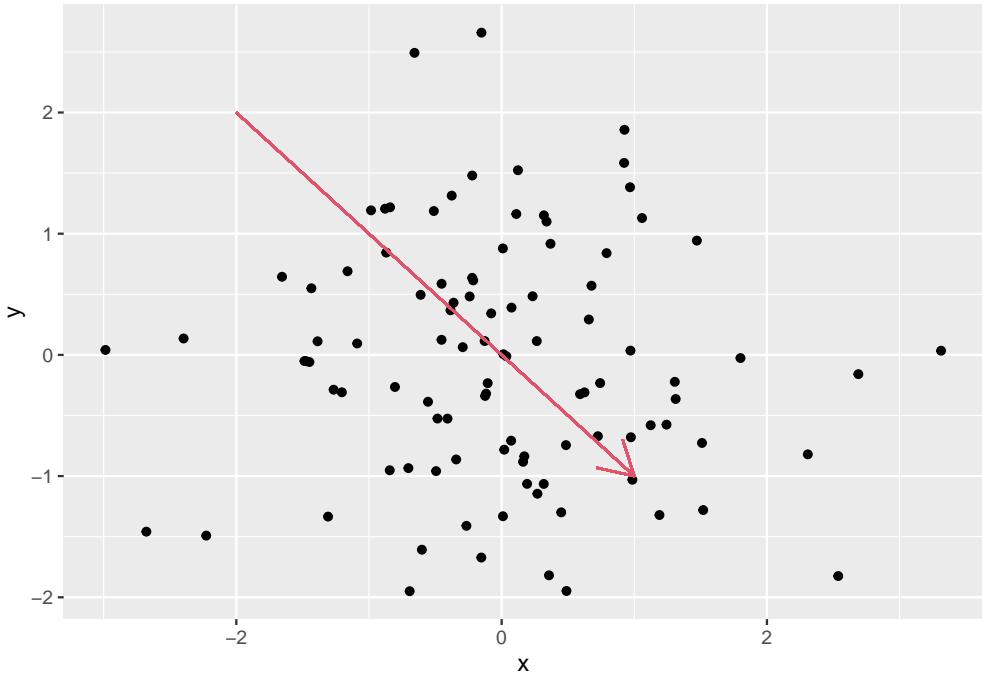


8.3.1 AGREGANDO UNA FLECHA

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_segment(x = -2, y = 2,
               xend = 1, yend = -1,
               color = 2,
               arrow = arrow())
```



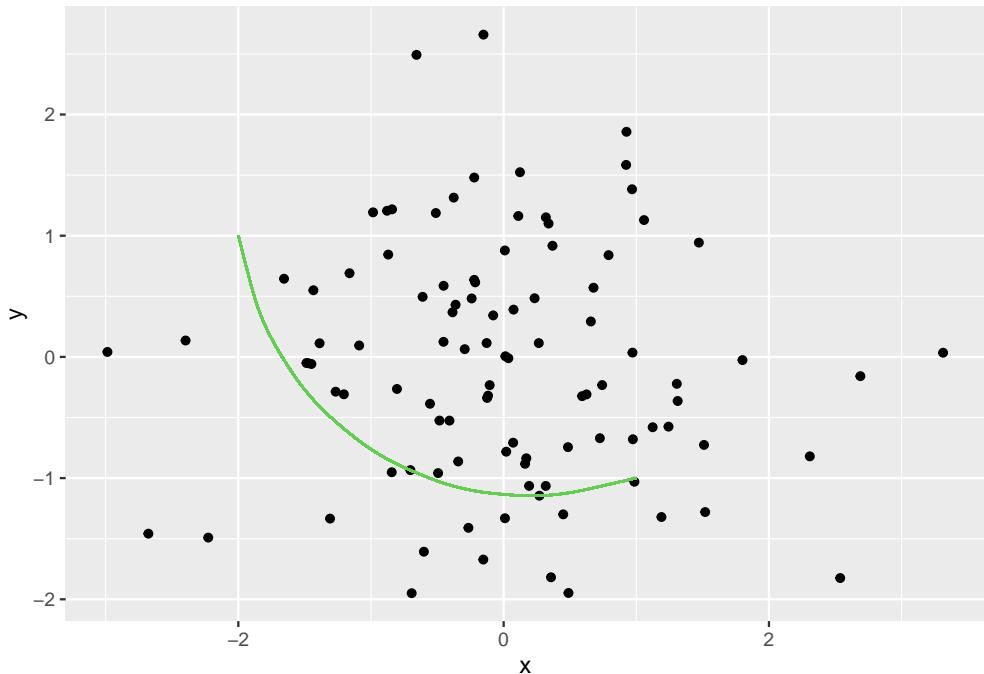
8.4 CURVAS CON `geom_curve()`

La función `geom_curve()` se comporta de la misma manera que `geom_segment()`. La única diferencia es que esta función creará una curva en lugar de una línea recta. También se puede crear una flecha con esta función.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 1,
             xend = 1, yend = -1,
             color = 3)
```

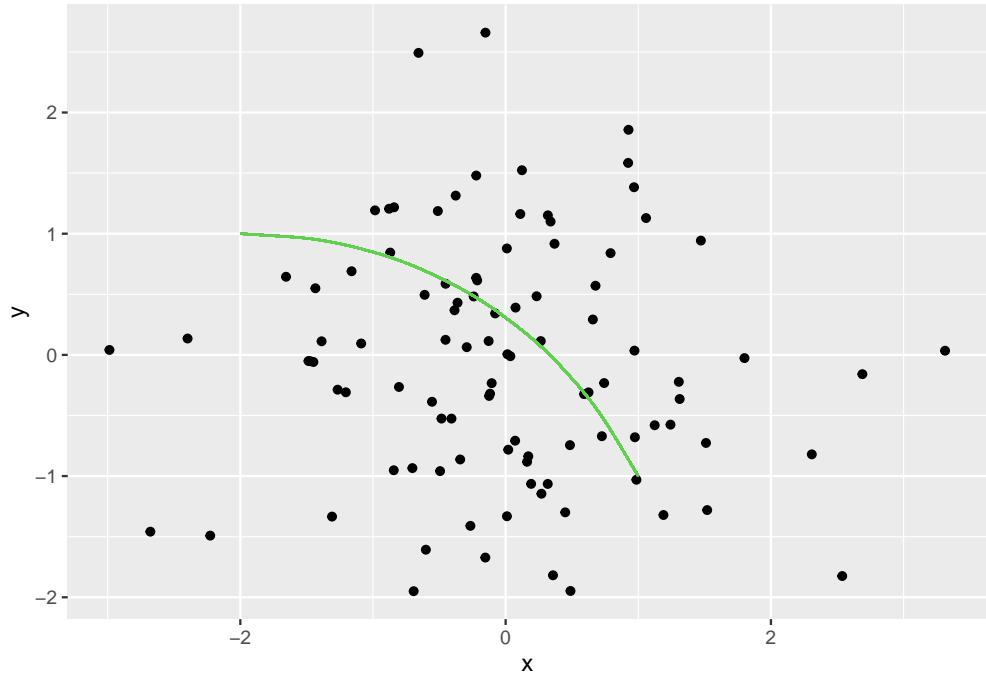


Hay que tener en cuenta que se puede establecer el nivel de curvatura, pasando un valor al argumento `curvature` de la función. Un cero será una línea recta, valores negativos producirán curvas hacia la izquierda y valores positivos producirán una curva hacia la derecha.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 1,
             xend = 1, yend = -1,
             color = 3,
             curvature = -0.3) # Grado de curvatura
```

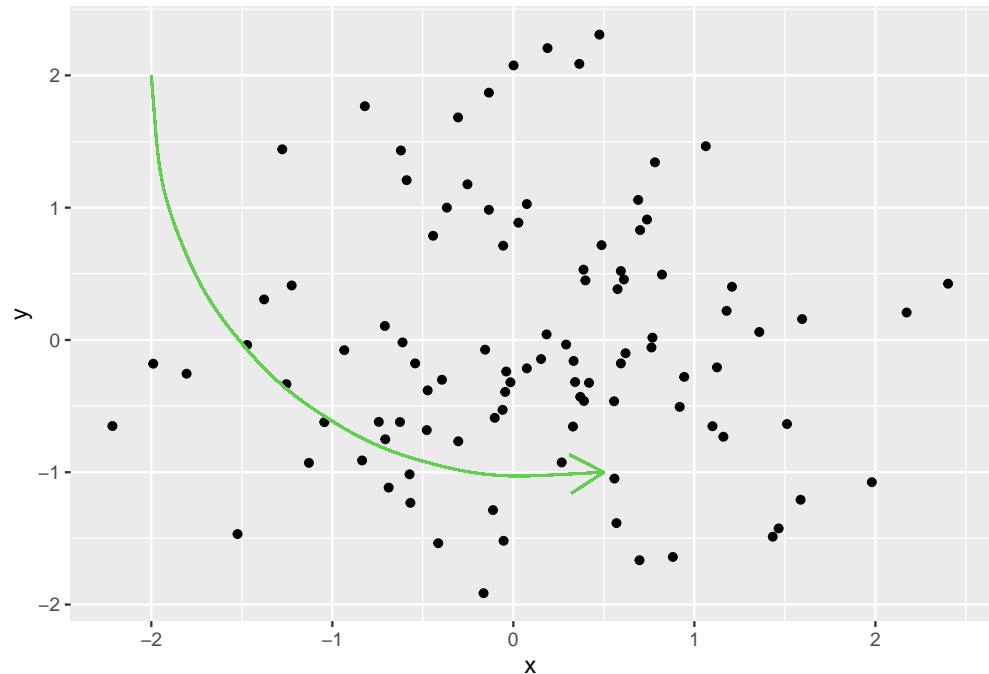


8.5 AGREGANDO FLECHAS

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1)
df = data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 2,
             xend = 0.5, yend = -1,
             color = 3,
             arrow = arrow())
```



PARTE 2: TODOS LOS GRÁFICOS DE GGPLOT2

En esta segunda parte de manual se contienen todos los capítulos de `ggplot2`. Cada capítulo contiene el código reproducible en diferentes personalizaciones del mismo diagrama o de uno equivalente. Para esto, se dividirá por categorías de tipos de diagramas las siguientes secciones de capítulos.

CAPÍTULO 9: DIAGRAMAS DE DISTRIBUCIÓN CON `ggplot2`



Los gráficos de distribución permiten, como su propio nombre lo indica, poder visualizar la forma en la que se distribuyen los datos sobre el soporte y comparar varios grupos.

9.1 BOX PLOT EN `ggplot2`

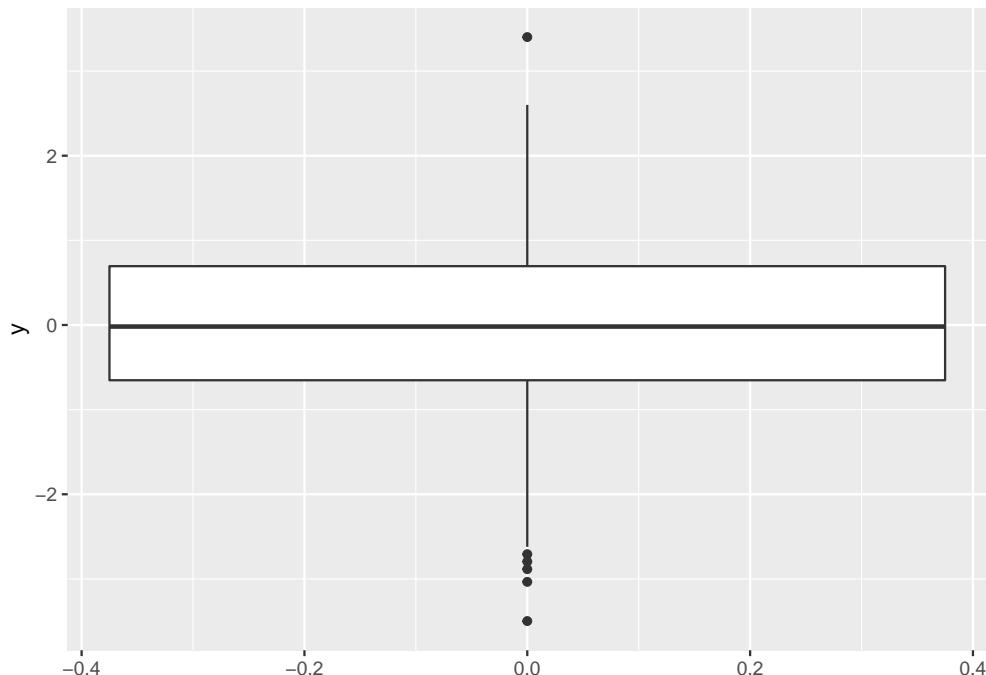
9.1.1 BOX PLOT EN `ggplot2` con la función “`geom_boxplot()`”

Si se tiene un **data frame** que contiene una variable numérica se puede usar la función `ggplot()` para crear un boxplot (diagrama de cajas) R usando el paquete `ggplot2`.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Diagrama básico
ggplot(df, aes(y = y)) +
  geom_boxplot()
```

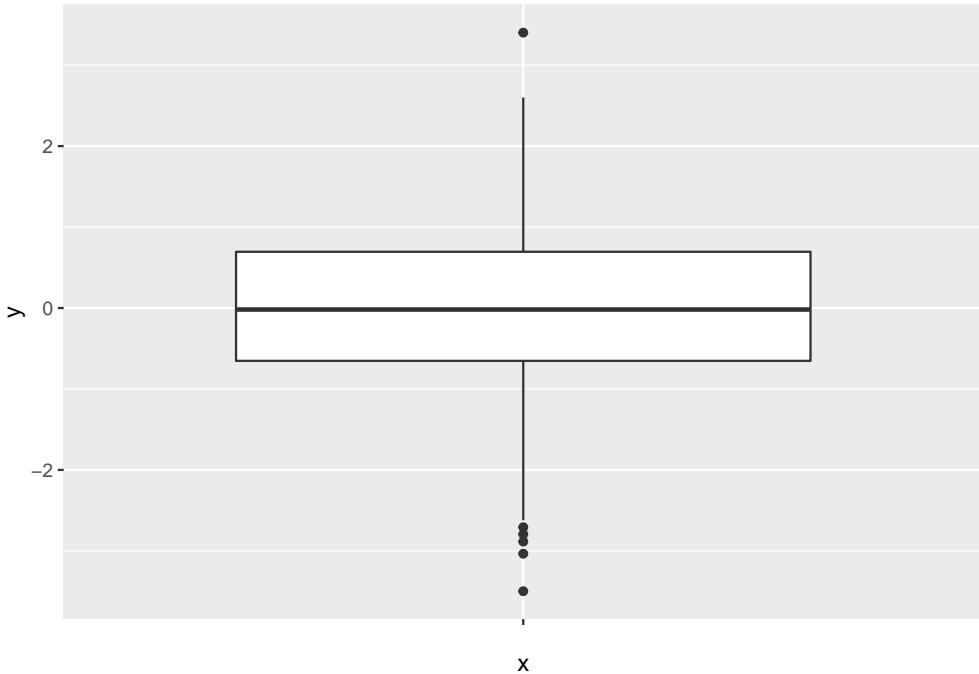


Alternativamente se puede establecer el argumento `x = ""`. Esto eliminará los valores del eje X y se hará la caja más estrecha.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Basic box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot()
```



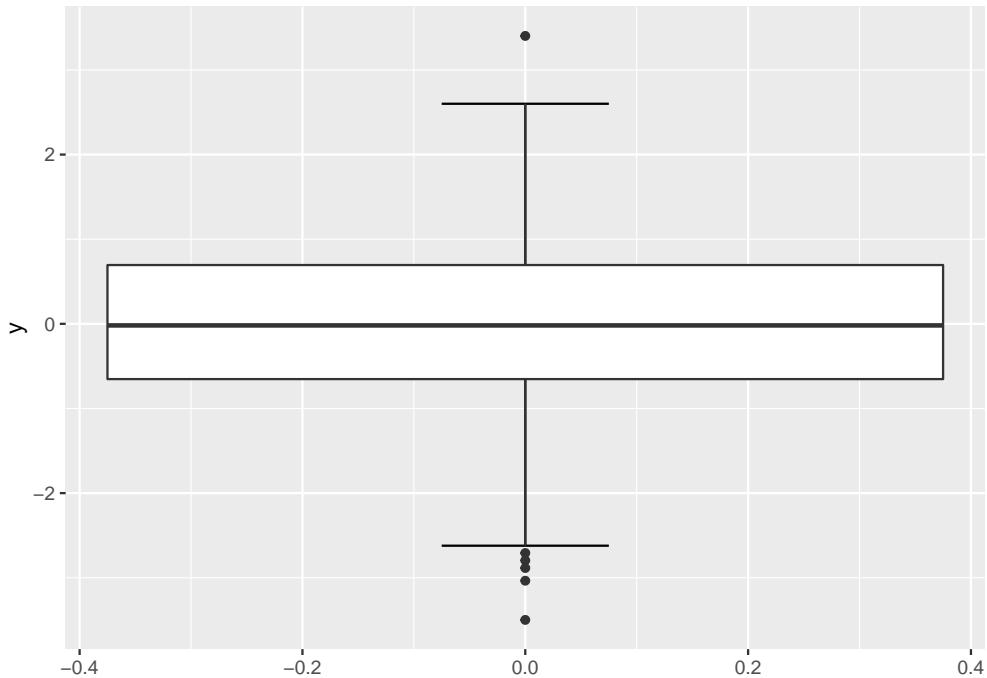
9.1.2 AGREGANDO BARRAS DE ERROR CON LA FUNCIÓN `stat_boxplot()`

El diagrama de caja y bigotes no añade las líneas horizontales de las barras de error, pero se pueden agregar con la función `stat_boxplot()`, estableciendo `geom = "errorbar"`. Hay que tener en cuenta que se puede cambiar su ancho con el argumento `width`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Basic box plot
ggplot(df, aes(y = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.15) +
  geom_boxplot()
```



9.1.3 DIAGRAMA DE CAJAS HORIZONTAL

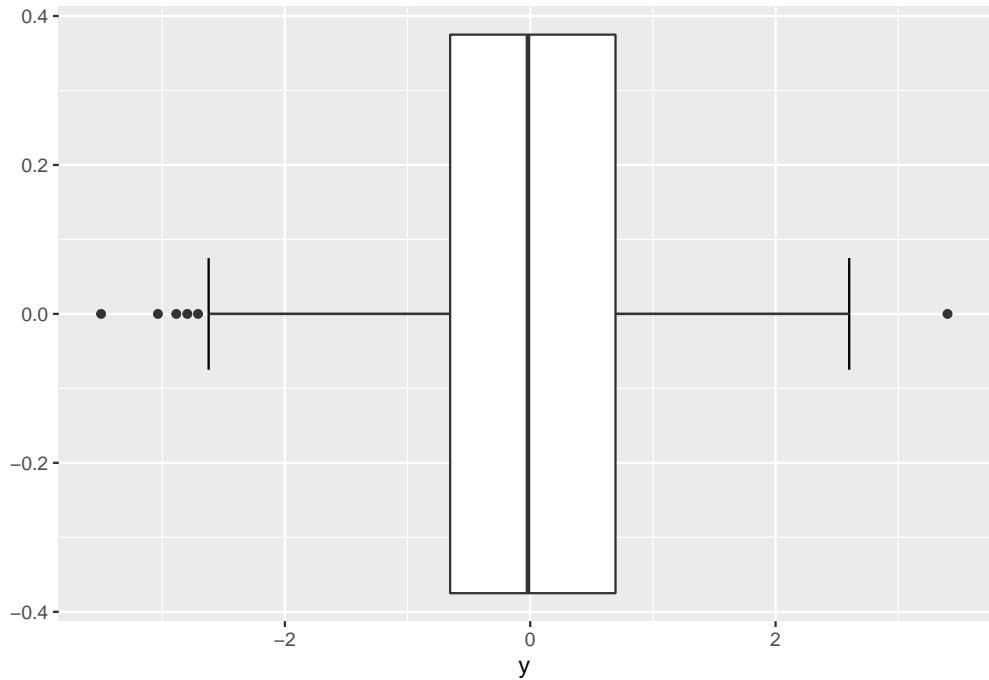
Existen dos maneras de cambiar la orientación de un box plot en ggplot2: cambiando la variable en `aes()` o usando la función `coord_flip()`, tal como se muestra en los siguientes ejemplos.

9.1.3.1 OPCIÓN 1: CAMBIANDO EL ARGUMENTO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Box plot horizontal
ggplot(df, aes(x = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.15) +
  geom_boxplot()
```

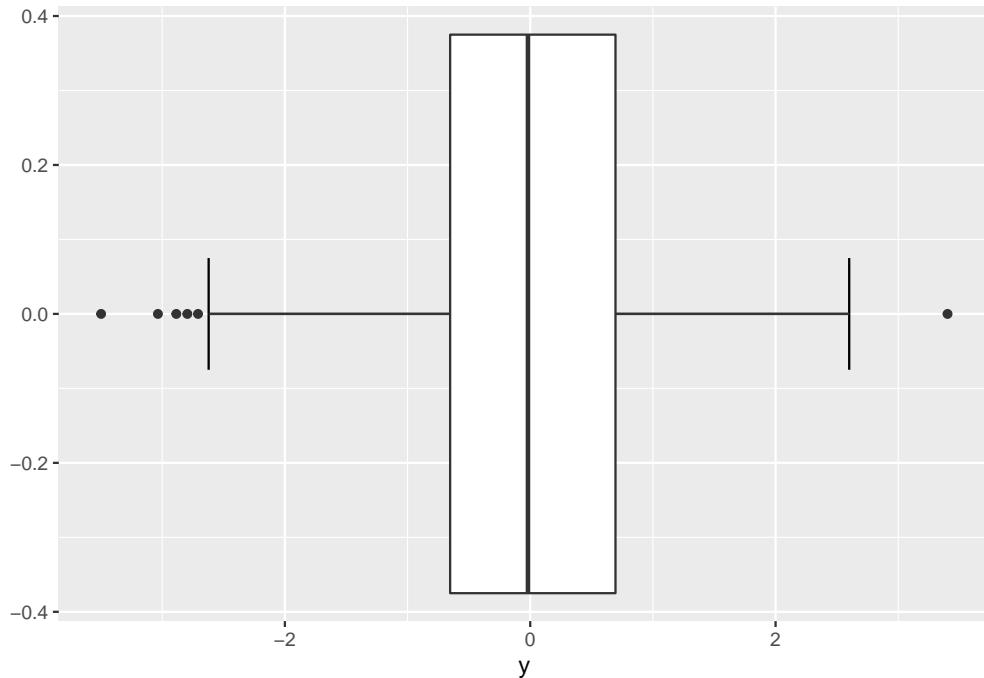


9.1.3.2 OPCIÓN 2: USANDO LA FUNCIÓN coord_flip()

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Box plot horizontal
ggplot(df, aes(y = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.15) +
  geom_boxplot() +
  coord_flip()
```



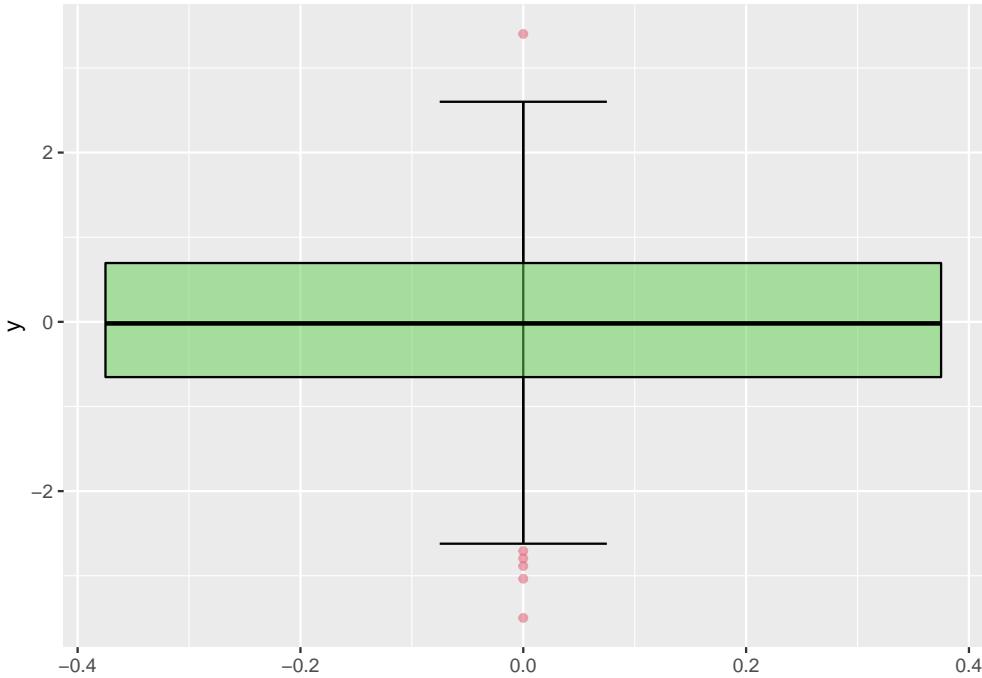
9.1.4 PERSONALIZACIÓN DEL BOXPLOT

Los box plots se pueden personalizar haciendo uso de los argumentos de las funciones `stat_boxplot()` y `geom_boxplot()`. En los ejemplos siguientes se cambian los colores y los tipos de líneas de los diagramas, resaltando los argumentos correspondientes.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Box plot customization
ggplot(df, aes(y = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.15,
               color = 1) + # Color barras error
  geom_boxplot(fill = 3,           # Color caja
               alpha = 0.5,        # Transparencia
               color = 1,          # Color del borde
               outlier.colour = 2) # Color atípicos
```

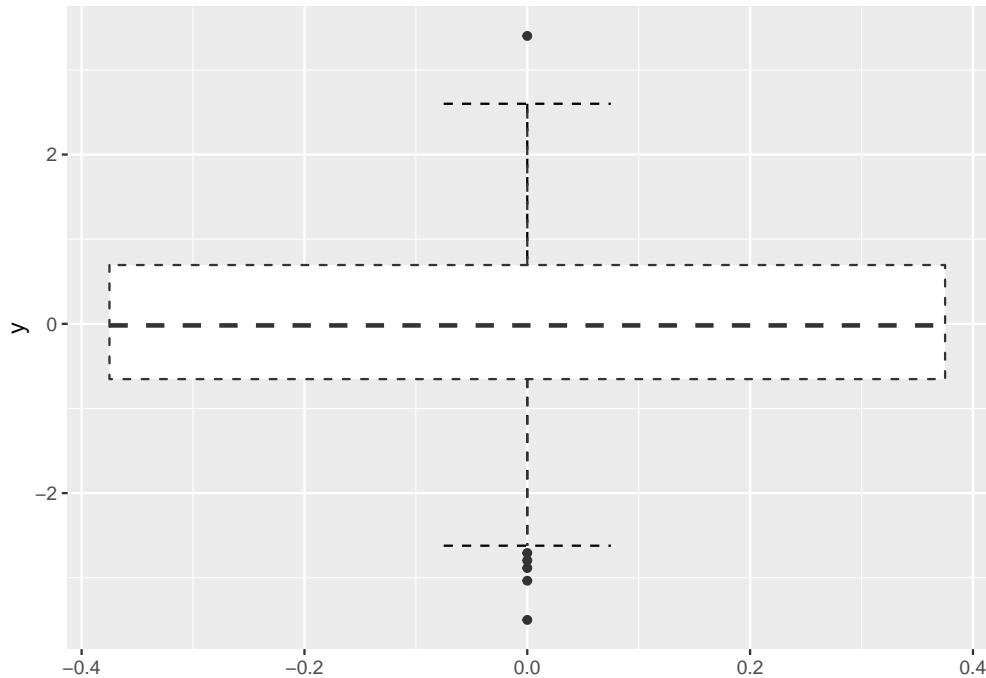


9.1.4.1 TIPOS DE LÍNEAS

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
y = rnorm(600)
df = data.frame(y)

# Basic box plot
ggplot(df, aes(y = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.15,
               linetype = 2, # Tipo de línea
               lwd = 0.5) +
  geom_boxplot(linetype = 2, # Tipo de línea
               lwd = 0.5) # Ancho de línea
```



9.2 BOX PLOT POR GRUPO EN `ggplot2`

En esta parte se utilizará el siguiente data frame, cuya primera columna es una variable numérica y la segunda variable categórica representando grupos.

```
# Base de ejemplo
set.seed(999)
df = data.frame(y = rnorm(300),
                 grupo = sample(LETTERS[1:3],
                                size = 300,
                                replace = TRUE))

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
kable_styling(
  latex_options = c("striped", "condensed", "HOLD_position"),
  position = "center",
  full_width = FALSE)
```

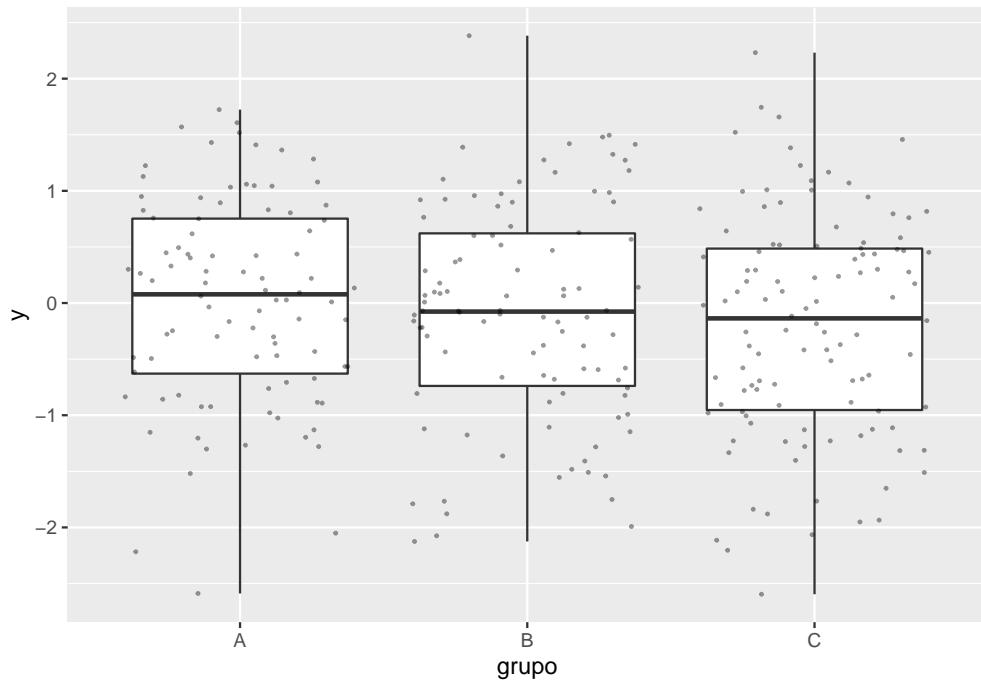
	y	grupo
-0.2817402	B	
-1.3125596	C	
0.7951840	C	
0.2700705	C	
-0.2773064	A	
-0.5660237	A	
-1.8786583	B	
-1.2667911	A	
-0.9677497	C	
-1.1210094	B	

9.2.1 BOX PLOT POR GRUPO CON `geom_boxplot()`

Para crear un box plot agrupado en R, se necesita pasar las variables a la función `aes()` y usar la función `geom_boxplot()` como el siguiente ejemplo:

```
# install.packages("ggplot2")
library(ggplot2)

# Box plot by grupo
ggplot(df, aes(x = grupo, y = y)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.4, size = 0.4)
```

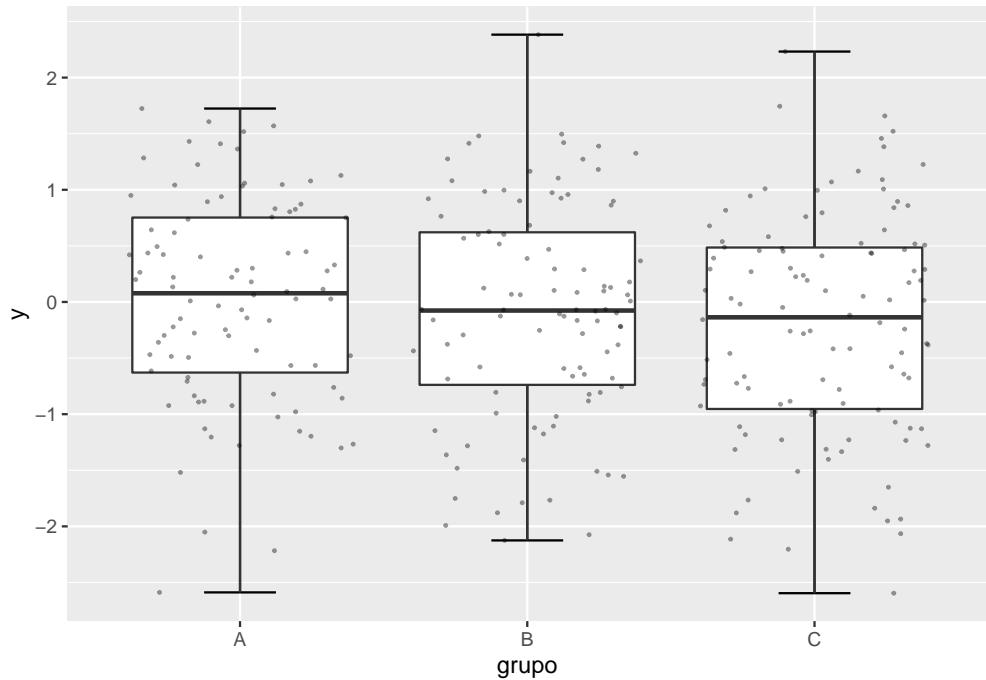


9.2.2 AGREGANDO BARRAS DE ERROR CON LA FUNCIÓN stat_boxplot()

Como ya se habrá notado, los diagramas de caja no añaden las líneas de barras de error por defecto. Si se quiere agregar se usa la función `stat_boxplot()` y establecer el argumento `geom = "errorbar"`. El ancho de las barras se puede personalizar con el argumento `width`.

```
# install.packages("ggplot2")
library(ggplot2)

# Box plot por grupo con barras de error
ggplot(df, aes(x = grupo, y = y)) +
  stat_boxplot(geom = "errorbar", # Error barras
               width = 0.25) +    # Bars tamaño
  geom_boxplot() +
  geom_jitter(alpha = 0.4, size = 0.4)
```



9.2.3 BOX PLOT HORIZONTAL POR GRUPO

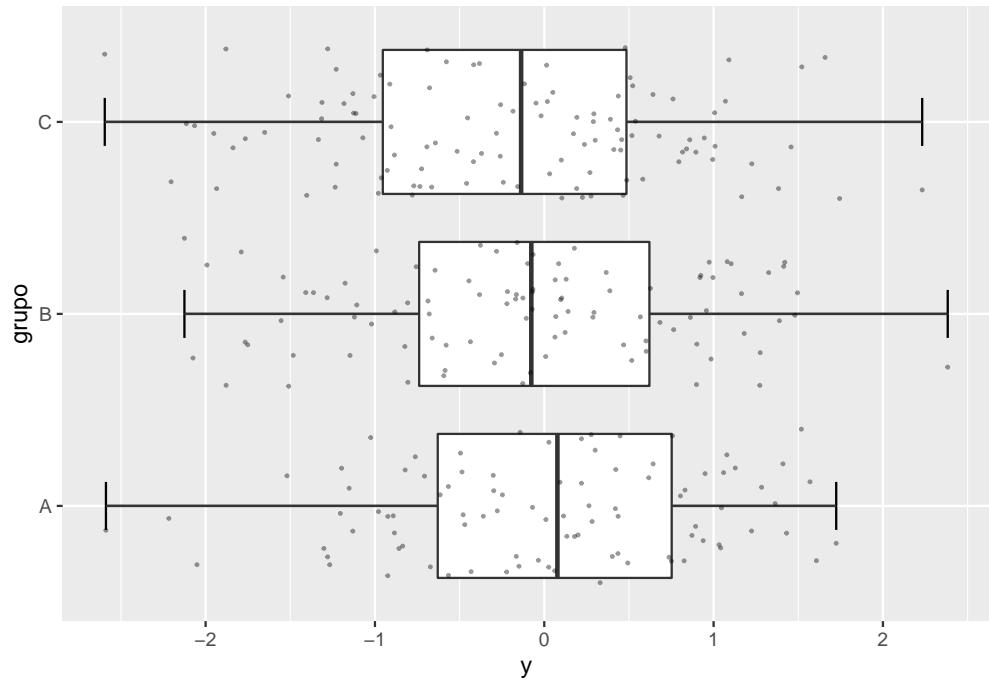
Los diagramas de cajas y bugotes también se pueden mostrar en modo horizontal. Para lograrlo se puede cambiar el orden de las variables dentro de la función `aes()` o usar la función `coord_flip()`, tal y como se muestra a continuación.

9.2.3.1 OPCIÓN 1: CAMBIAR EL ORDEN DE LAS VARIABLES

Establecer la variable categórica en el eje Y.

```
# install.packages("ggplot2")
library(ggplot2)

# Box plot horizontal en ggplot2
ggplot(df, aes(x = y, y = grupo)) +
  stat_boxplot(geom = "errorbar",
               width = 0.25) +
  geom_boxplot() +
  geom_jitter(alpha = 0.4, size = 0.4)
```

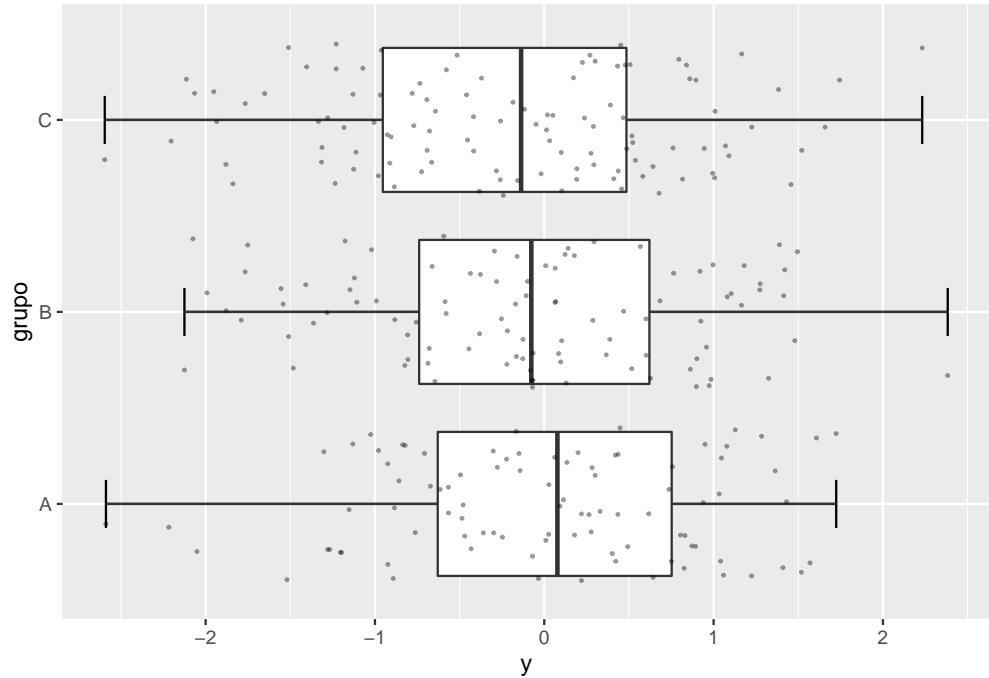


9.2.3.2 OPCIÓN 2: USAR LA FUNCIÓN `coord_flip()`

La función `coord_flip()` invertirá los ejes, por lo que un box plot vertical se convertirá en horizontal y viceversa.

```
# install.packages("ggplot2")
library(ggplot2)

# Box plot horizontal
ggplot(df, aes(x = grupo, y = y)) +
  stat_boxplot(geom = "errorbar",
               width = 0.25) +
  geom_boxplot() +
  coord_flip() +
  geom_jitter(alpha = 0.4, size = 0.4)
```

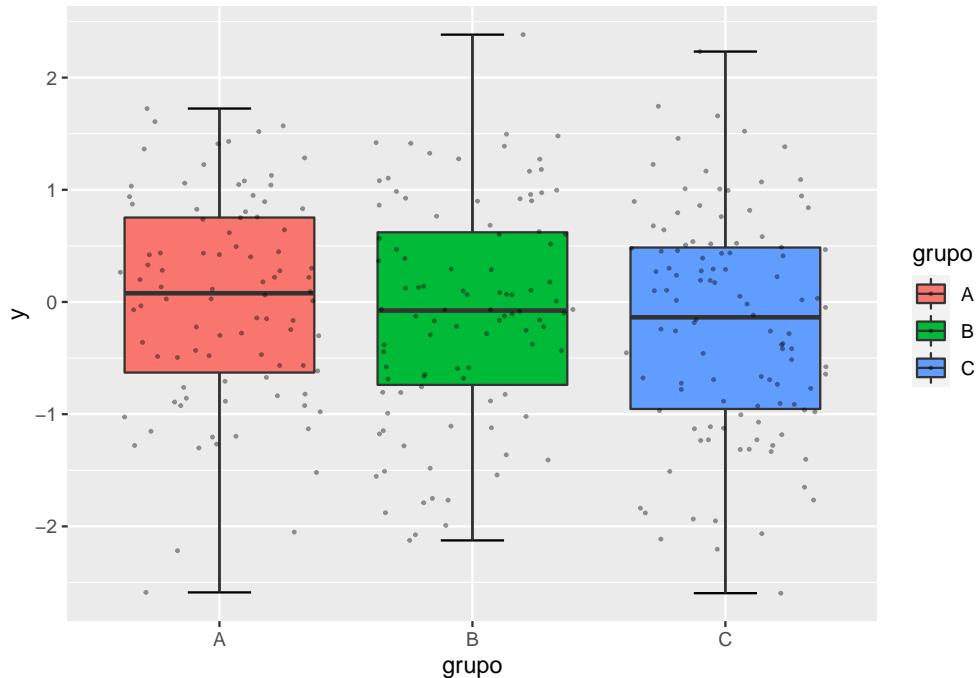


9.2.4 PERSONALIZACIÓN DE LOS COLORES

Si se pasa la variable categórica al argumento `fill` de la función `aes()`, cada caja se coloreará de un color y se mostrará una leyenda.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = grupo, y = y, fill = grupo)) +
  stat_boxplot(geom = "errorbar",
               width = 0.25) +
  geom_boxplot() +
  geom_jitter(alpha = 0.4, size = 0.4)
```

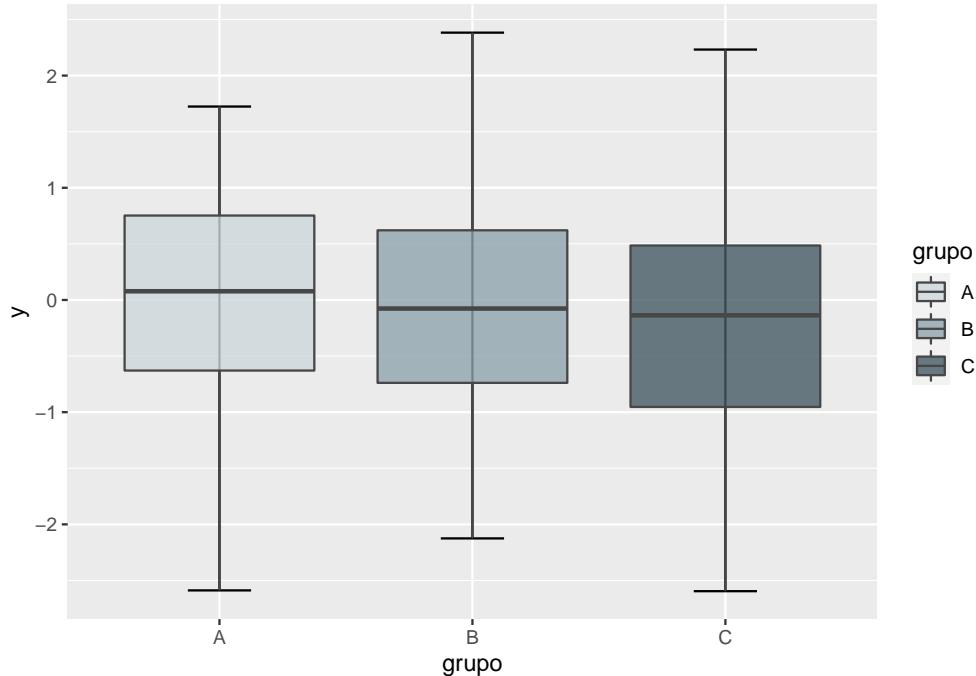


Los colores de los diagramas de cajas y bigotes son completamente personalizables. En los siguientes ejemplos se establece un color para cada grupo, cambiando el color del borde de las cajas y estableciendo el color de los atípicos como negro.

```
# install.packages("ggplot2")
library(ggplot2)

# Colores de las cajas
cols = c("#CFD8DC", "#90A4AE", "#455A64")

ggplot(df, aes(x = grupo, y = y, fill = grupo)) +
  stat_boxplot(geom = "errorbar",
               width = 0.25) +
  geom_boxplot(alpha = 0.8,           # Transparencia
               colour = "#474747",    # Color del borde
               outlier.colour = 1) + # Color atípicos
  scale_fill_manual(values = cols)  # Colores de las cajas +
```



```
geom_jitter(alpha = 0.4, size = 0.4)
```

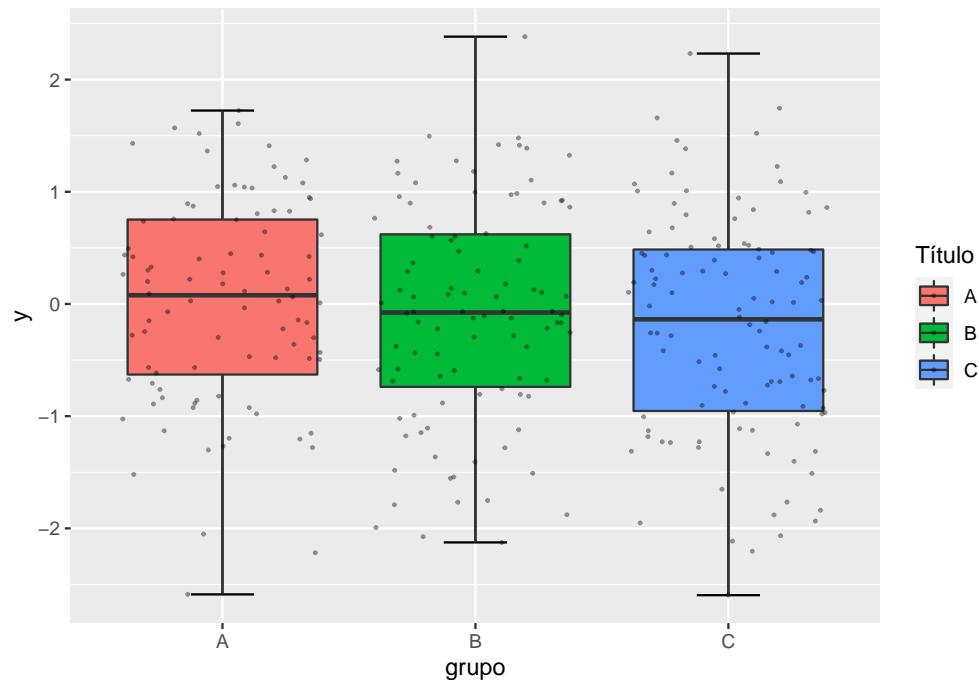
```
## geom_point: na.rm = FALSE  
## stat_identity: na.rm = FALSE  
## position_jitter
```

9.2.5 PERSONALIZACIÓN DE LA LEYENDA

9.2.5.1 CAMBIAR EL TÍTULO

Se puede cambiar el título por defecto de la leyenda con la función `guides()`.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
ggplot(df, aes(x = grupo, y = y, fill = grupo)) +  
  stat_boxplot(geom = "errorbar", width = 0.25) +  
  geom_boxplot() +  
  guides(fill = guide_legend(title = "Título")) +  
  geom_jitter(alpha = 0.4, size = 0.4)
```

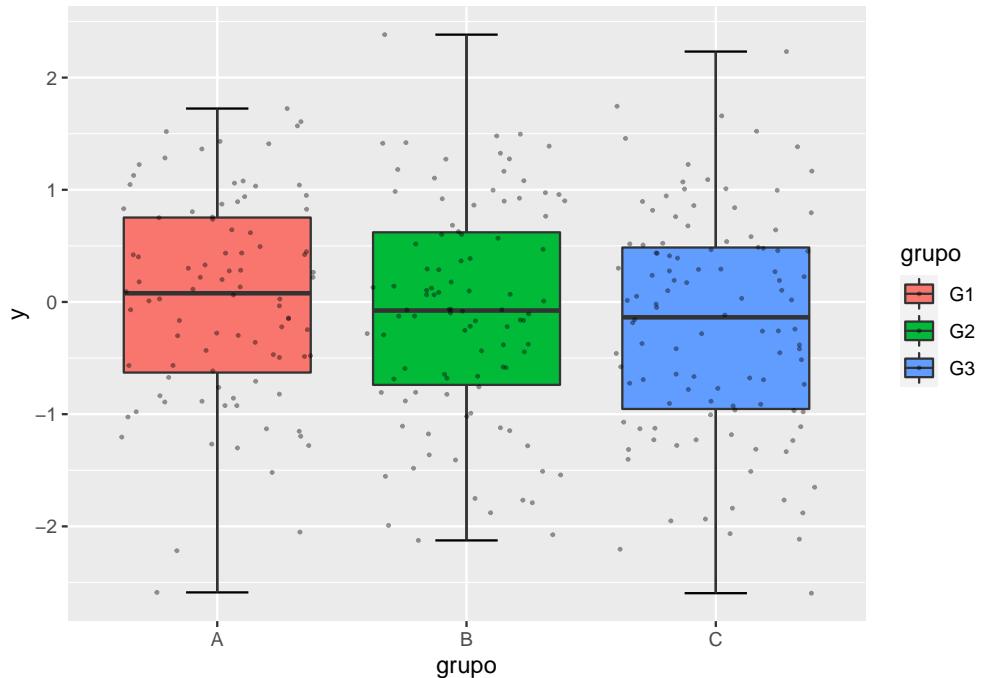


9.2.5.2 CAMBIAR LAS ETIQUETAS

Los textos de la leyenda son los niveles de la variable categórica. Se puede sobreescibir con la función `scale_fill_hue()`

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = grupo, y = y, fill = grupo)) +
  stat_boxplot(geom = "errorbar", width = 0.25) +
  geom_boxplot() +
  scale_fill_hue(labels = c("G1", "G2", "G3")) +
  geom_jitter(alpha = 0.4, size = 0.4)
```

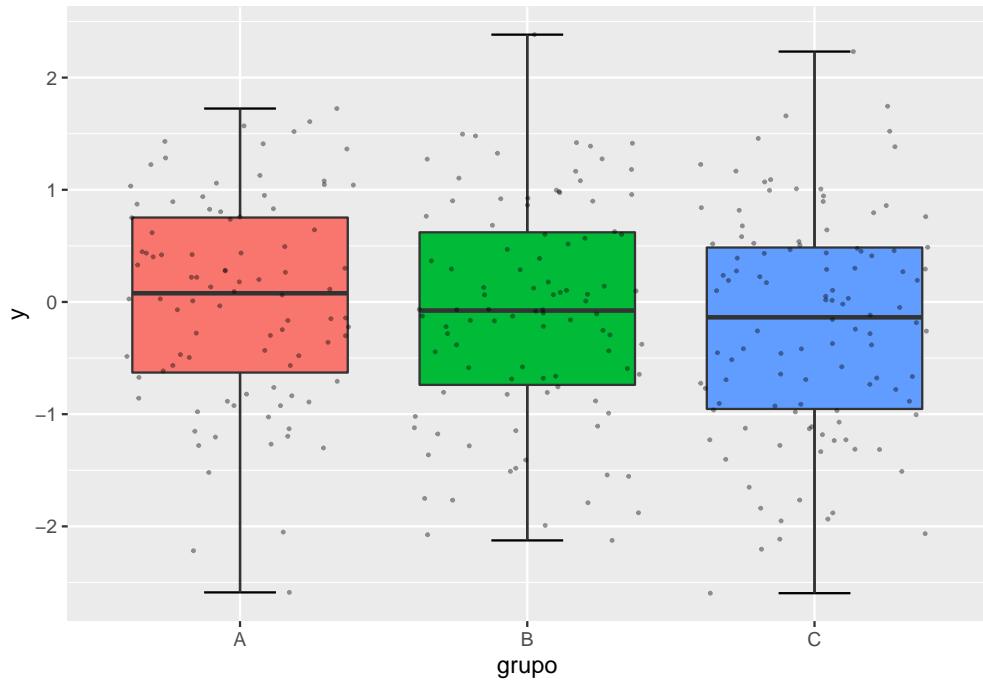


9.2.5.3 ELIMINAR LA LEYENDA

Si no se desea que se muestre la leyenda, se puede eliminar con la función `theme()`, pasando como argumento dentro `legend.position = "none"`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = grupo, y = y, fill = grupo)) +
  stat_boxplot(geom = "errorbar", width = 0.25) +
  geom_boxplot() +
  theme(legend.position = "none") +
  geom_jitter(alpha = 0.4, size = 0.4)
```



9.3 BOX PLOT CON PUNTOS DE DATOS EN `ggplot2`

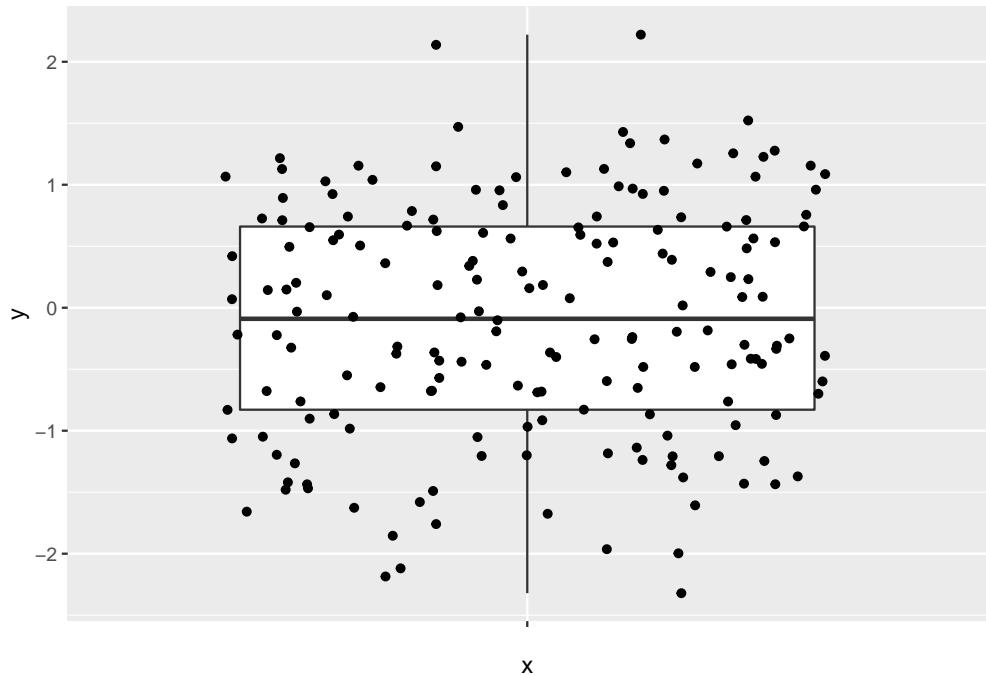
9.3.1 BOX PLOT CON OBSERVACIONES CON `geom_jitter()`

Agregar observaciones con ruido aleatorio (un stripchart) a un diagrama de cajas en `ggplot2` es muy fácil y útil para mostrar la distribución subyacente de los datos. Para ello se tendrá que usar la función `geom_jitter()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(180)
df = data.frame(y)

# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot() +
  geom_jitter()
```

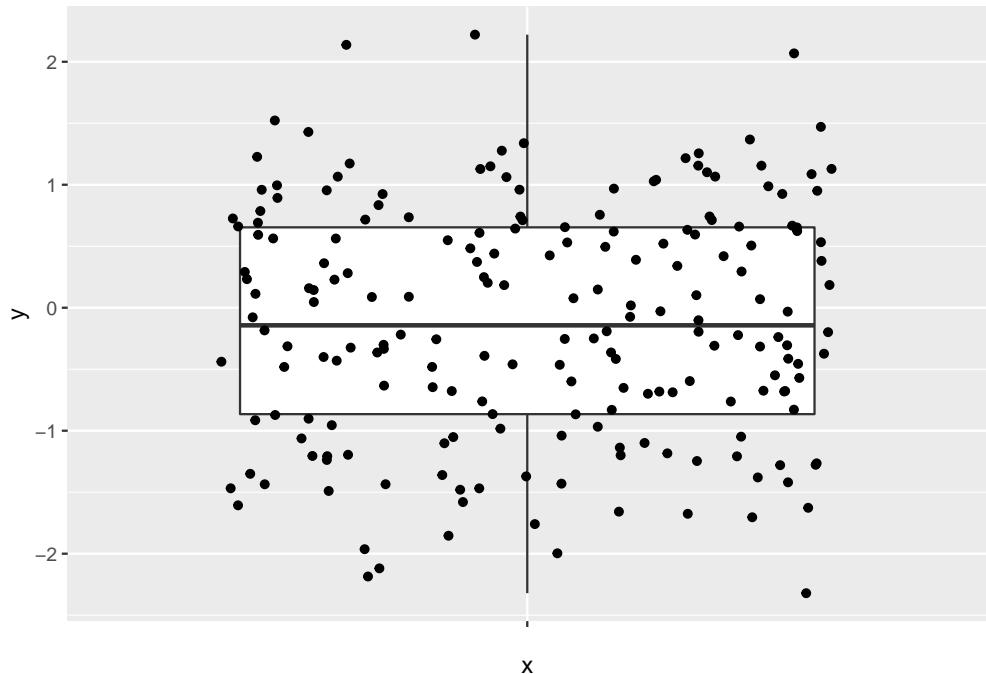


Una buena práctica es eliminar los atípicos con `outlier.shape = NA`, ya que con `geom_jitter()` también los mostrará.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(200)
df = data.frame(y)

# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter()
```

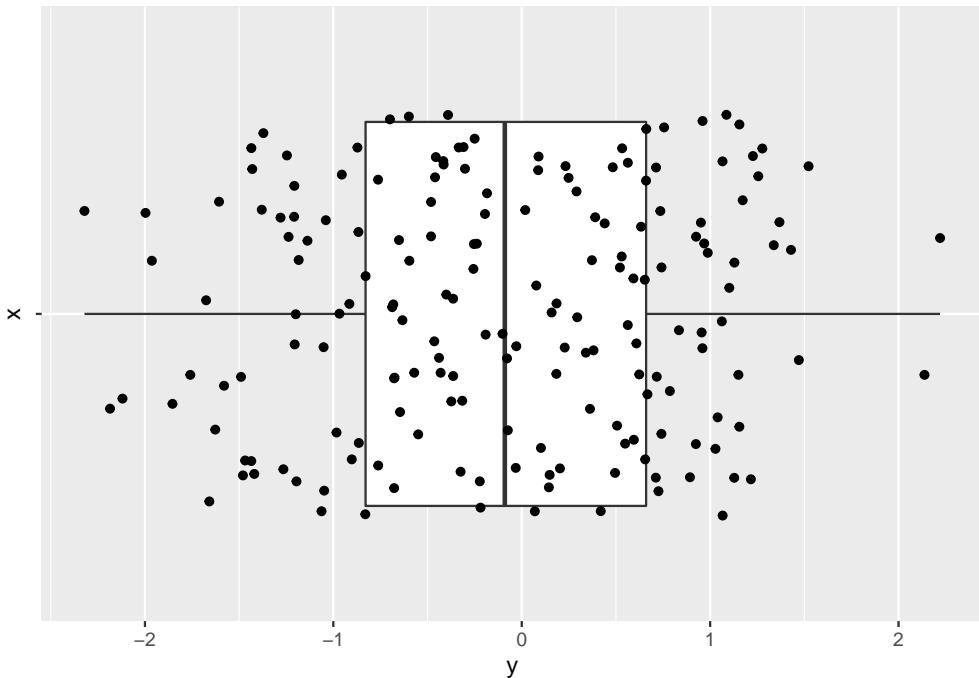


Hay que recordar que se puede rotar los ejes con la función `coord_flip()` o intercambiando las variables dentro de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(180)
df = data.frame(y)

# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter() +
  coord_flip()
```



9.3.2 PERSONALIZACIÓN DE LOS PUNTOS

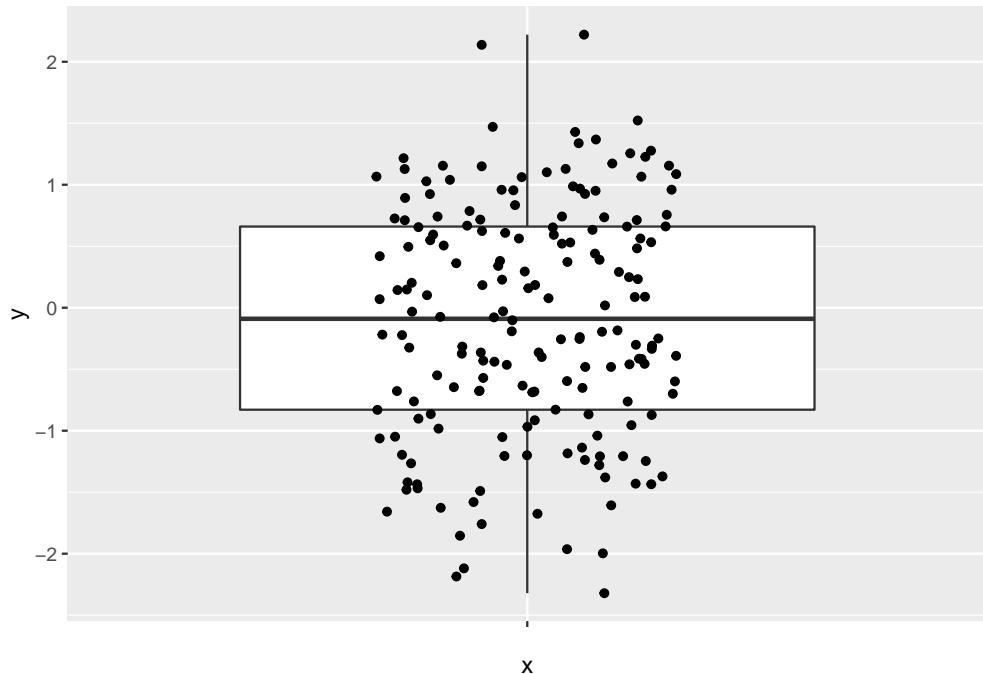
Las observaciones con ruido aleatorio se pueden personalizar de varias maneras. Se pueden cambiar la cantidad de ruido con `width`, el color de los puntos, su forma o su tamaño, como se muestra en los siguientes ejemplos.

9.3.2.1 ANCHO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(180)
df = data.frame(y)

# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0.2)
```

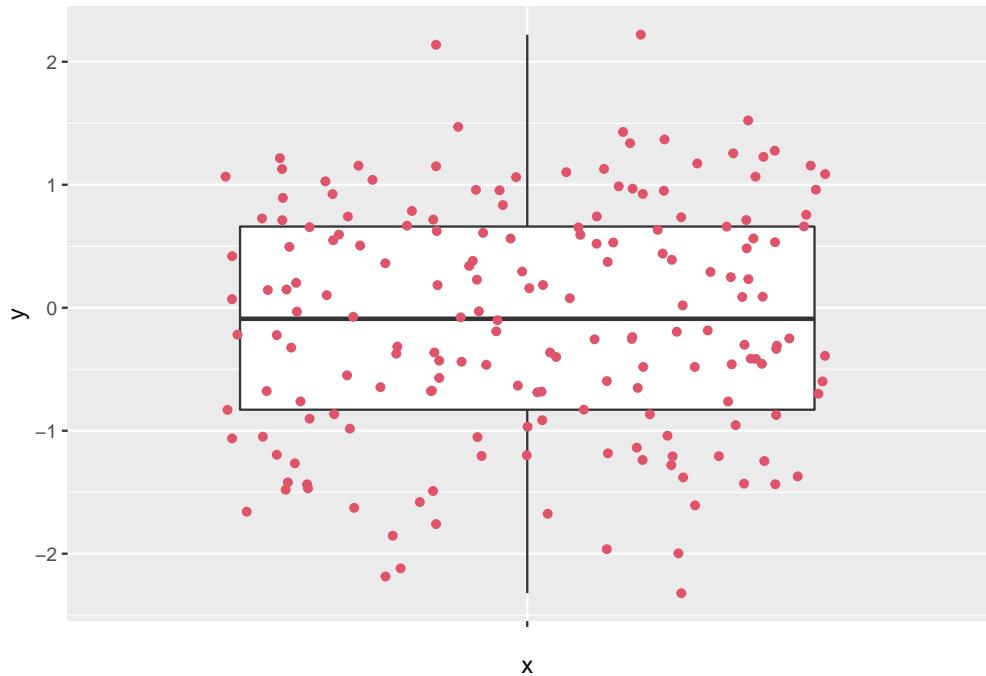


9.3.2.2

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(180)
df = data.frame(y)

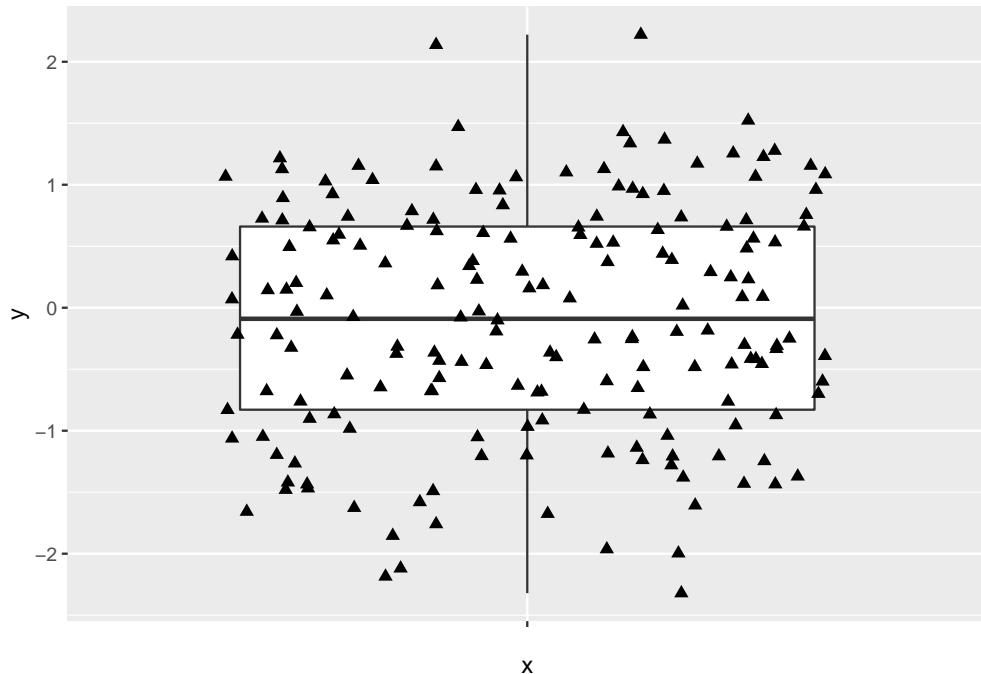
# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(colour = 2)
```



```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(10)
y = rnorm(180)
df = data.frame(y)

# Box plot
ggplot(df, aes(x = "", y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(shape = 17, size = 2)
```



9.3.3 BOX PLOT POR GRUPO CON PUNTOS DE DATOS

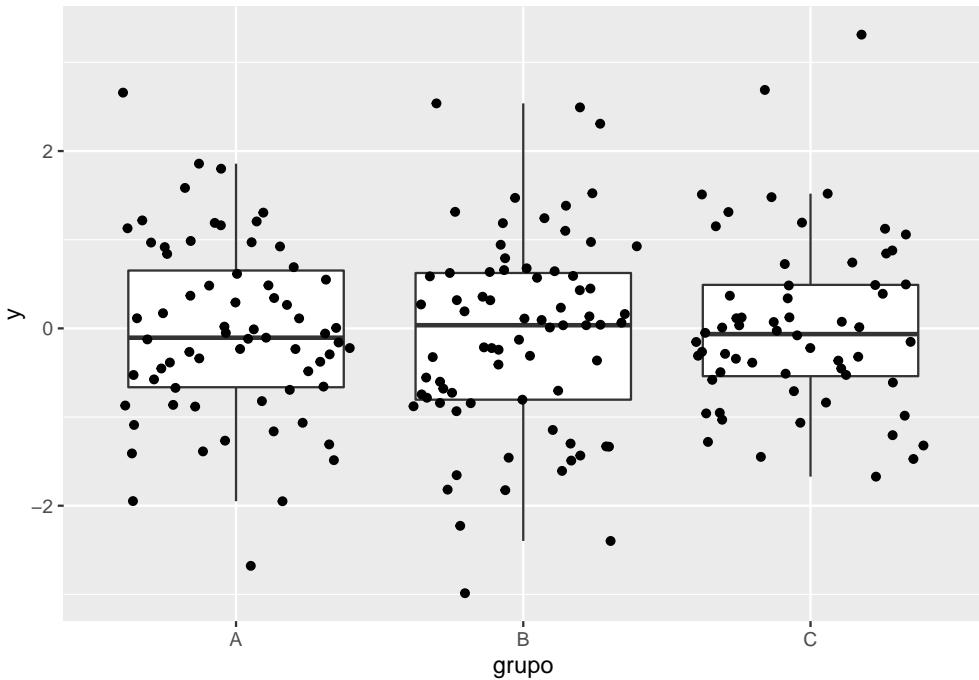
Si se tiene una variable categórica que represente grupos, se puede crear un diagrama de cajas por grupo y añadir las observaciones a cada grupo y personalizar su color, tamaño y forma.

9.3.3.1 OBSERVACIONES POR GRUPO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
y = rnorm(200)
grupo = sample(LETTERS[1:3], size = 200,
               replace = TRUE)
df = data.frame(y, grupo)

# Box plot por grupo con observaciones
ggplot(df, aes(x = grupo, y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter()
```

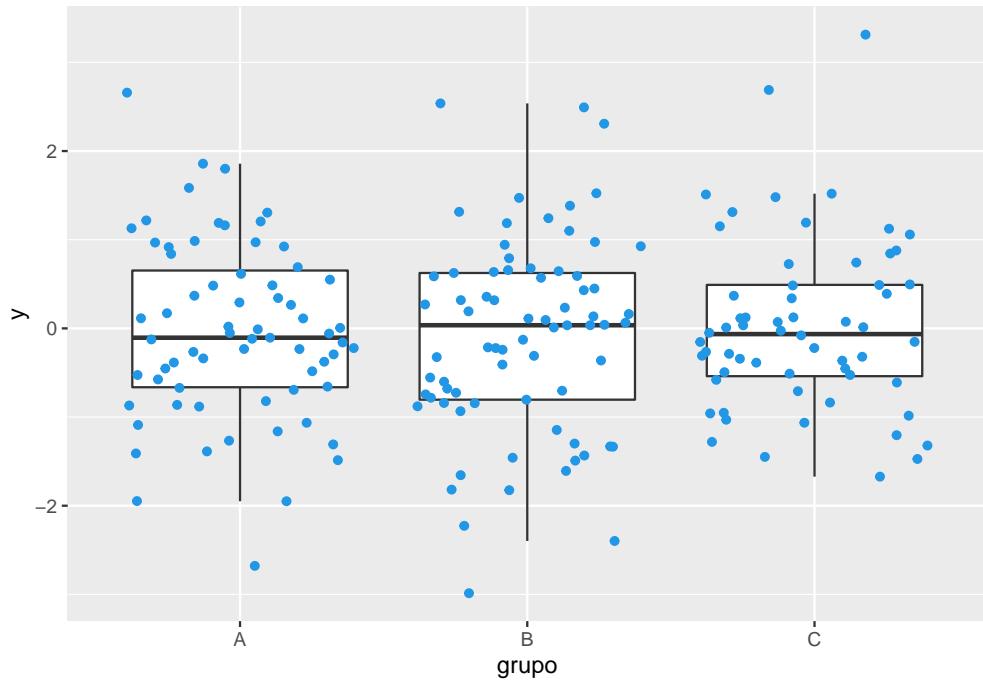


9.3.3.2 COLOR DE LOS PUNTOS

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
y = rnorm(200)
grupo = sample(LETTERS[1:3], size = 200,
              replace = TRUE)
df = data.frame(y, grupo)

# Box plot por grupo con observaciones
ggplot(df, aes(x = grupo, y = y)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(colour = 4)
```

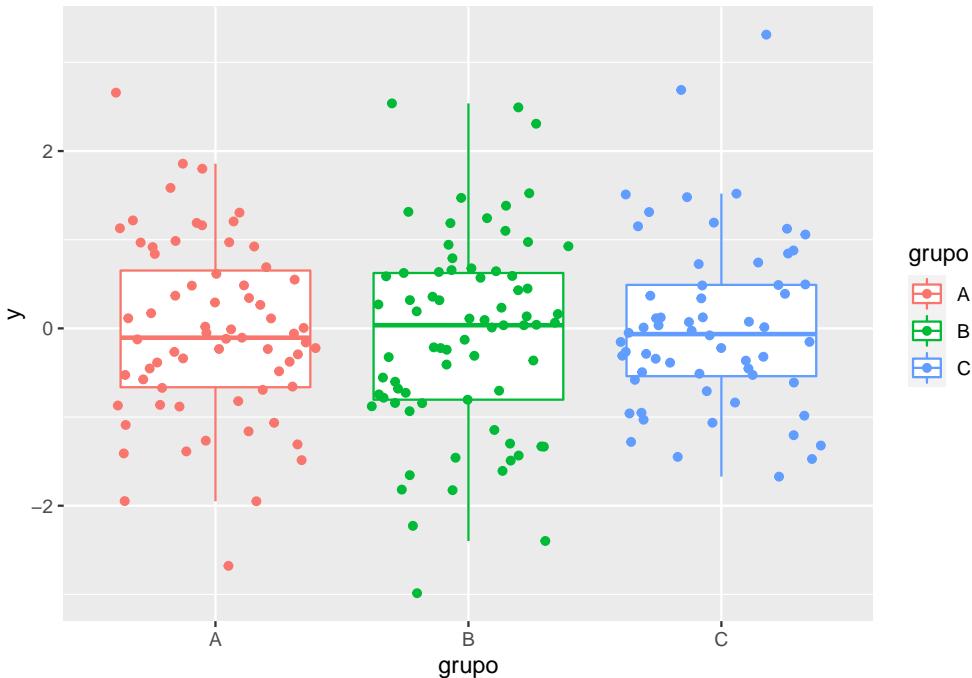


9.3.3.3 COLOR DE LOS PUNTOS POR GRUPO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
y = rnorm(200)
grupo = sample(LETTERS[1:3], size = 200,
              replace = TRUE)
df = data.frame(y, grupo)

# Box plot por grupo con observaciones
ggplot(df, aes(x = grupo, y = y, colour = grupo)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter()
```

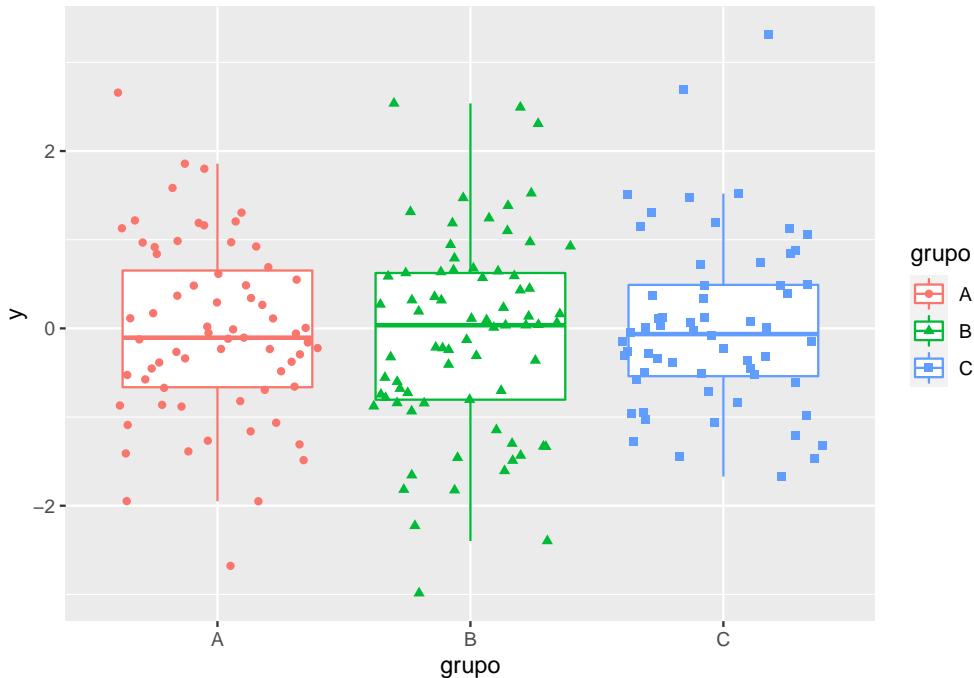


9.3.3.4 FORMA POR GRUPO

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(80)
y = rnorm(200)
grupo = sample(LETTERS[1:3],
              size = 200,
              replace = TRUE)
df = data.frame(y, grupo)

# Box plot por grupo con observaciones
ggplot(df, aes(x = grupo, y = y,
                colour = grupo,
                shape = grupo)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter()
```



9.4 HISTOGRAMA EN `ggplot2` CON EL MÉTODO DE STURGES

9.4.1 HISTOGRAMA POR DEFECTO EN R BASE Y EN `ggplot2`

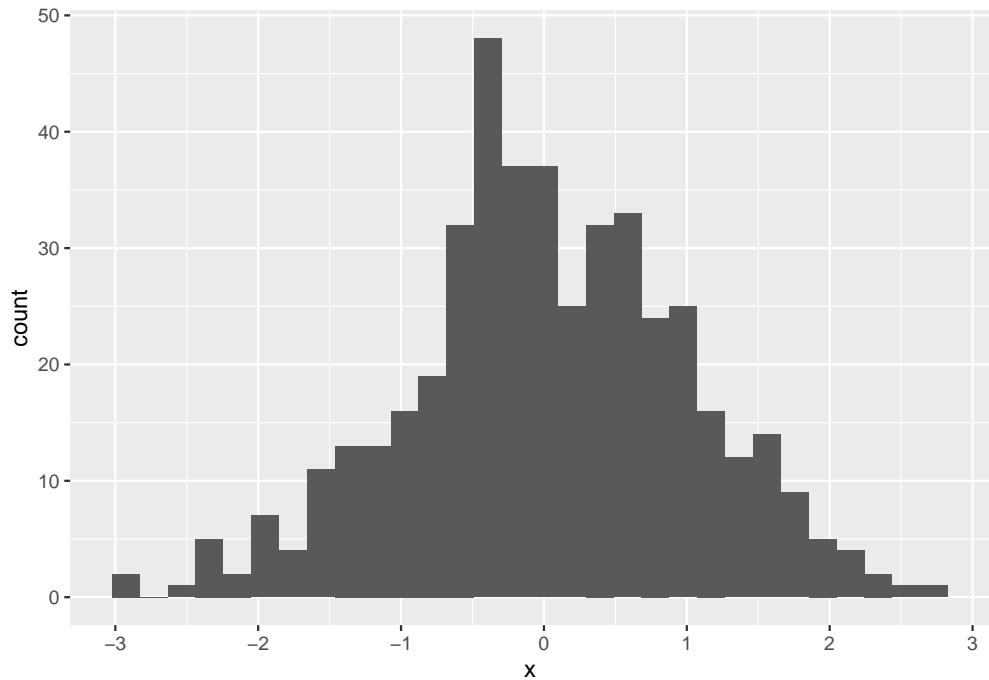
Los histogramas por defecto en `ggplot2` y R base son diferentes, ya que en `ggplot2` utiliza 30 clases, mientras que la función `hist()` de R base usa el método de Sturges para calcular el número de clases.

Como se puede ver, los histogramas de `ggplot2` tienden a tener demasiadas clases por defecto. Se puede cambiar el ancho de las barras o el número de clases al valor que se deseé.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1)
x = rnorm(450)
df = data.frame(x)

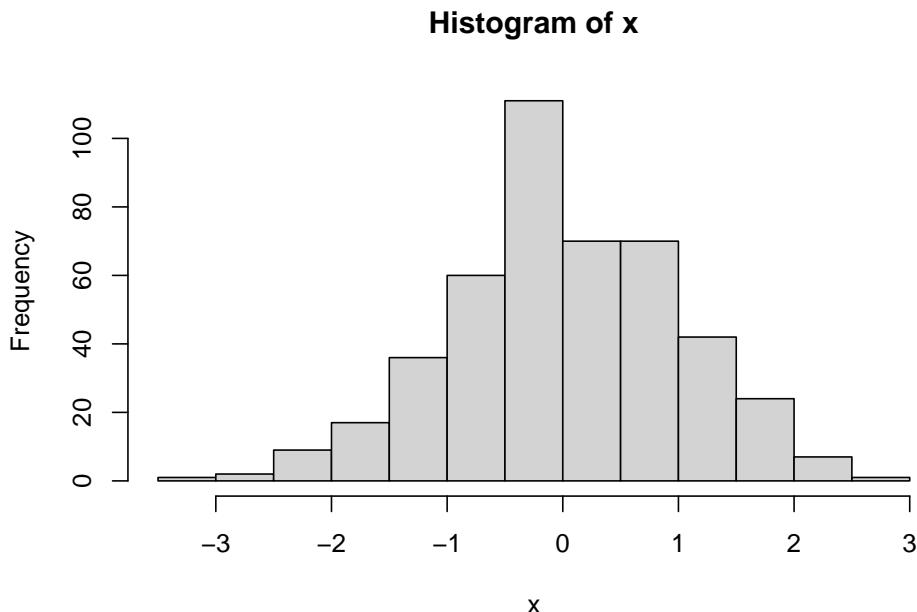
# Histograma por defecto en ggplot2
ggplot(df, aes(x = x)) +
  geom_histogram()
```



Los histogramas por defecto con la función `hist()` son más adecuados, ya que utiliza el método de Sturges.

```
# Datos
set.seed(1)
x = rnorm(450)
df = data.frame(x)

# Histograma por defecto en R base
hist(x)
```



9.4.2 Método de Sturges

Si se desea crear un histograma en `ggplot2` que use el método de Sturges, se puede calcular los puntos de corte de la siguiente manera, pasando el argumento `breaks`.

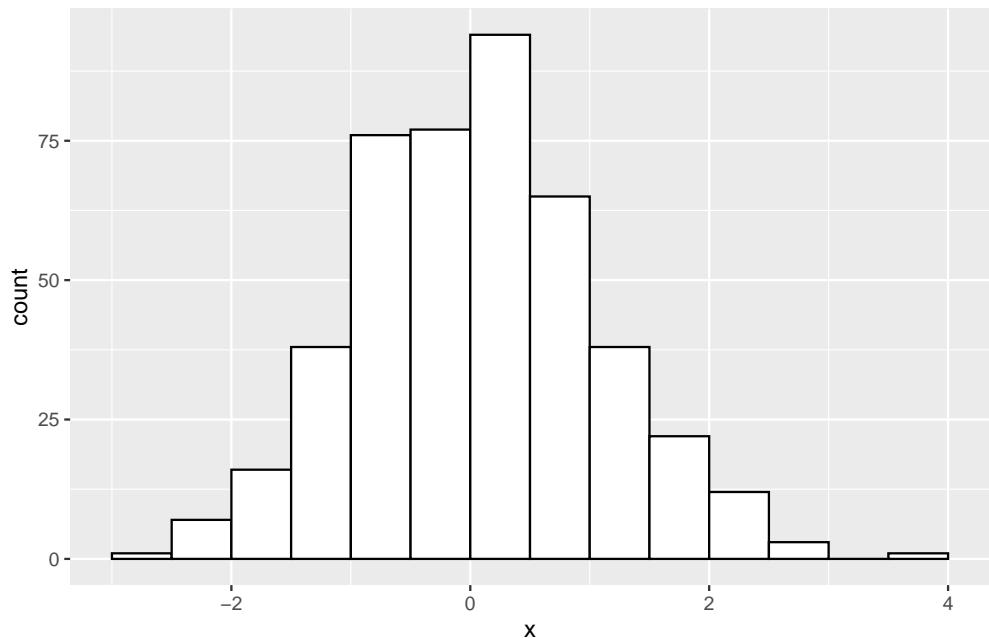
```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(3)
x = rnorm(450)
df = data.frame(x)

# Calcular los puntos de corte
breaks = pretty(range(x),
               n = nclass.Sturges(x),
               min.n = 1)
df$breaks = breaks

# Histograma con el método de Sturges
ggplot(df, aes(x = x)) +
  geom_histogram(color = 1, fill = "white",
                 breaks = breaks) +
  ggtitle("Método de Sturges")
```

Método de Sturges



9.5 HISTOGRAMA CON DENSIDAD EN `ggplot2`

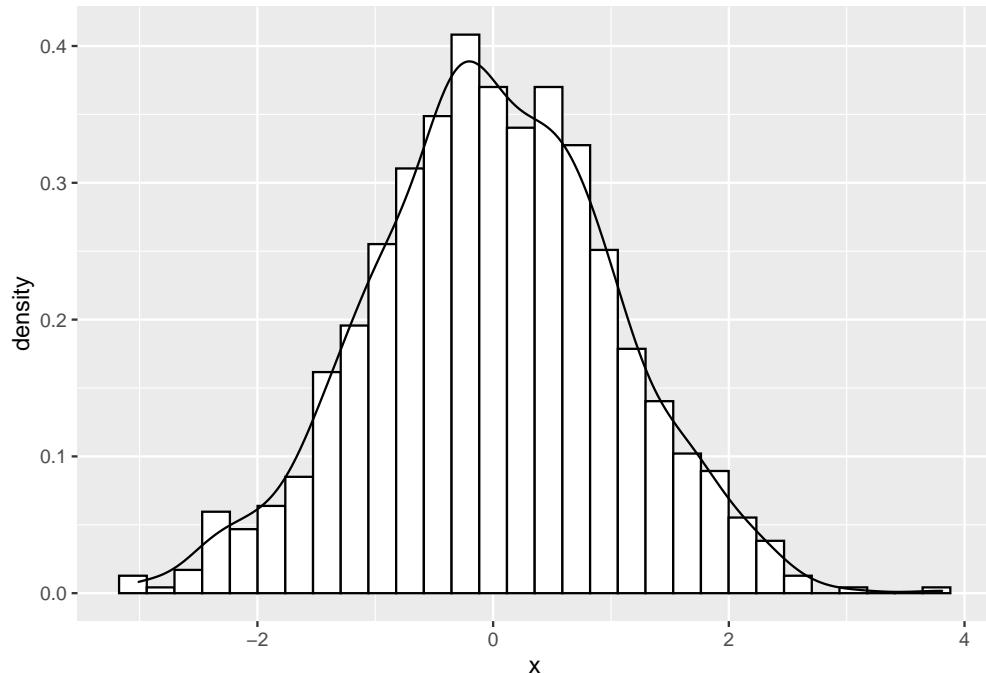
9.5.1 HISTOGRAMA CON ESTIMACIÓN KERNEL DE LA DENSIDAD

Para superponer una densidad de kernel sobre un histograma, se tendrá que pasar la función `aes(y = ..density..)` a la función `geom_histogram()` y usar la función `geom_density()`, como el siguiente ejemplo.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1)
x = rnorm(1000)
df = data.frame(x)

# Histograma con densidad
ggplot(df, aes(x = x)) +
  geom_histogram(aes(y = ..density..),
                 colour = 1, fill = "white") +
  geom_density()
```



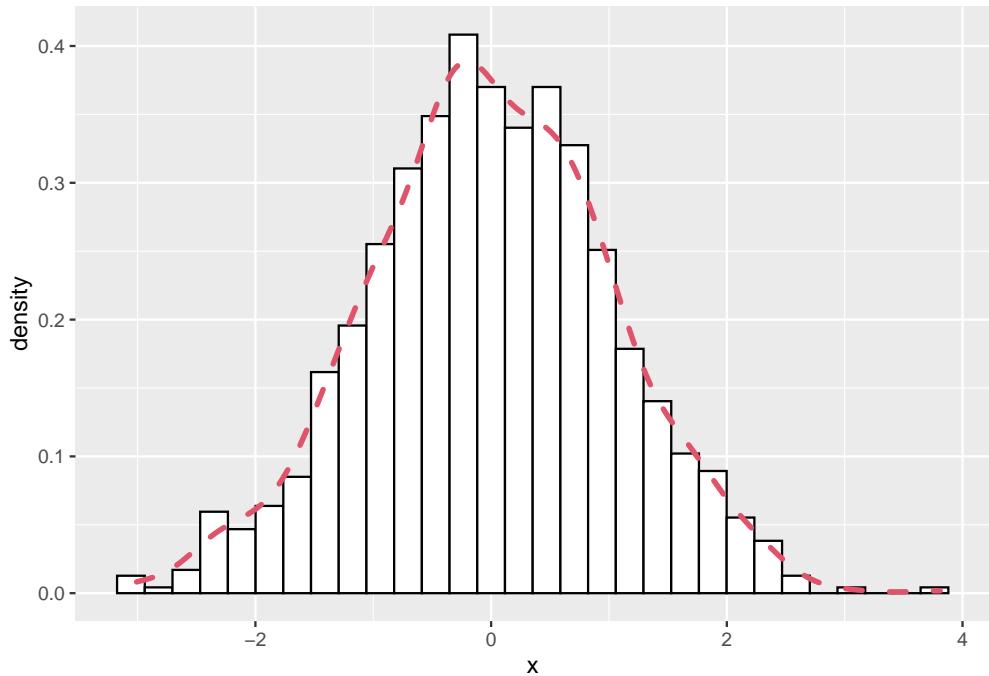
9.5.2 PERSONALIZACIÓN DE LA CURVA

El color, ancho y tipo de línea de la densidad de kernel se puede personalizar con `colour`, `lwd` y `linetype`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1)
x = rnorm(1000)
df = data.frame(x)

# Histograma con densidad
ggplot(df, aes(x = x)) +
  geom_histogram(aes(y = ..density..),
                 colour = 1, fill = "white") +
  geom_density(lwd = 1.2,
              linetype = 2,
              colour = 2)
```



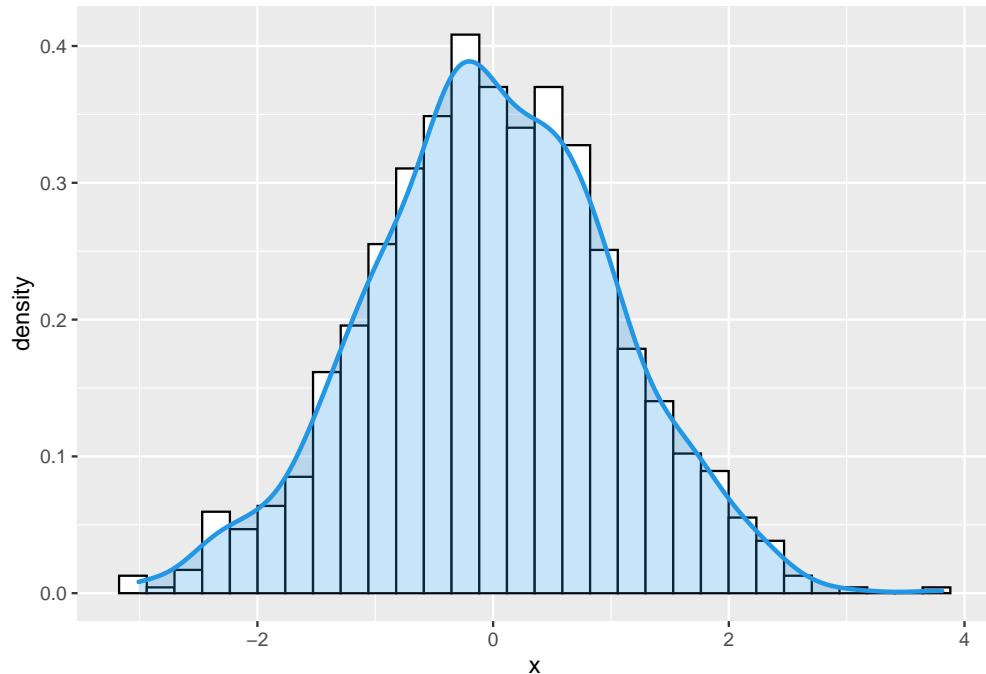
9.5.3 CURVA DE DENSIDAD CON ÁREA COLOREADA

También se puede sombrear el área debajo de la curva, especificando un color con el argumento `fill` de la función `geom_density()`. Es recomendable usar un nivel de transparencia (entre 0 y 1) con el argumento `alpha`, de modo que el histograma permanezca visible.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1)
x = rnorm(1000)
df = data.frame(x)

# Histograma con densidad
ggplot(df, aes(x = x)) +
  geom_histogram(aes(y = ..density..),
                 colour = 1, fill = "white") +
  geom_density(lwd = 1, colour = 4,
              fill = 4, alpha = 0.25)
```



9.6 HISTOGRAMA POR GRUPO EN ggplot2

9.6.1 DATOS DE MUESTRA

El siguiente data frame contiene una columna con dos distribuciones con diferente media y misma varianza y una variable categórica que representa qué observaciones corresponden a cada distribución.

```
set.seed(2)
x1 = rnorm(500)
x2 = rnorm(500, mean = 3)
x = c(x1, x2)
grupo = c(rep("G1", 500), rep("G2", 500))

df = data.frame(x, group = grupo)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
```

```
kable_styling(latex_options = c("striped",
                                "condensed", "HOLD_position"),
              position = "center",
              full_width = FALSE)
```

x	group
-0.8969145	G1
0.1848492	G1
1.5878453	G1
-1.1303757	G1
-0.0802518	G1
0.1324203	G1
0.7079547	G1
-0.2396980	G1
1.9844739	G1
-0.1387870	G1

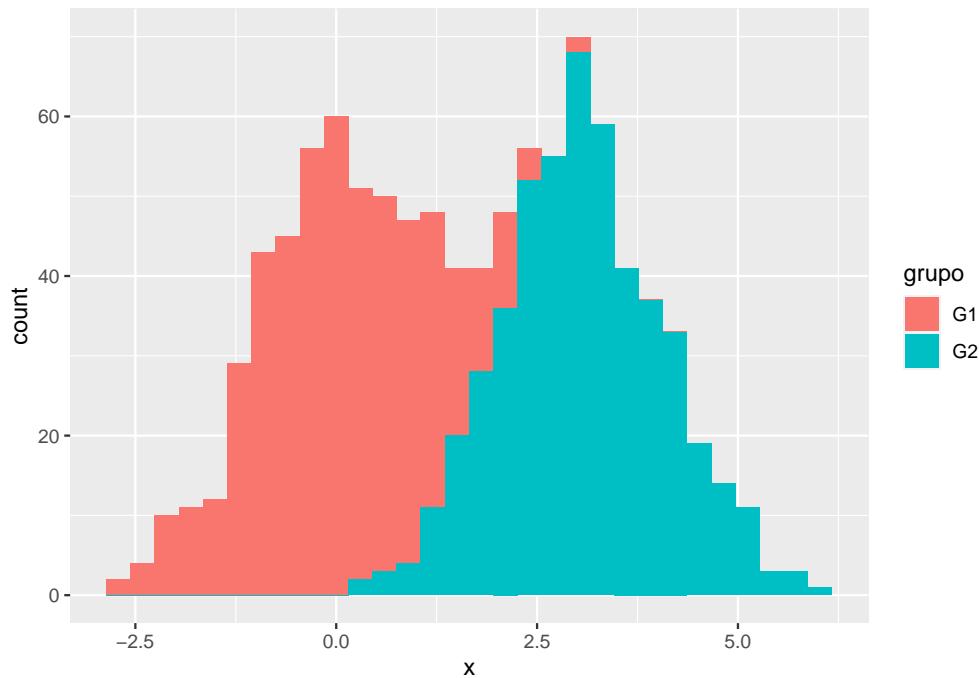
9.6.2 HISTOGRAMA POR GRUPO CON `geom_histogram()`

9.6.2.1 FILL

Para crear un histograma por grupo, se tendrá que pasar una variable numérica y la categórica dentro de la función `aes()` y usar la función `geom_histogram()` de la siguiente manera.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo)) +
  geom_histogram()
```



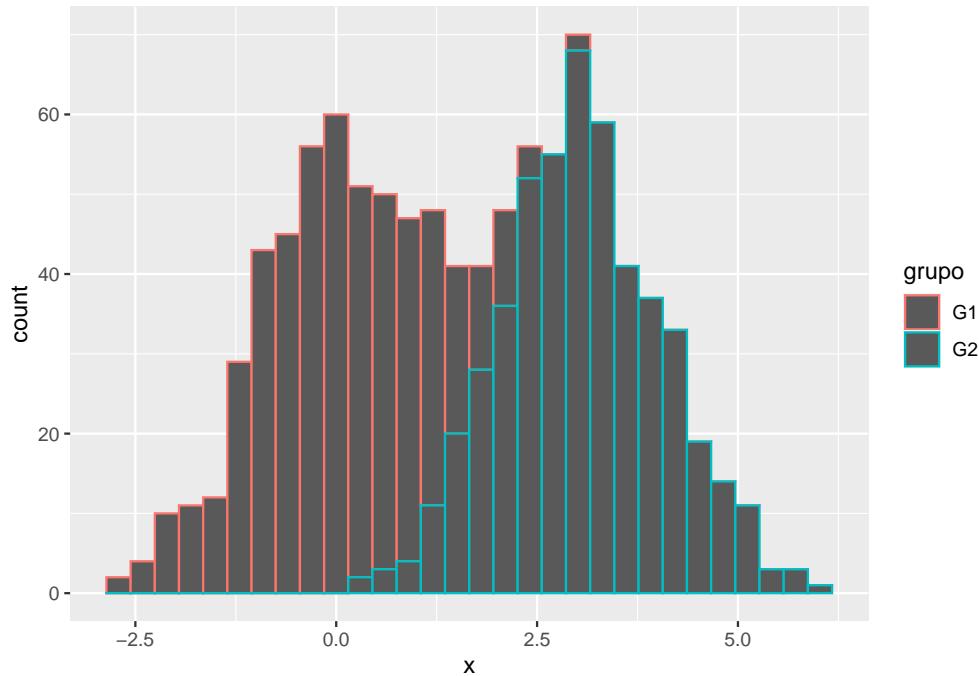
9.6.2.2 COLOUR

También se puede pasar la variable categórica al argumento `colour`, de modo que los bordes de cada histograma sean de un color diferente.¹⁴

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, colour = grupo)) +
  geom_histogram()
```

¹⁴Por defecto, si los histogramas se solapan, los valores se apilarán. Otra opción es cambiar la posición a `identity` (y usar colores con transparencia) o a `dodge` como en los siguientes ejemplos.

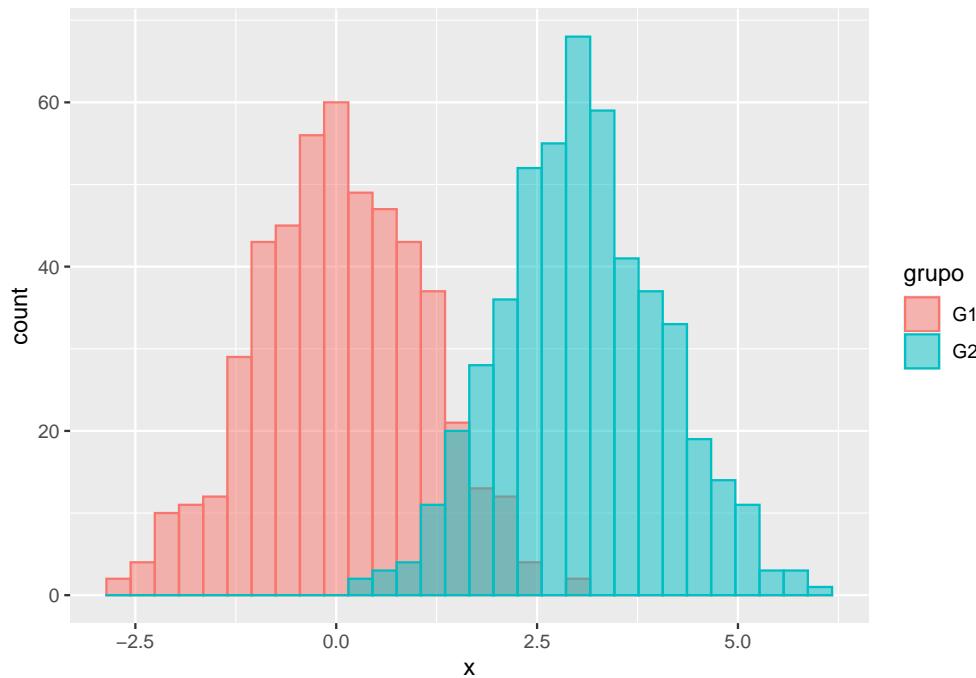


9.6.2.3 POSICIÓN IDENTITY

Establecer `position = "identity"` es lo más recomendable en la mayoría de los casos, pero hay que recordar que al usar un color transparente con `alpha` será para que ambos histogramas sean completamente visibles.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo , colour = grupo)) +
  geom_histogram(alpha = 0.5, position = "identity")
```

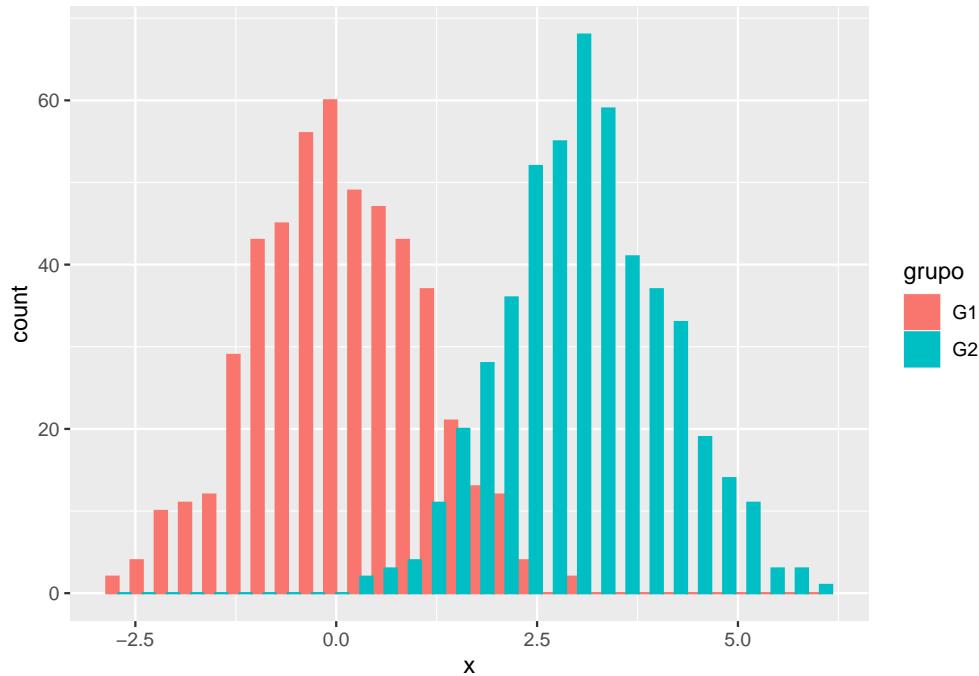


9.6.2.3 POSICIÓN DODGE

Otra opción es usar el argumento `position = "dodge"`, que agrega un espacio entre cada barra de forma que se puedan ver ambos histogramas.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo , colour = grupo)) +
  geom_histogram(position = "dodge")
```



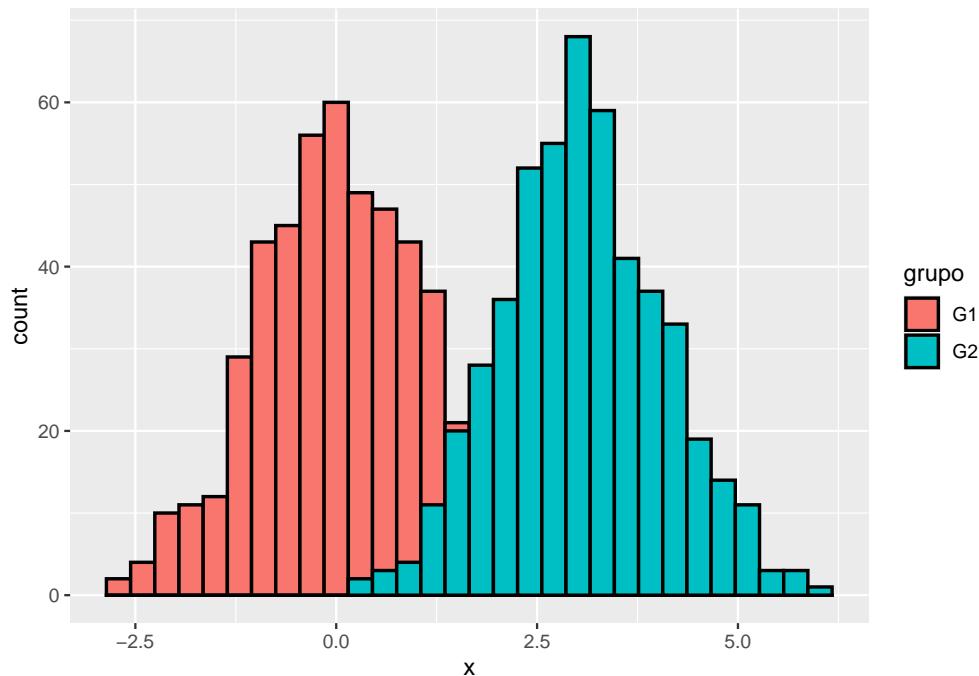
9.6.3 PERSONALIZACIÓN DE LOS COLORES

9.6.3.1 COLOR DEL BORDE

Si se establece la función `fill` dentro de `aes()` pero no `colour`, se puede cambiar el color del borde para todos los histogramas; así como el ancho y el tipo de línea con los argumentos de `geom_histogram()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo)) +
  geom_histogram(colour = "black",
                 lwd = 0.75,
                 linetype = 1,
                 position = "identity")
```

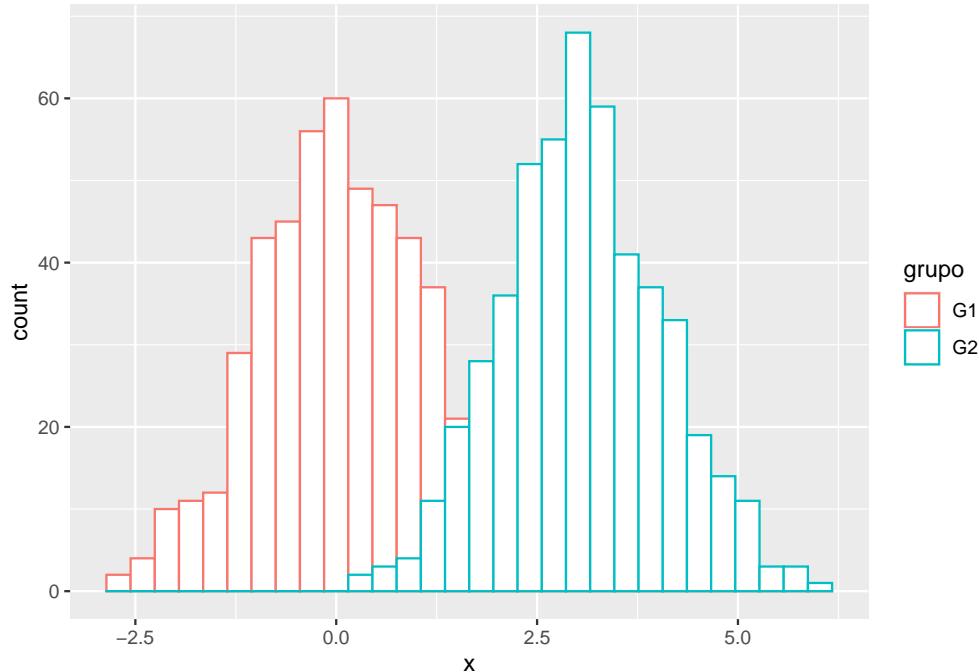


9.6.3.2 COLOR DEL BORDE

Si se establece `colour` pero no `fill`, se puede cambiar el color de fondo de todos los histogramas con el argumento `fill` de la función `geom_histogram()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, colour = grupo)) +
  geom_histogram(fill = "white",
                 position = "identity")
```

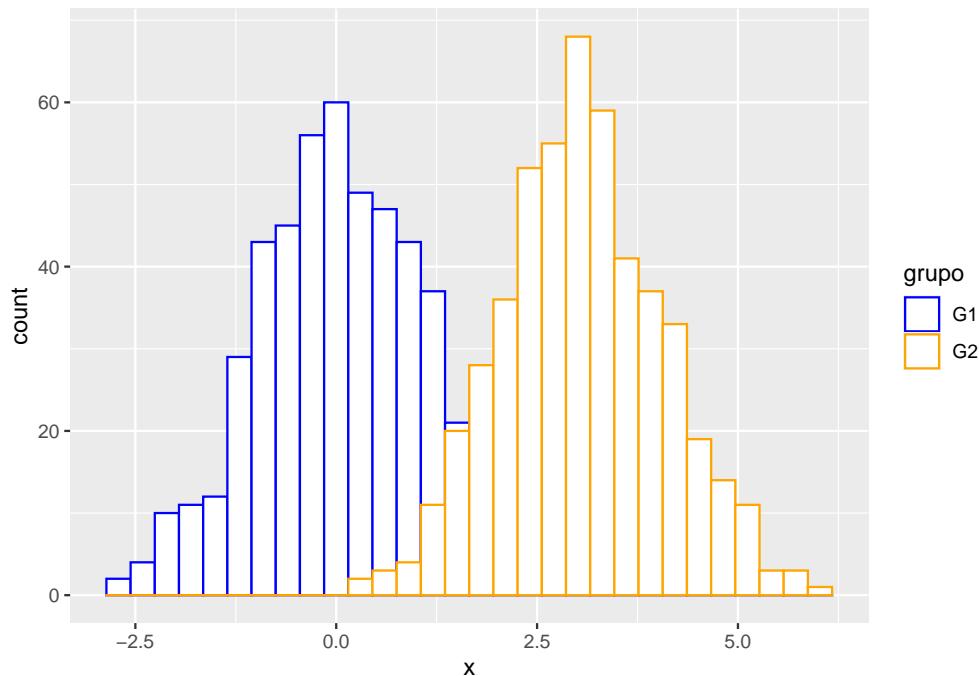


9.6.4 PERSONALIZAR EL COLOR DEL BORDE PARA CADA GRUPO

El color de los bordes se pueden personalizar para cada histograma con la función `scale_color_manual()`. Si se quiere usar la paleta predefinida se puede usar, por ejemplo, `scale_color_brewer()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, colour = grupo)) +
  geom_histogram(fill = "white",
                 position = "identity") +
  scale_color_manual(values = c("blue", "orange"))
```

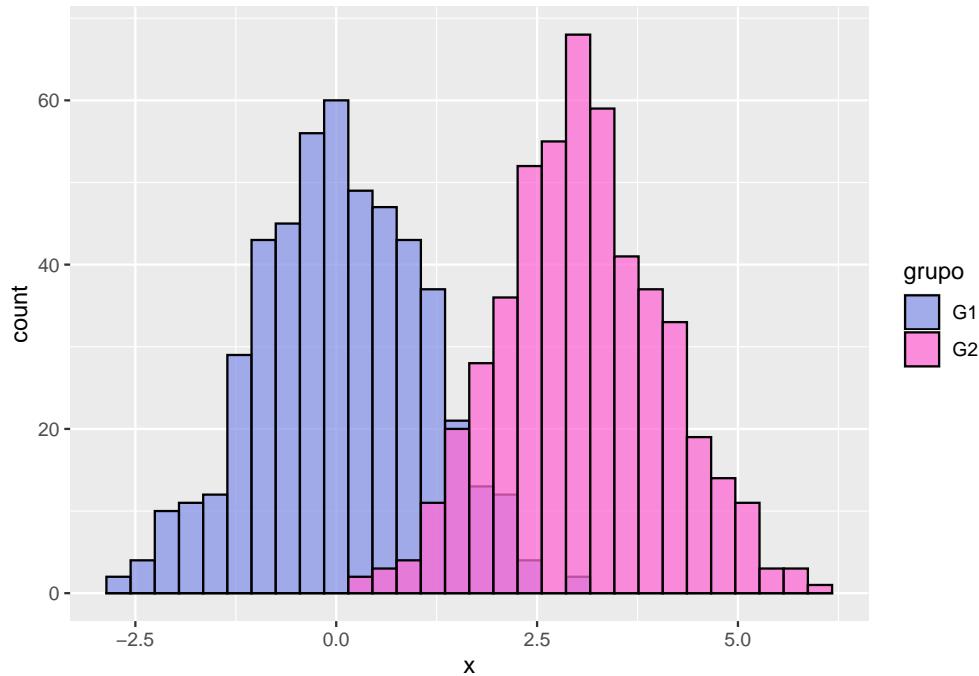


9.6.4.1 PERSONALIZAR EL COLOR DE FONDO PARA CADA GRUPO

De manera similar a personalizar los colores de los bordes, los colores de fondo se pueden cambiar con la función `scale_fill_manual()` o una función equivalente.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo)) +
  geom_histogram(color = 1, alpha = 0.75,
                 position = "identity") +
  scale_fill_manual(values = c("#8795E8", "#FF6AD5"))
```



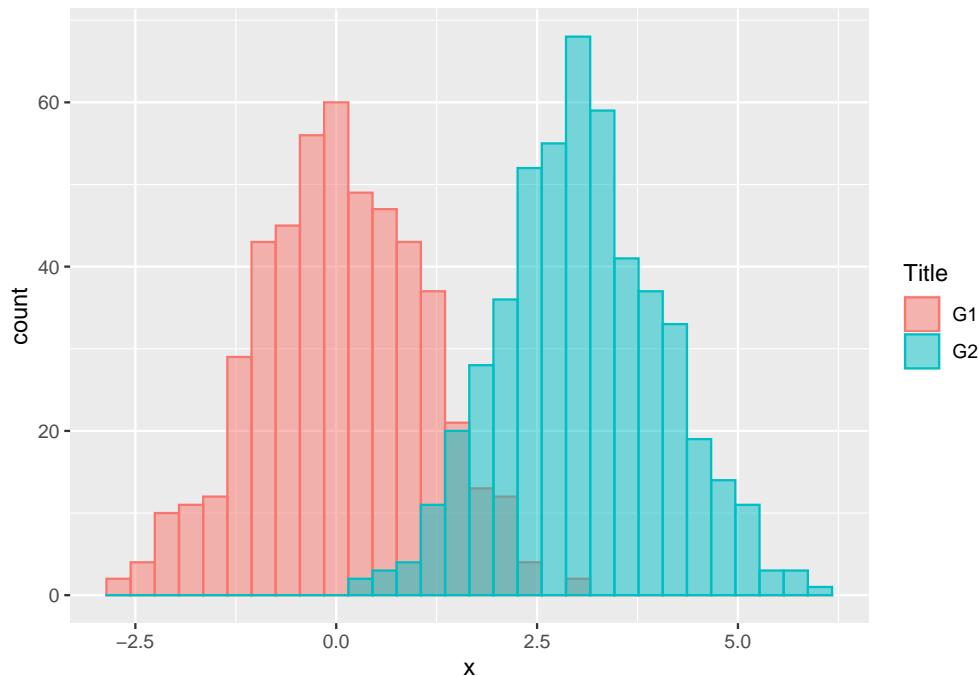
9.6.5 PERSONALIZAR LA LEYENDA

9.6.5.1 TÍTULO DE LA LEYENDA

El título de la leyenda es el nombre de la columna de la variable categórica del conjunto de datos. Pueden ser cambiados con los argumentos `fill` y/o `colour` de la función `guides()`. Como se están pasando los argumentos `fill` y `colour` dentro de la función `aes()` estableciendo ambas para que no se creen dos leyendas.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo , colour = grupo)) +
  geom_histogram(alpha = 0.5, position = "identity") +
  guides(fill = guide_legend(title = "Title"),
        colour = guide_legend(title = "Title"))
```

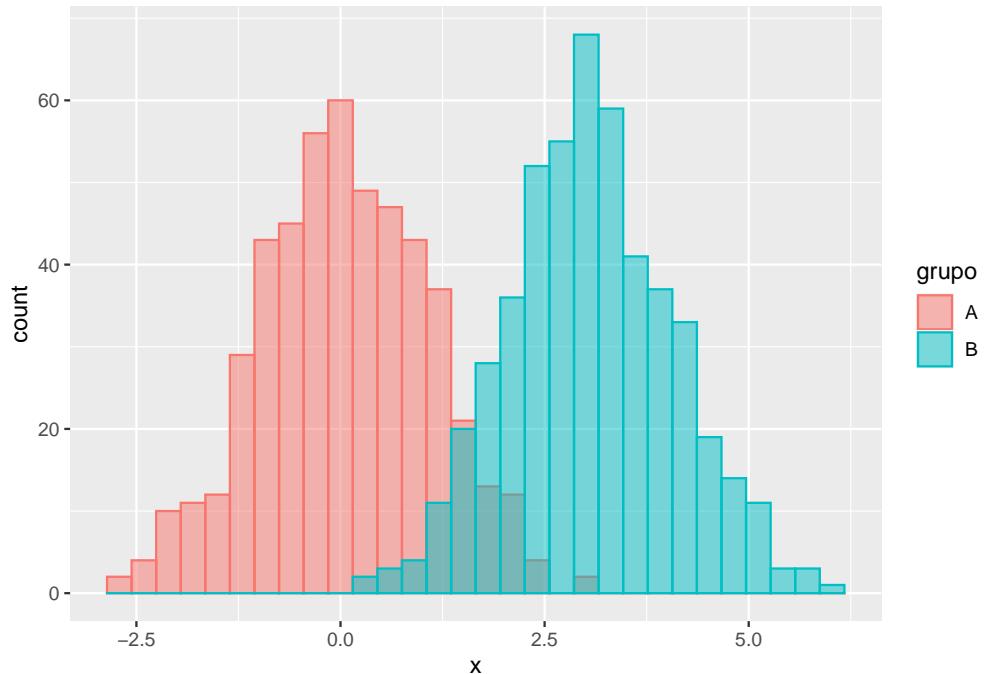


9.6.5.2 ETIQUETAS DE LA LEYENDA

La leyenda mostrará por defecto los nombres de la variable categórica, pero se pueden cambiar con la función `scale_color_discrete()` y/o `scale_fill_discrete()`. Hay que tener en cuenta que esto dependerá del `aes()` que se establezca.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo, colour = grupo)) +
  geom_histogram(alpha = 0.5, position = "identity") +
  scale_color_discrete(labels = c("A", "B")) +
  scale_fill_discrete(labels = c("A", "B"))
```

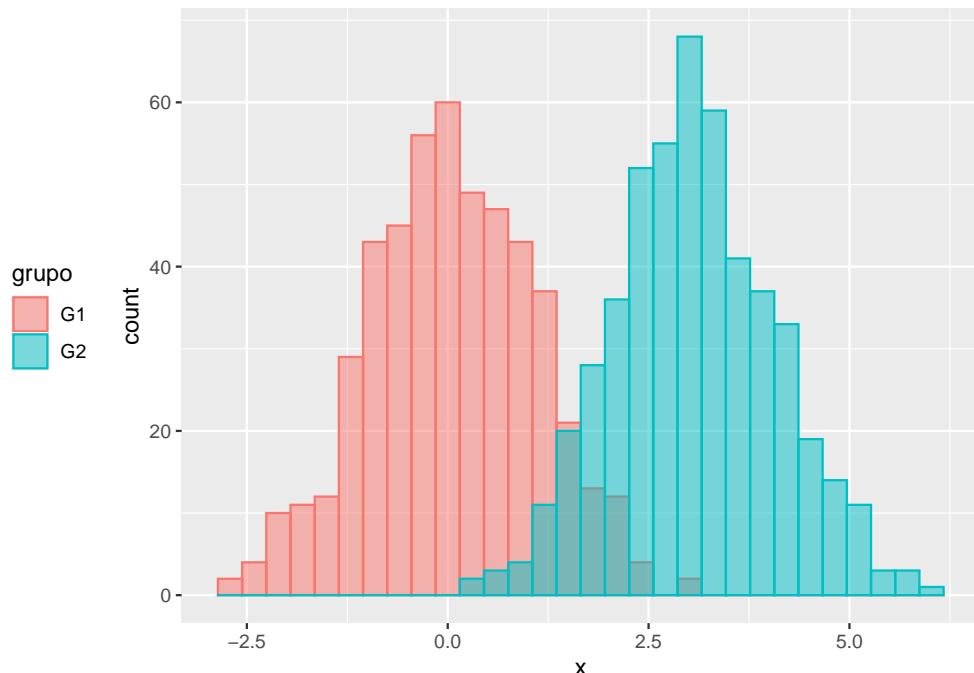


9.6.5.3 POSICIÓN DE LA LEYENDA

La posición por defecto de la leyenda es la derecha, pero se puede cambiar con el argumento `legend.position` de la función `theme()` como se muestra en el siguiente ejemplo:

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo , colour = grupo)) +
  geom_histogram(alpha = 0.5, position = "identity") +
  theme(legend.position = "left") # Izquierda
```

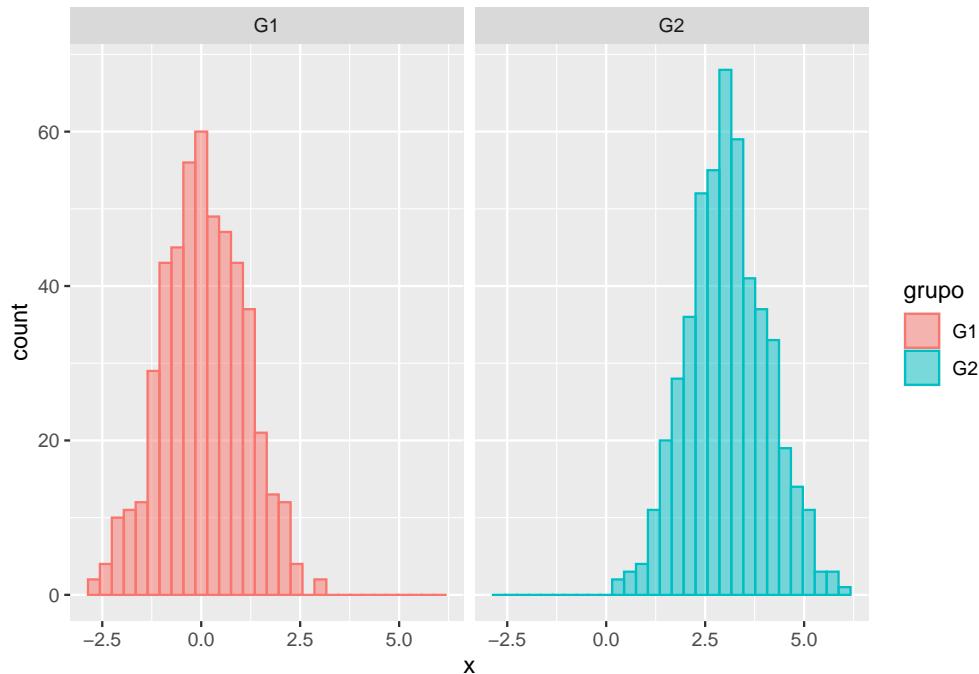


9.6.6 DIVIDIR POR DIAGRAMA CON LA FUNCIÓN `facet_grid()`

Por último, se muestra una forma dinámica de mostrar los histogramas en dos diagramas separados. Esto se hace gracias a la función `facet_grid()`, pasando por dentro la variable categórica, con el argumento `~`. Esto puede resultar útil para separar variables dentro del mismo data frame y hacer un análisis estadístico por variable categórica.

```
# install.packages("ggplot2")
library(ggplot2)

# Histograma por grupo en ggplot2
ggplot(df, aes(x = x, fill = grupo, colour = grupo)) +
  geom_histogram(alpha = 0.5, position = "identity") +
  facet_grid(~df$group)
```



9.7 NÚMERO Y ANCHO DE LAS BARRAS O CLASES DE UN HISTOGRAMA EN `ggplot2`

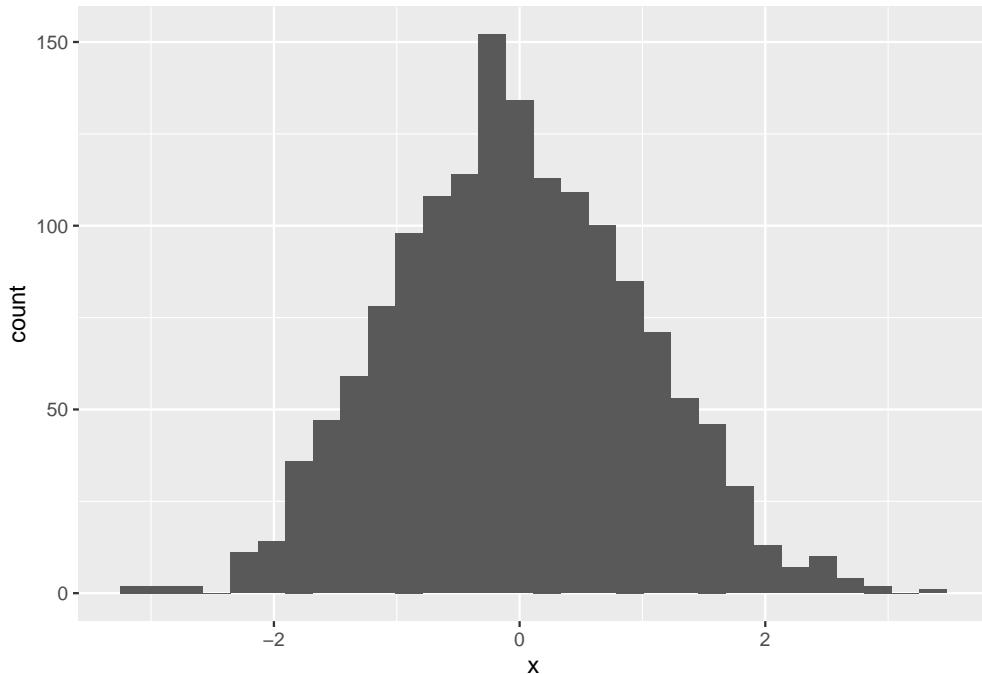
9.7.1 HISTOGRAMA POR DEFECTO

Por defecto, los cálculos para crear un histograma con `geom_histogram()` a través de la función `stat_bin()` usan 30 barras o clases, lo que no siempre es un buen valor por defecto.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
x = rnorm(1500)
df = data.frame(x)

# Histograma por defecto
ggplot(df, aes(x = x)) +
  geom_histogram()
```



Esta es la razón por las que se recibe el mensaje cada vez que se crea un histograma por defecto en ggplot2: `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

Las posibles opciones para solucionar este problema son seleccionar el número de clases o barras con el argumento `bins` o modificar el ancho de cada barra con el argumento `binwidth`.

9.7.2 EL ARGUMENTO `bins`

El número de clase o barras del histograma se puede personalizar con el argumento `bins` de la función `geom_histogram()`. En este ejemplo 15 clases parece una buena elección, mientras que 50 son demasiadas.

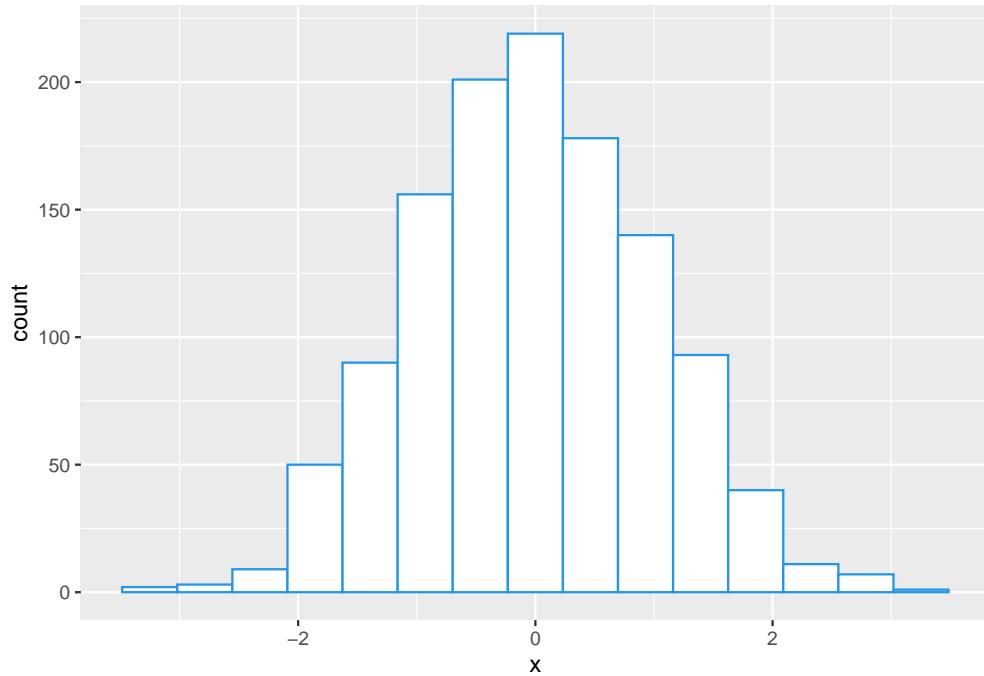
9.7.2.1 CON 15 CLASES

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
x = rnorm(1200)
df = data.frame(x)

# Clases del histograma
ggplot(df, aes(y = x)) +
```

```
geom_histogram(colour = 4, fill = "white",
               bins = 15)
```

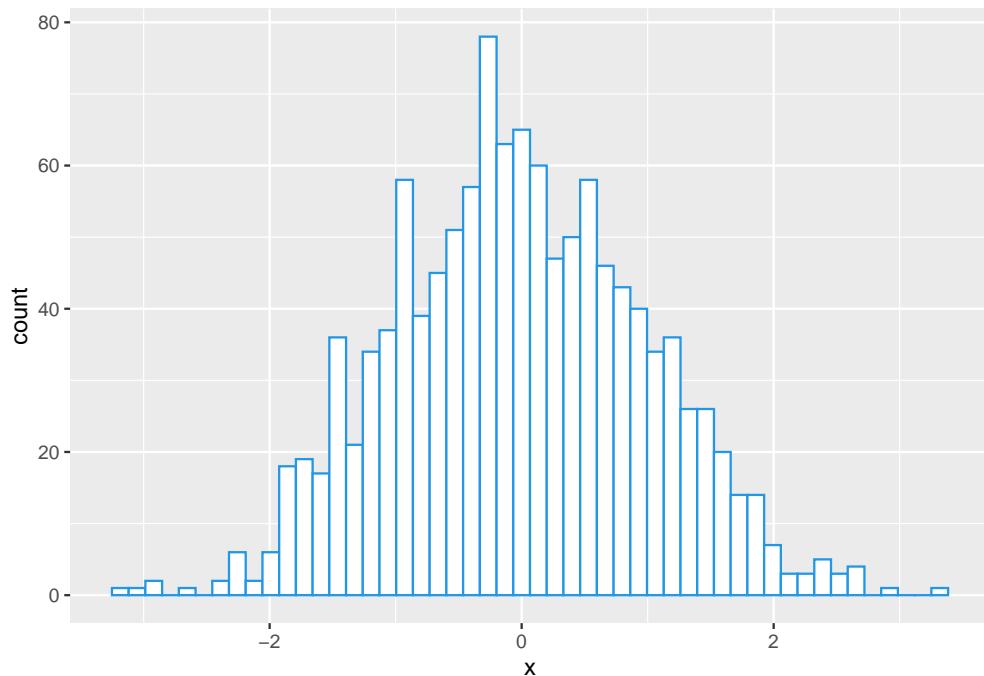


9.7.2.2 CON 50 CLASES

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
x = rnorm(1200)
df = data.frame(x)

# Clases del histograma
ggplot(df, aes(x = x)) +
  geom_histogram(colour = 4, fill = "white",
                 bins = 50)
```



9.7.3 el argumento `binwidth`

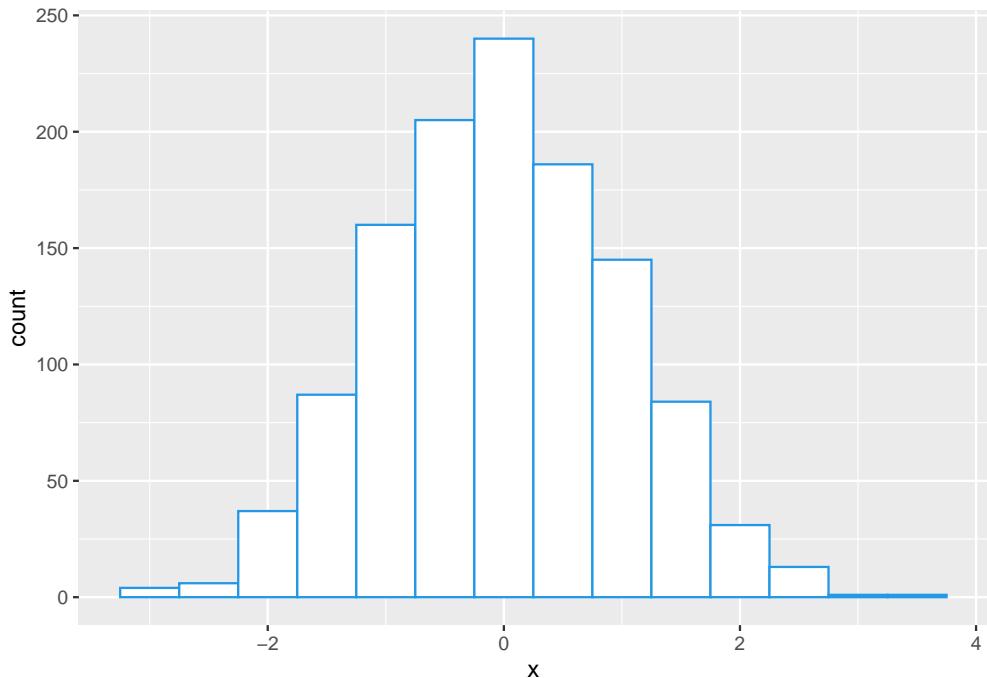
La otra opción es usar el argumento `binwidth` de la función `geom_histogram()`. Este argumento controla el ancho de cada barra sobre el eje X. Hay que tener en cuenta que **este argumento sobrescribe al argumento bins**.

9.7.3.1 BINWIDTH DE 0.5

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
x = rnorm(1200)
df = data.frame(x)

# Ancho de las barras del histograma
ggplot(df, aes(x = x)) +
  geom_histogram(colour = 4, fill = "white",
                binwidth = 0.5)
```



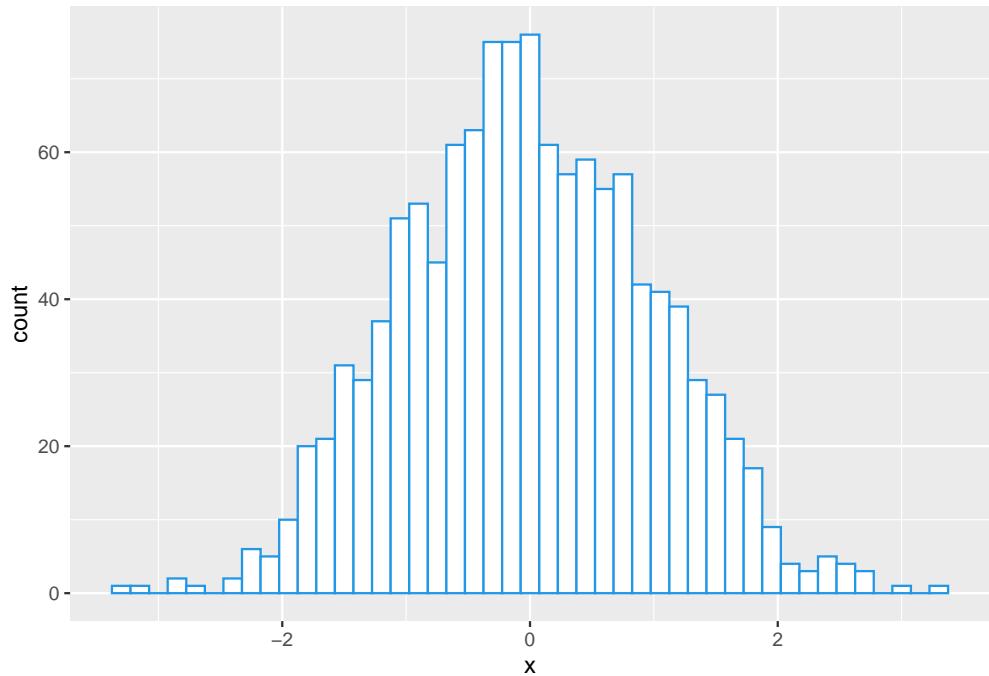
9.7.3.2 binwidth de 0.15¹⁵

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
x = rnorm(1200)
df = data.frame(x)

# Ancho de las barras del histograma
ggplot(df, aes(x = x)) +
  geom_histogram(colour = 4, fill = "white",
                 binwidth = 0.15)
```

¹⁵La función `hist()` de R base usa el **método de Sturges** para calcular el número de clases, siendo este un buen valor por defecto.



9.8 DIAGRAMA DE VIOLÍN CON MEDIA EN ggplot22

9.8.1 DATOS

Se considera el conjunto de datos de la base de R de `ToothGrowth` y transformar la columna `dose` en un factor.

```
# Conjunto de datos de muestra
df = ToothGrowth
df$dose = as.factor(df$dose)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE,format = "latex") %>%
  kable_styling(latex_options = c("striped",
                                  "condensed","HOLD_position"),
  position = "center",
  full_width = FALSE)
```

	len	supp	dose
	4.2	VC	0.5
	11.5	VC	0.5
	7.3	VC	0.5
	5.8	VC	0.5
	6.4	VC	0.5
	10.0	VC	0.5
	11.2	VC	0.5
	11.2	VC	0.5
	5.2	VC	0.5
	7.0	VC	0.5

9.8.2 AGREGANDO LA MEDIA CON LA FUNCIÓN `stat_summary()`

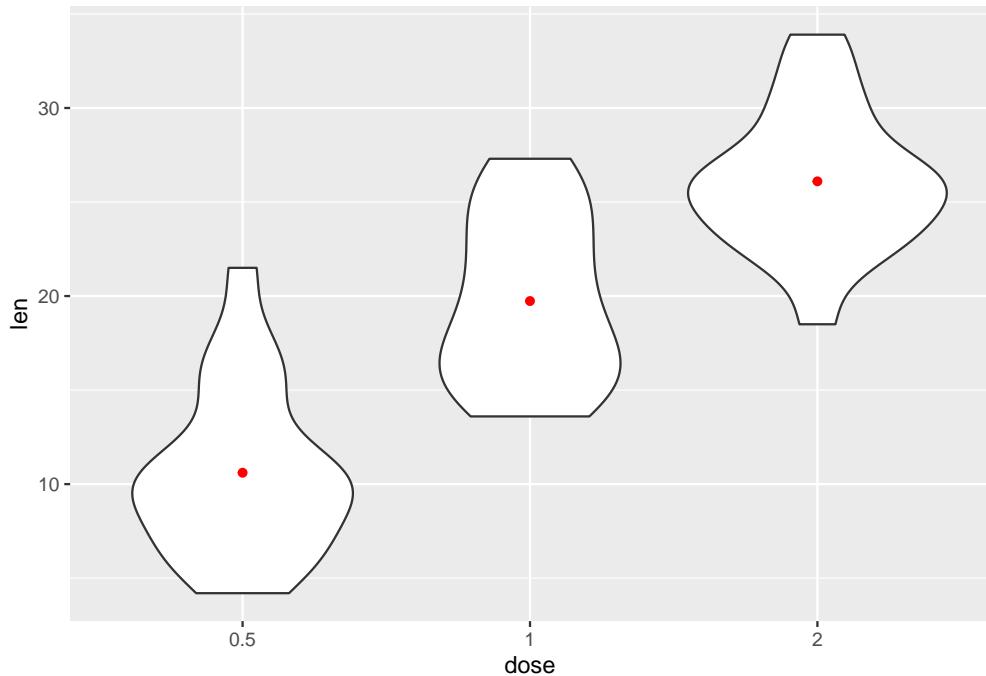
Para añadir la media a un diagrama de violín se tendrá que usar la función `stat_summary()` y se especifica la función al ser calculada, el `geometrico` a ser usado y los argumentos `geom`.

9.8.2.1 MEDIA COMO PUNTO

En caso de que se quiera mostrar la media con puntos se puede pasar la función `mean()` y establecer el argumento `"point"` como `geom`. Hay que recordar que se puede personalizar otros argumentos como `shape` (forma) y `size` (tamaño).

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len)) +
  geom_violin() +
  stat_summary(fun = "mean",
              geom = "point",
              color = "red")
```

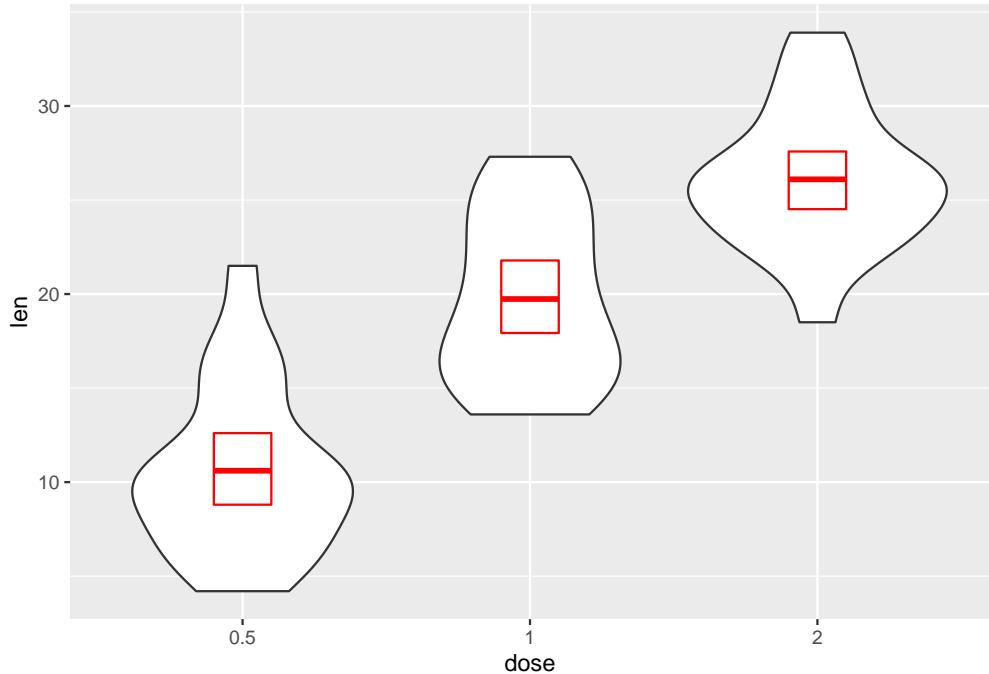


9.8.2.2 MEDIA COMO CROSSBAR

También se puede agregar intervalos de confianza a la media. Para ello, se tiene que pasar la función (`a_fun.data()`) que calcule los intervalos de confianza para la media, como `mean_cl_boot()` para un bootstrap no paramétrico. Solamente se escribe `?mean_cl_boot` para obtener información sobre más funciones relacionadas.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len)) +
  geom_violin() +
  stat_summary(fun.data = "mean_cl_boot", geom = "crossbar",
              colour = "red", width = 0.2)
```

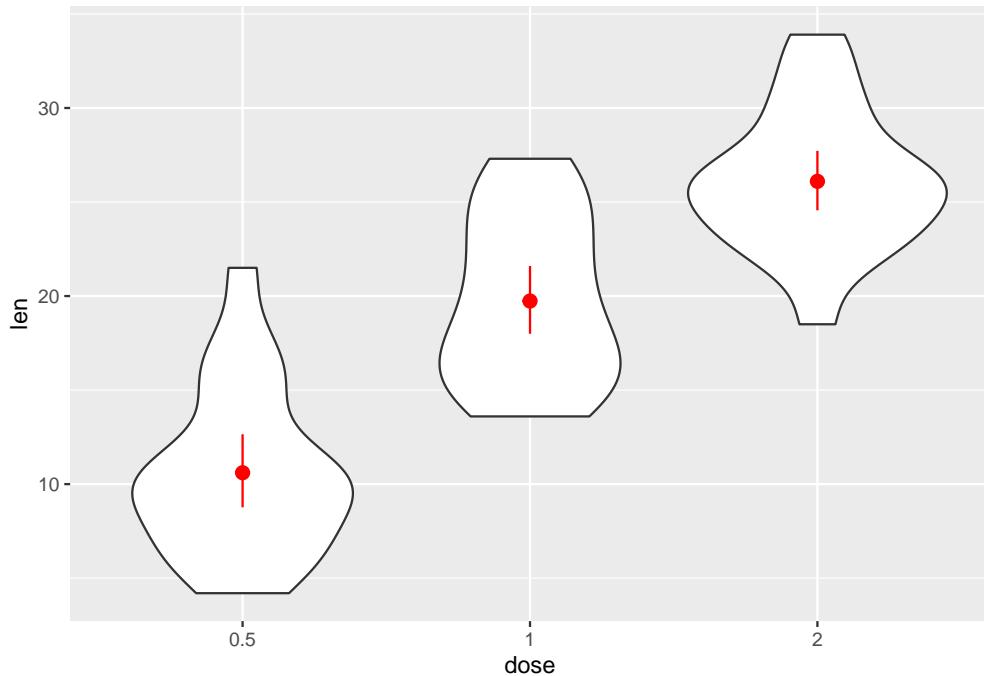


9.8.2.3 MEDIA CON BARRAS DE ERROR

La desviación típica también se puede mostrar con barras de error (point range) en lugar de un crossbow.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len)) +
  geom_violin() +
  stat_summary(fun.data = "mean_cl_boot", geom = "pointrange",
              colour = "red")
```



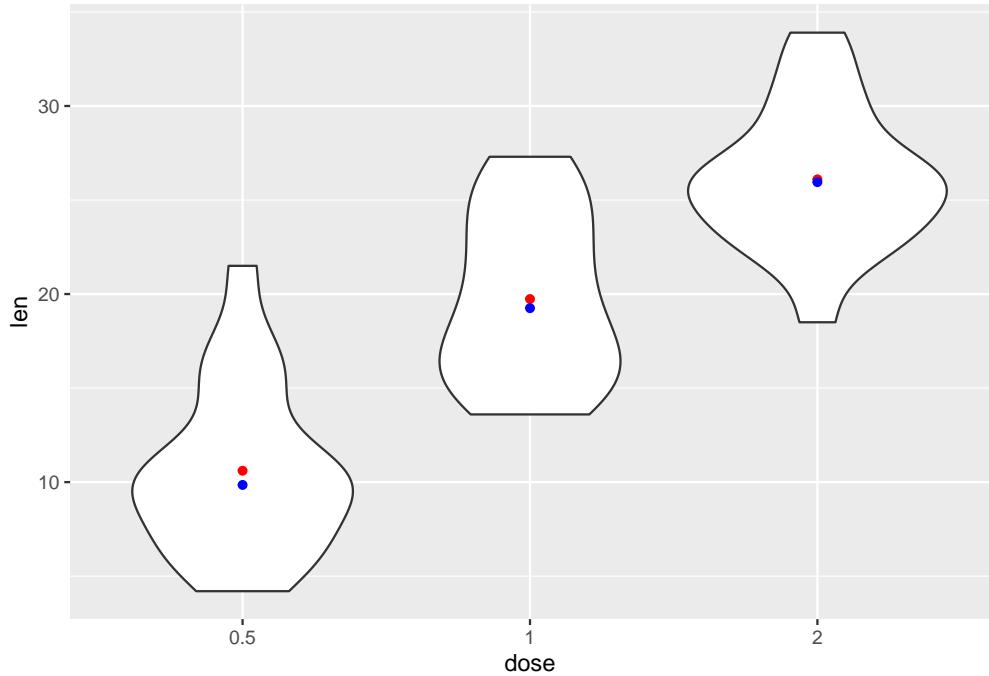
9.8.3 MEDIA Y MEDIANA

9.8.3.1 MEDIA Y MEDIANA

Si se agrega otra capa con `stat_summary()` pero con la mediana, se puede tener ambas métricas. Puede establecerse el geom que se quiera como en la sección anterior.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len)) +
  geom_violin() +
  stat_summary(fun = "mean",
              geom = "point",
              color = "red") +
  stat_summary(fun = "median",
              geom = "point",
              color = "blue")
```

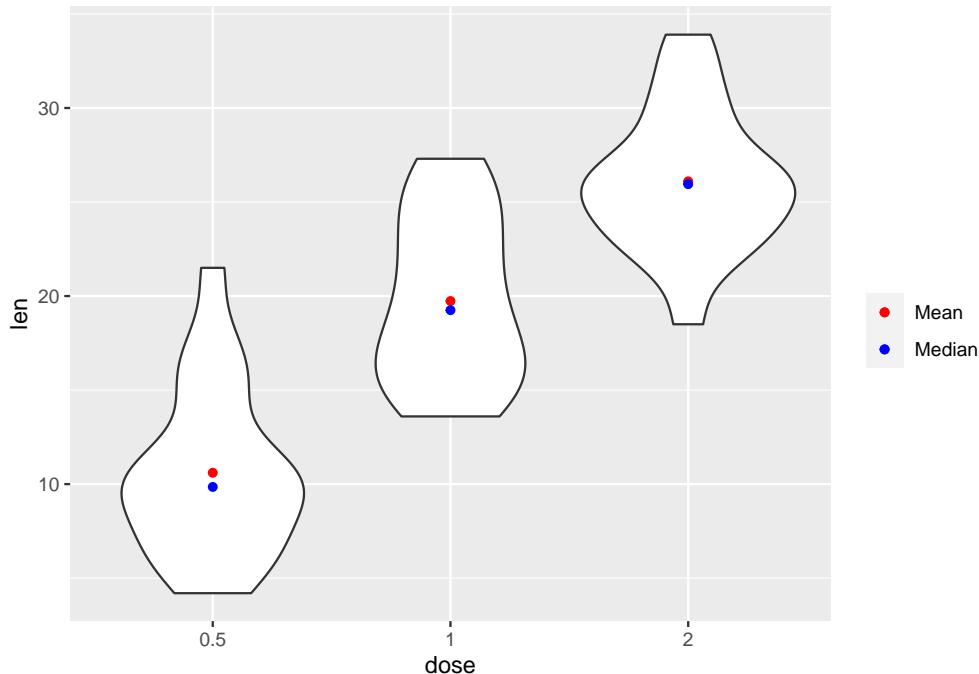


9.8.3.2 MEDIA Y MEDIANA (CON LEYENDA)

Hay que tener en cuenta que estableciendo una función `aes()` se creará una leyenda. Esta leyenda se puede personalizar, por ejemplo, con la función `scale_color_manual()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len)) +
  geom_violin() +
  stat_summary(fun = "mean",
              geom = "point",
              aes(color = "Mean")) +
  stat_summary(fun = "median",
              geom = "point",
              aes(color = "Median")) +
  scale_colour_manual(values = c("red", "blue"), # Colores
                      name = "") # Eliminar el título de la leyenda
```



9.9 GRÁFICO DE VIOLÍN CON PUNTOS EN `ggplot2`

9.9.1 DATOS DE MUESTRA

El siguiente conjunto de datos contiene el efecto de la vitamina C en el crecimiento de los dientes. La columna `dose` se ha convertido a factor y será usada como variable categórica.

```
# Conjunto de datos de muestra
df = ToothGrowth
df$dose = as.factor(df$dose)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE,format = "latex") %>%
  kable_styling(latex_options = c("striped",
                                  "condensed","HOLD_position"),
                position = "center",
                full_width = FALSE)
```

	len	supp	dose
	4.2	VC	0.5
	11.5	VC	0.5
	7.3	VC	0.5
	5.8	VC	0.5
	6.4	VC	0.5
	10.0	VC	0.5
	11.2	VC	0.5
	11.2	VC	0.5
	5.2	VC	0.5
	7.0	VC	0.5

9.9.2 PUNTOS DE DATOS CON RUIDO ALEATORIO

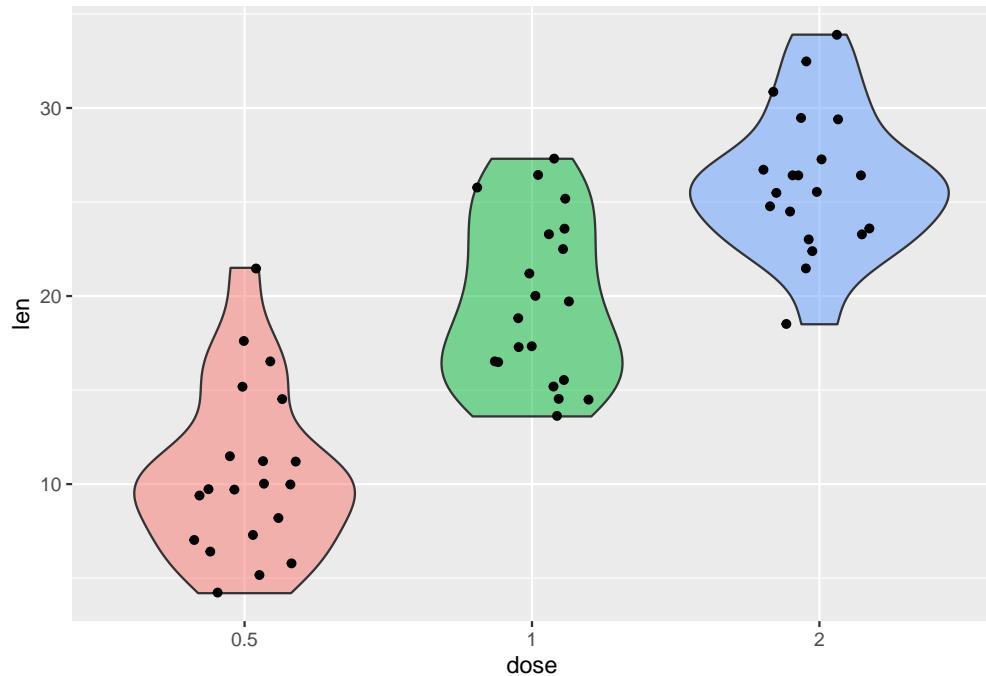
9.9.2.1 OPCIÓN 1

Las observaciones se pueden agregar sobre un diagrama de violín con la función `geom_point()`. Sin embargo, es más recomendable añadirlos con algo de ruido aleatorio con `position_jitter()`, donde `seed` es **la semilla del generador de números pseudoaleatorios¹⁶**(opcional), y el argumento `width`, siendo la cantidad de ruido a lo ancho.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_point(position = position_jitter(seed = 1, width = 0.2)) +
  theme(legend.position = "none")
```

¹⁶Se puede acceder al enlace al dar click en las letras en negritas.

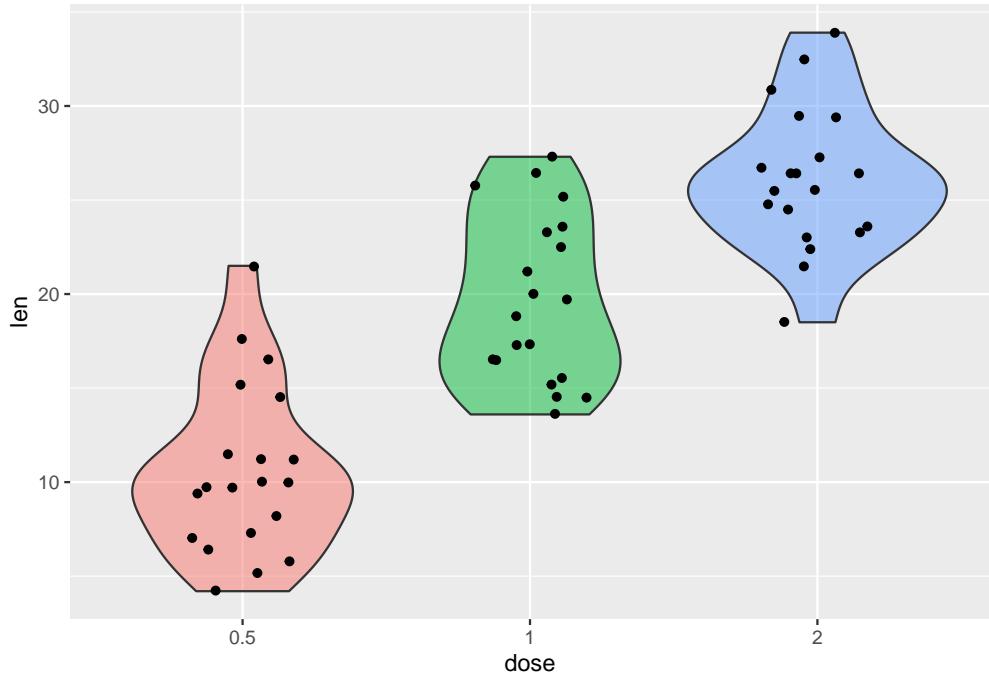


9.9.2.2 OPCIÓN 2

Se puede obtener el mismo resultado usando la función `geom_jitter()`

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_jitter(position = position_jitter(seed = 1, width = 0.2)) +
  theme(legend.position = "none")
```

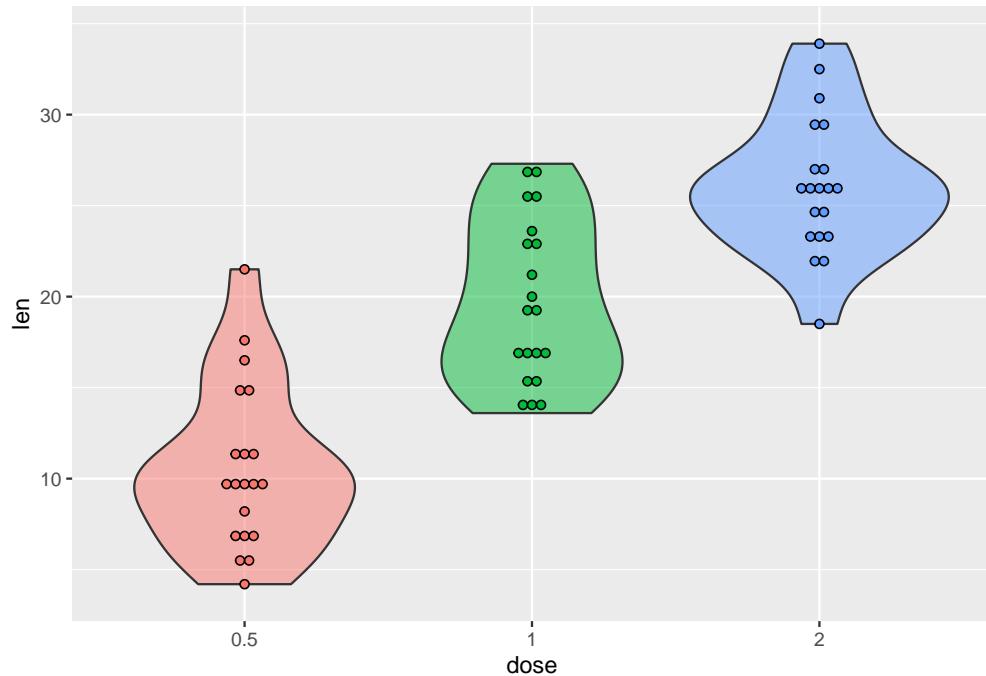


9.9.3 AGREGAR UN DOT PLOT

Si se desea crear un gráfico de violín con puntos en ggplot2, se puede usar la función `geom_dotplot()`, estableciendo los argumentos `binaxis = "y"` y `stackdir = "center"`. Hay que tener en cuenta que el argumento `dotsize` controla el tamaño de los puntos.

```
# install.packages("ggplot2")
library(ggplot2)

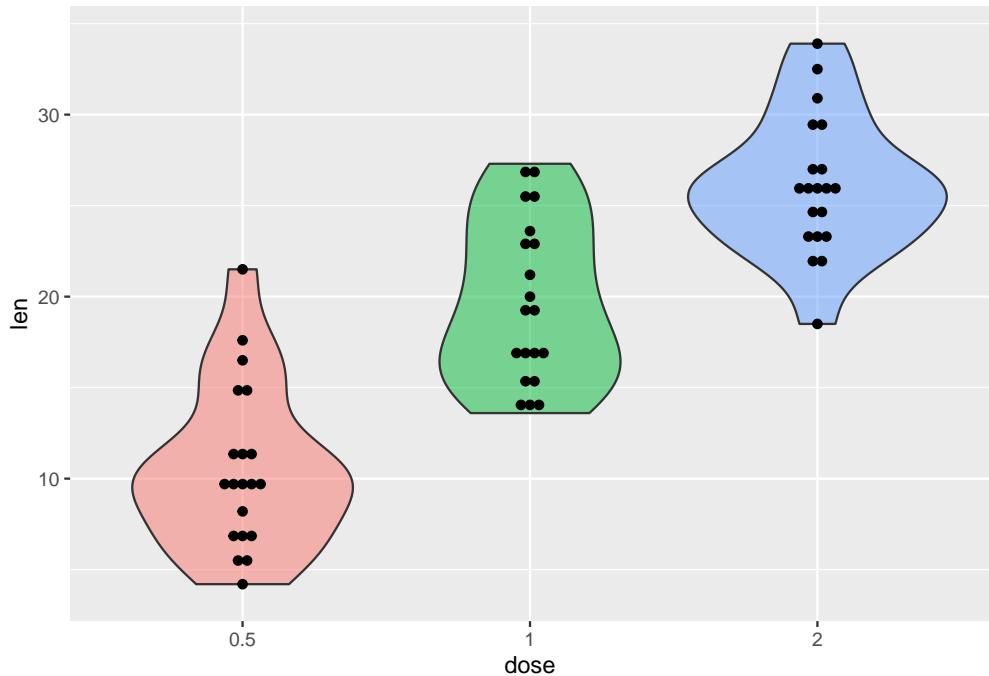
ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_dotplot(binaxis = "y",
               stackdir = "center",
               dotsize = 0.5) +
  theme(legend.position = "none")
```



Los puntos, por defecto, son del mismo color que los grupos. Para sobreescribirlo se puede especificar un color dentro de la función `geom_dotplot()`

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_dotplot(binaxis= "y",
               stackdir = "center",
               dotsizes = 0.5,
               fill = 1) +
  theme(legend.position = "none")
```

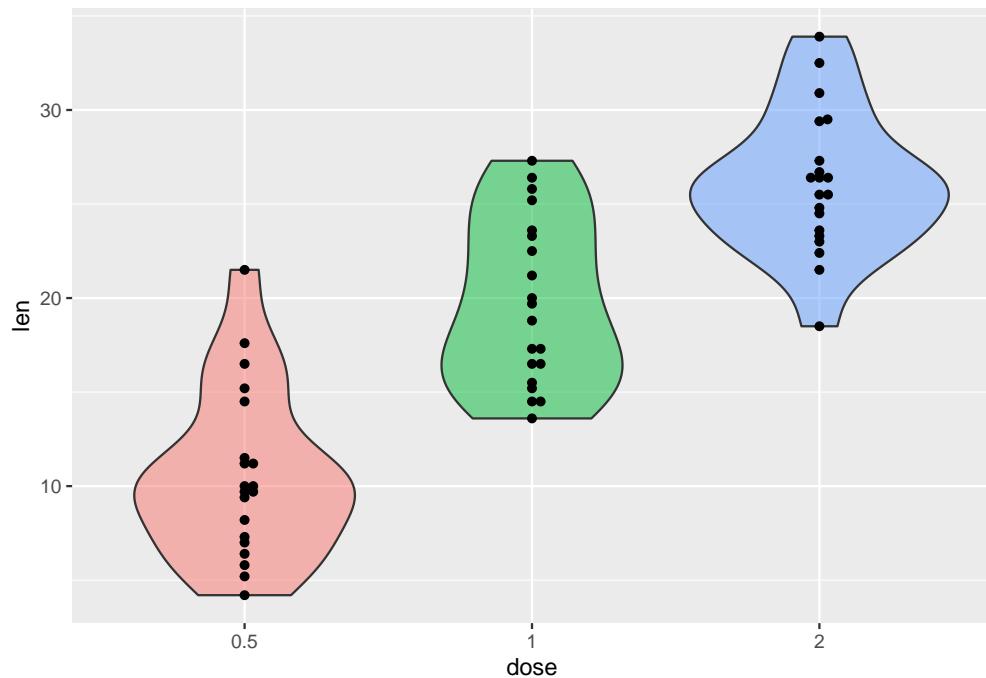


9.9.4 AGREGAR UN BEESWARM

Una alternativa para agregar puntos de datos sobre un diagrama de violín en `ggplot2` es usar la función `geom_beeswarm()` del paquete `ggbeeswarm`.

```
# install.packages("ggplot2")
# install.packages("ggbeeswarm")
library(ggplot2)
library(ggbeeswarm)

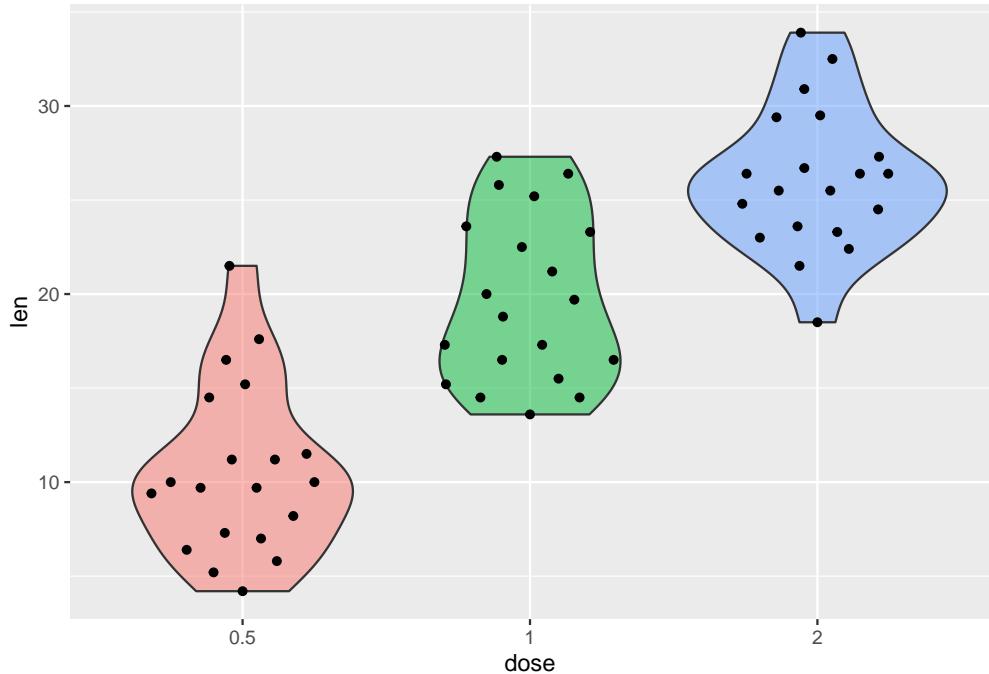
ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_beeswarm() +
  theme(legend.position = "none")
```



El paquete `ggbeeswarm` también contiene una función llamada `geom_quasirandom()`, que puede ser usada para añadir los puntos dentro de los violines, tal y como se puede ver en el siguiente ejemplo:

```
# install.packages("ggplot2")
# install.packages("ggbeeswarm")
library(ggplot2)
library(ggbeeswarm)

ggplot(df, aes(x = dose, y = len, fill = dose)) +
  geom_violin(alpha = 0.5) +
  geom_quasirandom() +
  theme(legend.position = "none")
```



9.10 GRÁFICO DE VIOLÍN POR GRUPO EN `ggplot2`

9.10.1 DATOS DE MUESTRA

El conjunto de datos usado en los siguientes ejemplos es `warpbreaks`, que proporciona el número de roturas de urdimbre en un hilo, el tipo de lana y el nivel de tensión.

```
# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

# Conjunto de datos de la muestra
warpbreaks %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center",
    full_width = FALSE
)
```

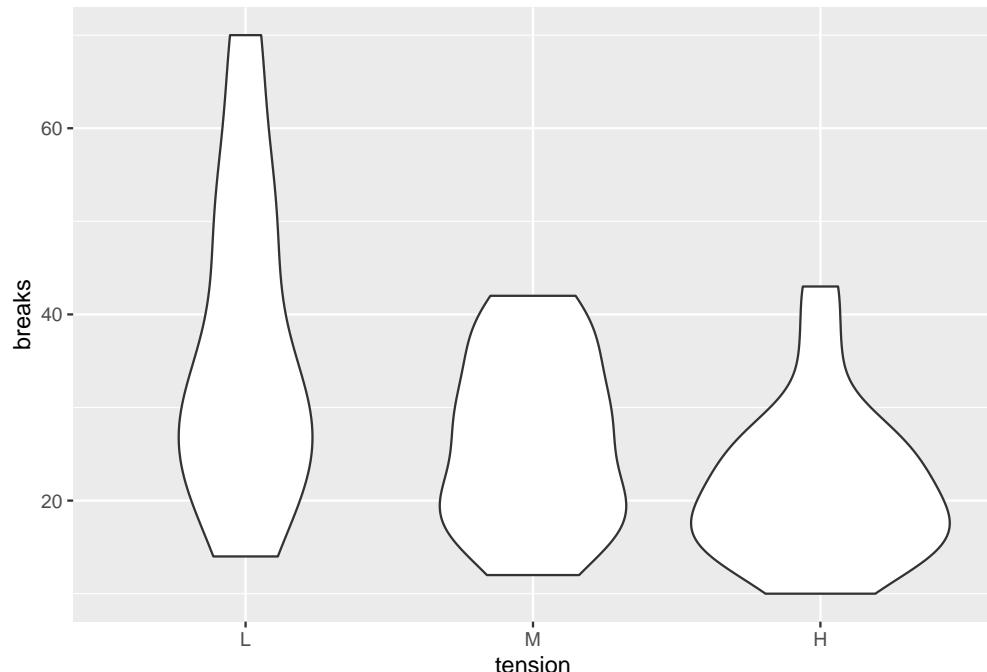
	breaks	wool	tension
	26	A	L
	30	A	L
	54	A	L
	25	A	L
	70	A	L
	52	A	L
	51	A	L
	26	A	L
	67	A	L
	18	A	M

9.10.2 GRÁFICO DE VIOLÍN CON `geom_violin`

Un diagrama de violín por grupo se puede crear en `ggplot2` pasando la variable numérica (`breaks`) y la categórica (`tension`) a la función `aes()` y usando la función `geom_violin()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin()
```

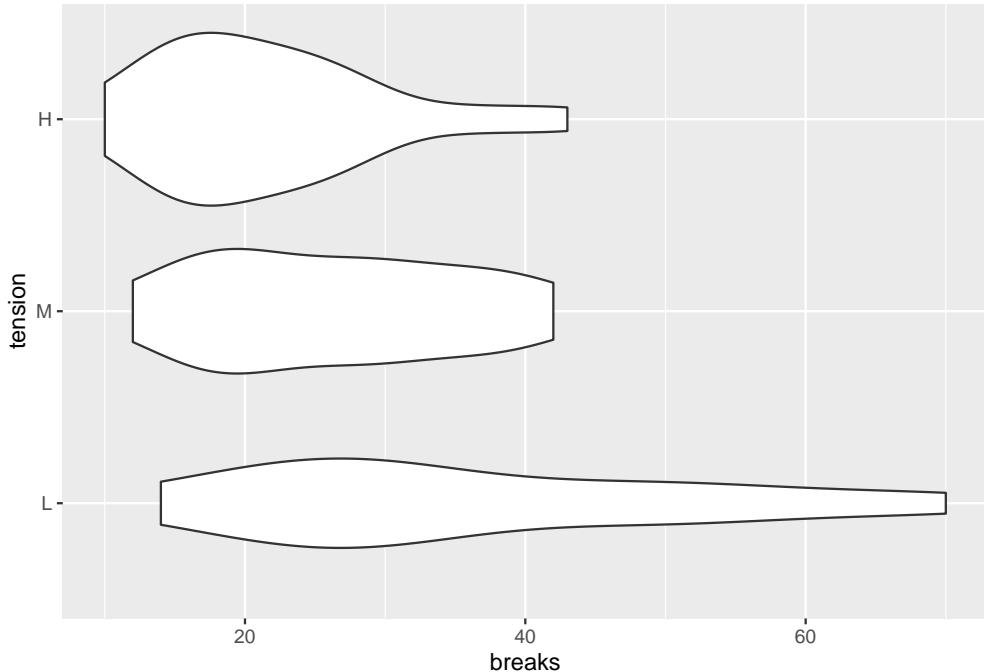


9.10.2.1 DIAGRAMA DE VIOLÍN HORIZONTAL

Si se desea que el gráfico de violín sea horizontal en lugar de vertical se puede pasar la variable categórica (`tension`) al eje y o usar la función `coord_flip()`, como en el siguiente ejemplo.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin() +
  coord_flip()
```

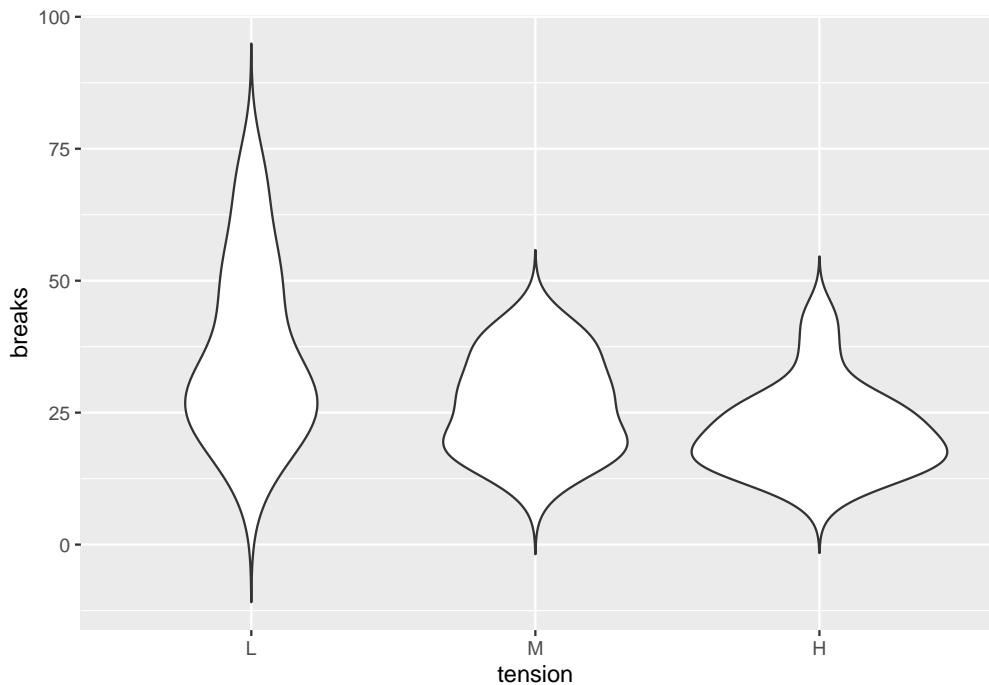


9.10.2.2 EVITAR EL CORTE DE LAS COLAS DE LOS GRÁFICOS DE VIOLÍN

Por defecto, las colas de los violines se cortan con base al rango de los datos. Si se quiere evitar esto solamente se establece el argumento `trim = FALSE` dentro de la función `'geom_violin()'`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin(trim = FALSE)
```

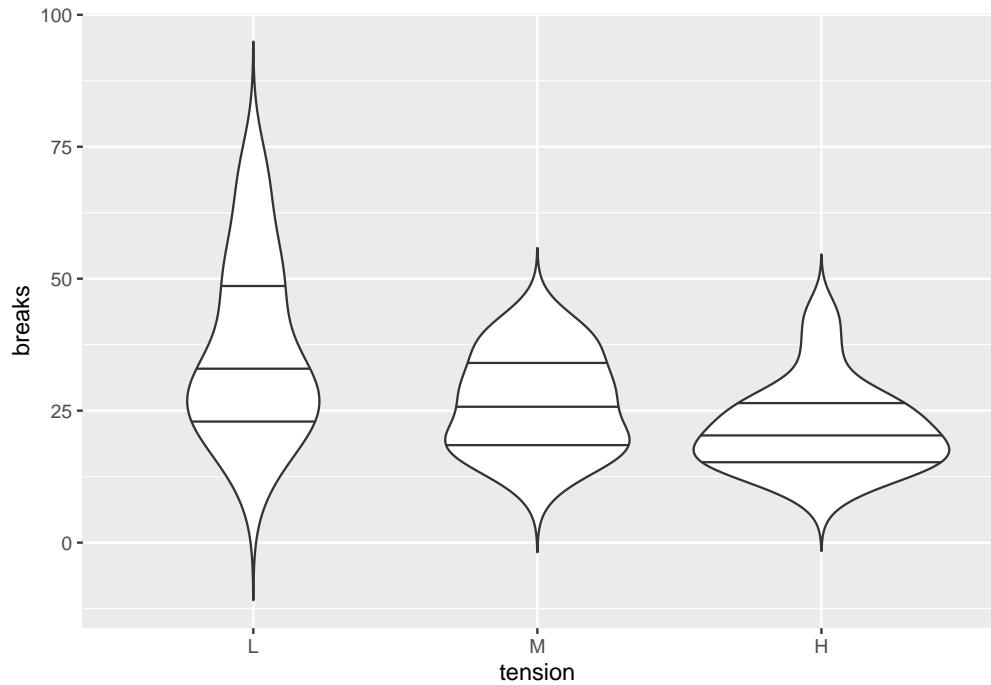


9.10.2.3 AÑADIR CUARTILES

Se puede agregar los cuartiles que se quieran, pasando un vector de cuartiles al argumento `draw_quantiles`, como en el siguiente ejemplo.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin(trim = FALSE,
              draw_quantiles = c(0.25, 0.5, 0.75))
```

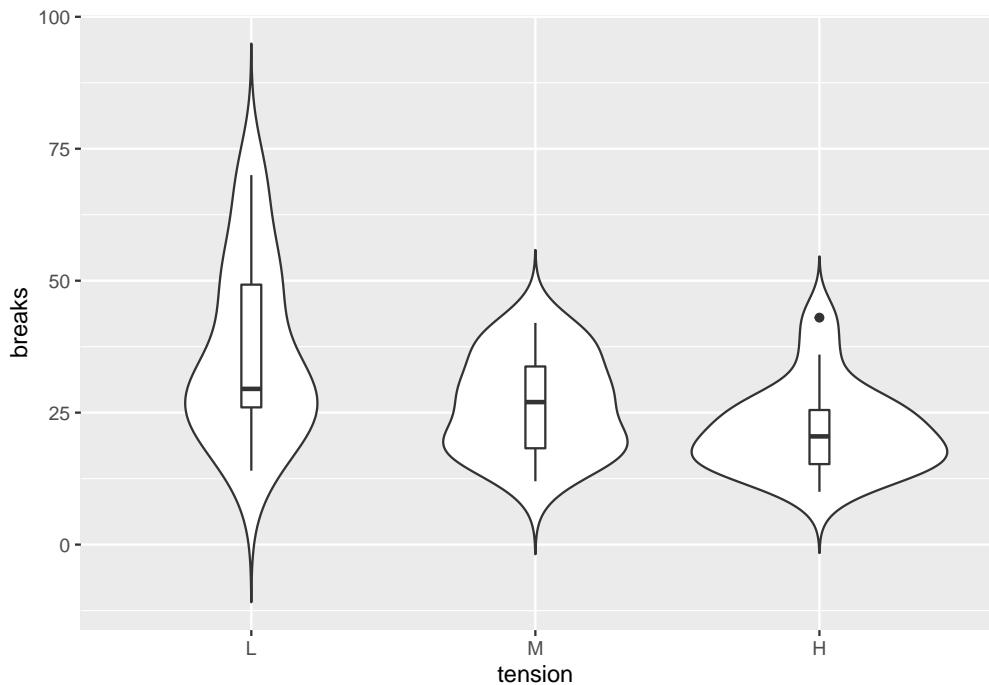


9.10.2.4 AÑADIR BOX PLOT AL DIAGRAMA DE VIOLÍN

Se puede superponer un **box plot** a los diagramas para mostrar la mediana o los datos atípicos. Hay que recordar que se debe de crear un box plot estrecho.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07)
```

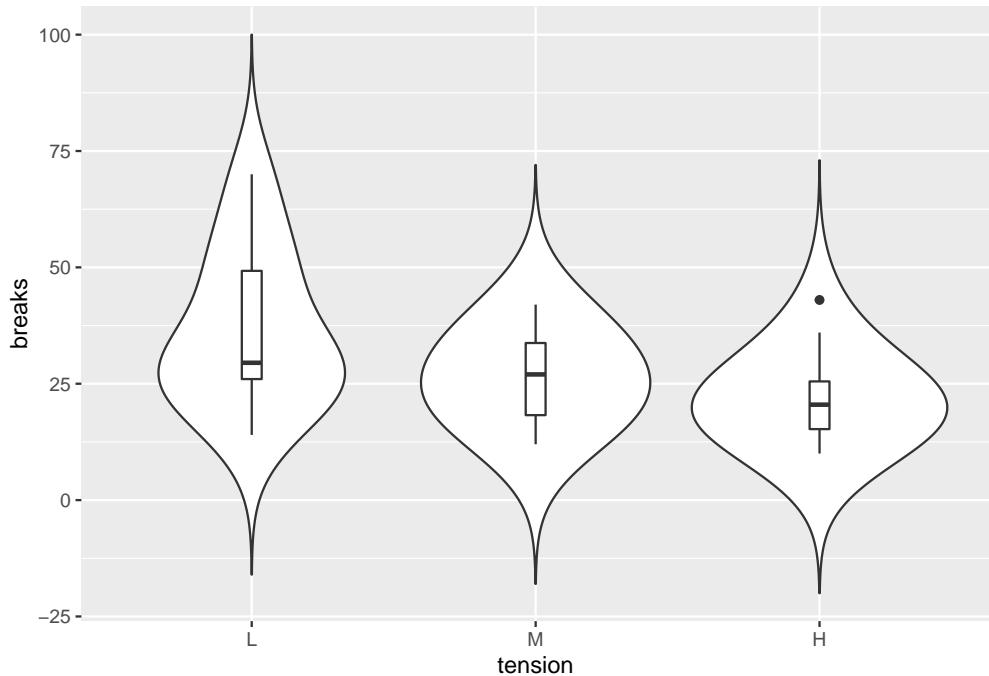


9.10.2.5 SELECCIÓN DE VENTANA

Los diagramas de violín son estimaciones de densidad tipo núcleo de los datos, por lo que para su cálculo se necesita una ventana. La ventana por defecto se puede cambiar con el argumento `bw`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks)) +
  geom_violin(trim = FALSE, bw = 10) +
  geom_boxplot(width = 0.07)
```



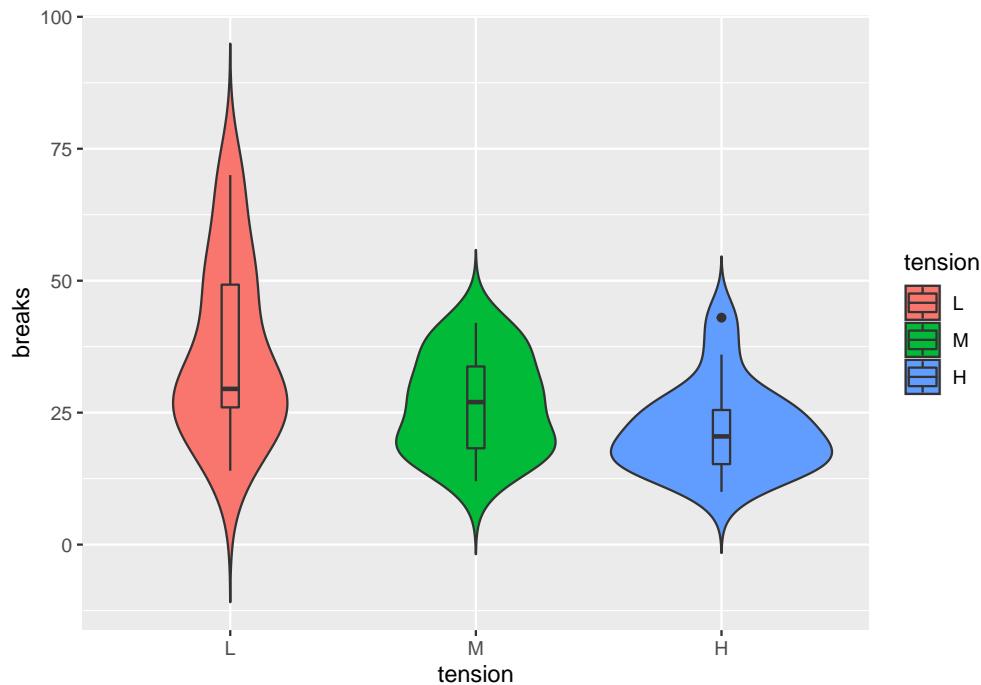
9.10.3 COLOR DE FONDO Y DEL BORDE

9.10.3.1 COLOR DE FONDO PARA CADA GRUPO

Si se quiere colorear los violines por grupo, se pasa la variable categórica al argumento `fill` de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07)
```



9.10.3.2 COLOR DE FONDO PARA CADA SUBGRUPO

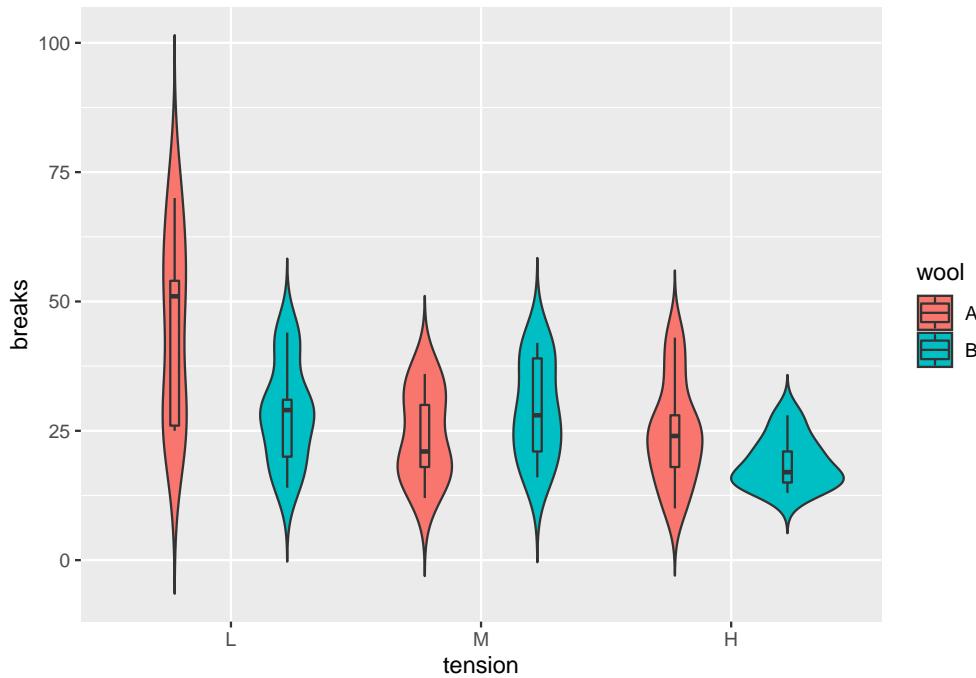
Si se tiene otra variable categórica se puede crear un subgrupo y colorear las áreas con base a estos subgrupos.

9.10.3.2.1 EJEMPLO 1

En este caso lo que se hace es dividir el color dentro del mismo diagrama la categoría y subcategoría.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = wool)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07, position = position_dodge(width = 0.9))
```

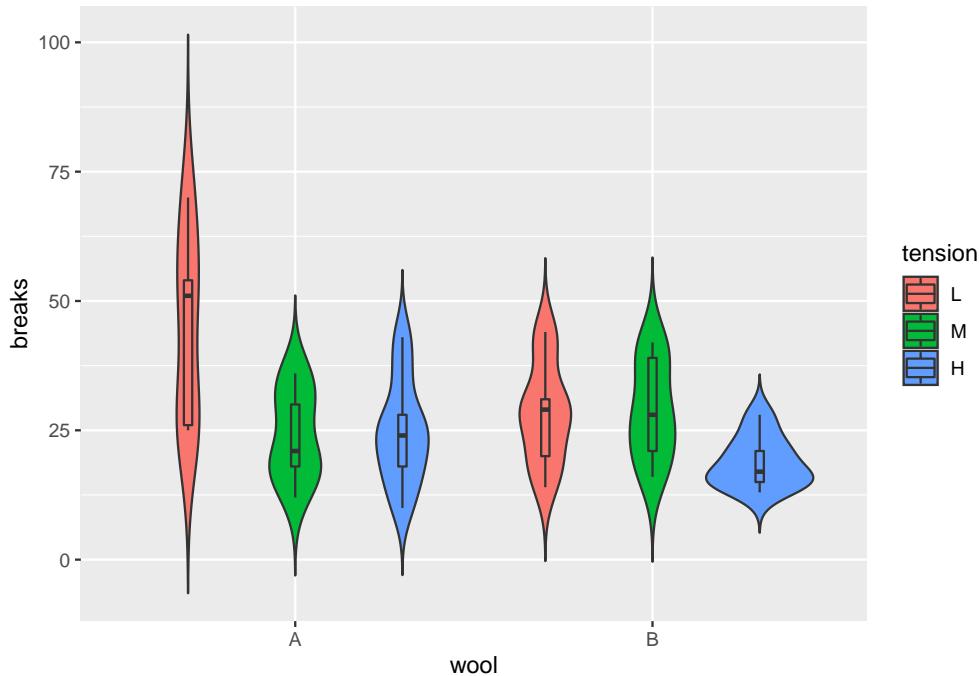


9.10.3.2.2 EJEMPLO 2

En este caso, se trata de manera inversa. La subcategoría se coloca como categoría, y la categoría se utiliza como subcategoría.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = wool, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07, position = position_dodge(width = 0.9))
```

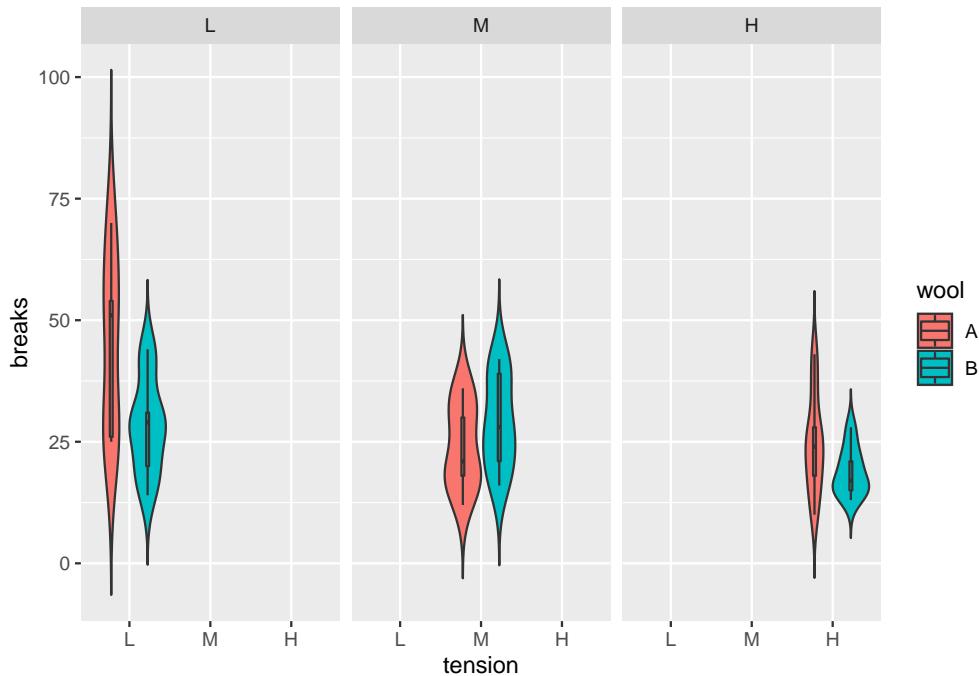


9.10.3.2.3 EJEMPLO 3

Para este caso, lo que se pretende es utilizar el ejemplo 1, pero dividiendo el gráfico por sus subcategorías.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = wool)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07,
               position = position_dodge(width = 0.9)) +
  facet_grid(~tension)
```

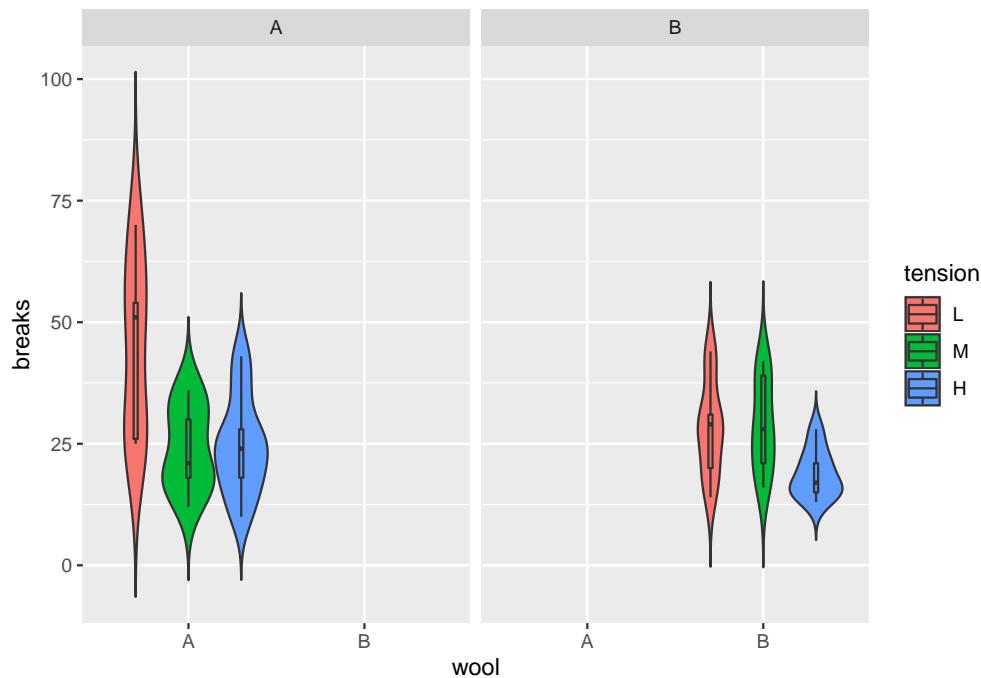


EJEMPLO 4

Para este último caso, lo que se usa es el ejemplo 2, sólo que el diagrama se divide por la categoría.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = wool, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07,
               position = position_dodge(width = 0.9)) +
  facet_grid(~wool)
```

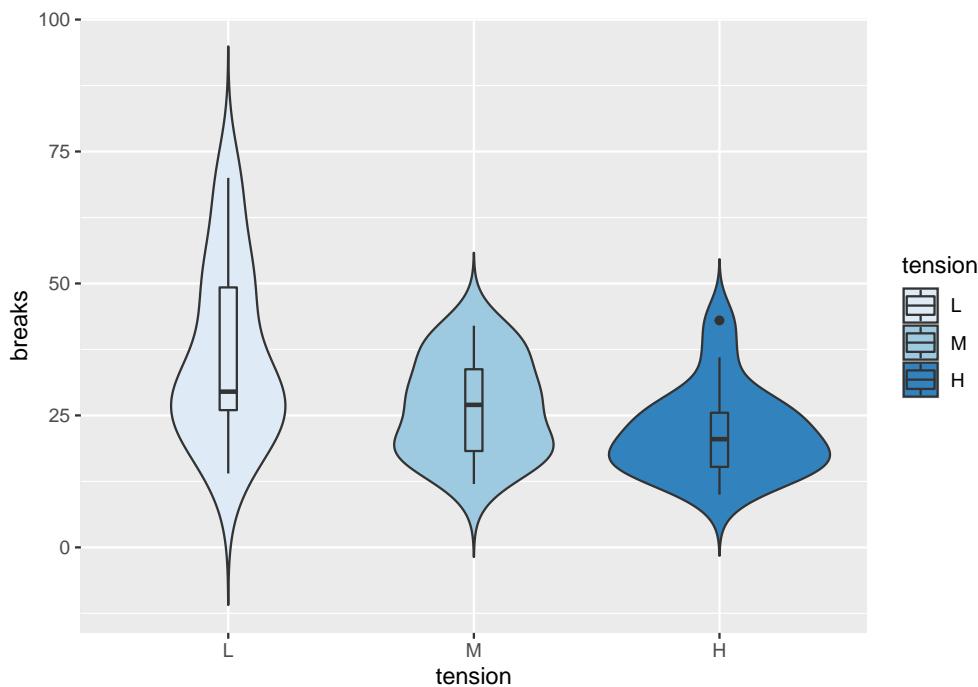


9.10.3.3 ESCALA DE COLOR

Los colores por defecto se pueden cambiar. Por ejemplo, se puede usar la paleta de colores brewer de la siguiente manera:

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07) +
  scale_fill_brewer()
```

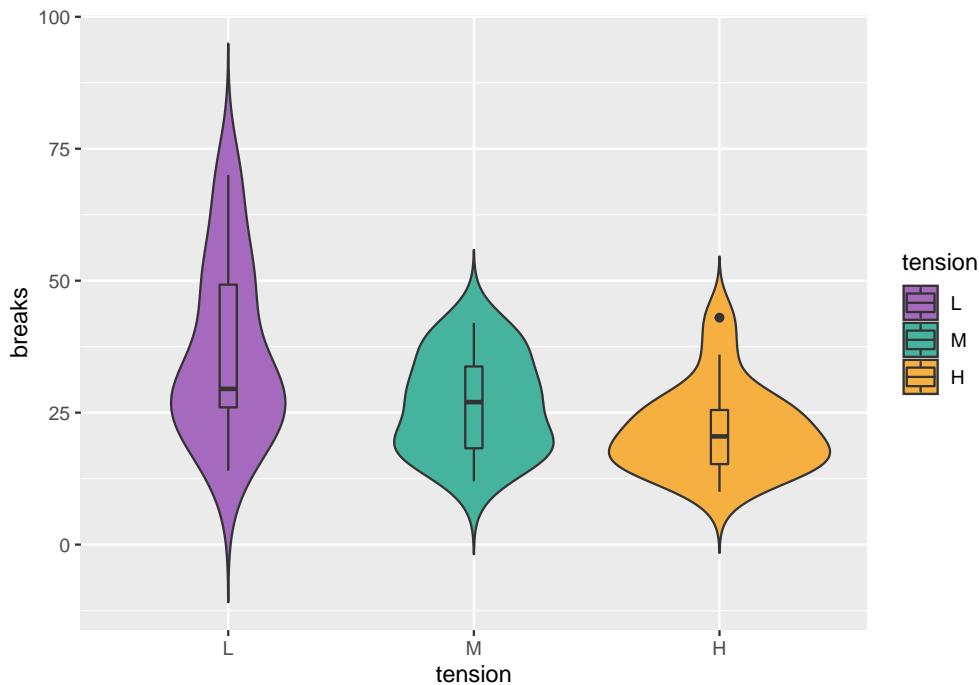


9.10.3.4 COLORES PERSONALIZADOS

Si se desea usar una paleta propia de colores, se puede usar la función `scale_fill_manual()` y pasar los colores que se dese en el argumento `values = c()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07) +
  scale_fill_manual(values = c("#A569BD", "#45B39D", "#F5B041"))
```

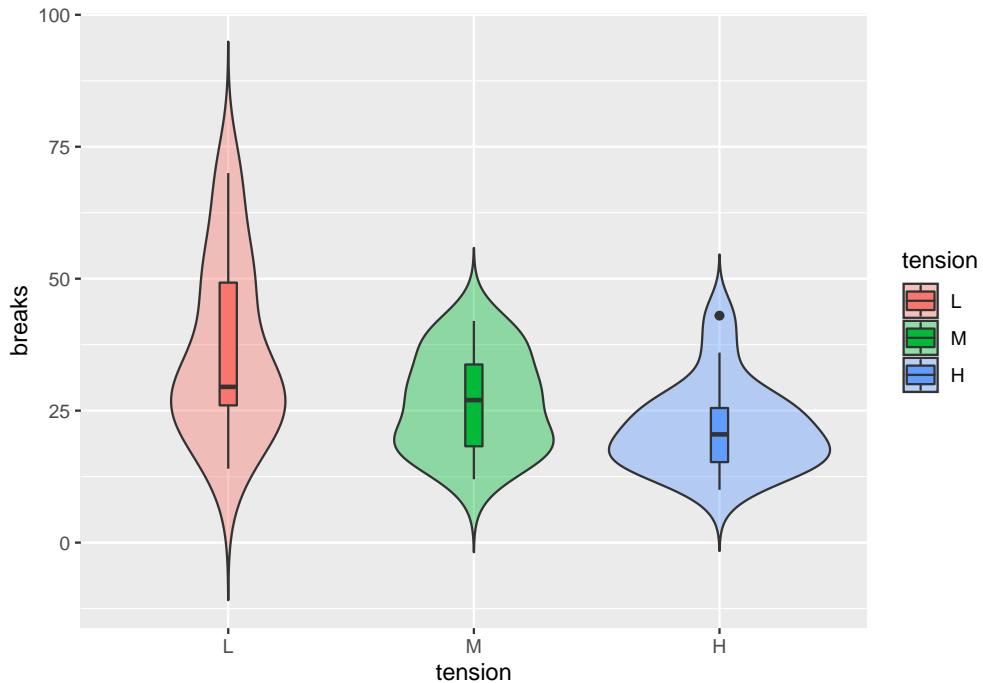


9.10.3.5 TRANSPARENCIA AL VIOLÍN

La transparencia se puede modificar con el argumento `alpha` de la función `geom_violin()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE,
              alpha = 0.4) +
  geom_boxplot(width = 0.07)
```

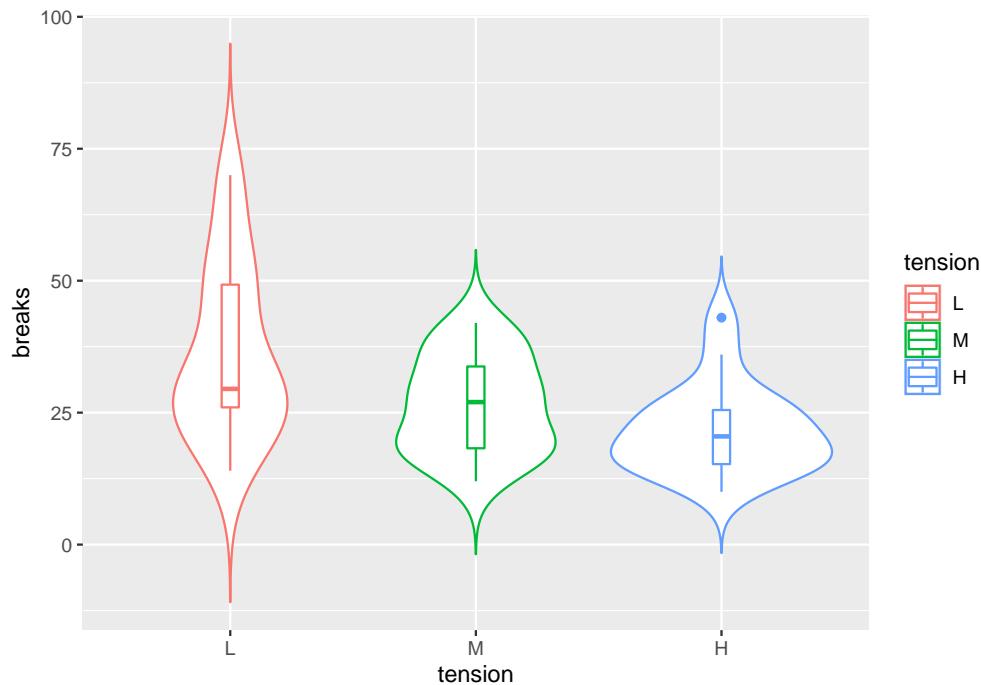


9.10.3.6 COLOR DEL BORDE POR GRUPO

Sin embargo, si se desea establecer un color de borde basado en grupos, se puede pasar la variable categórica al argumento `color` de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, color = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07)
```

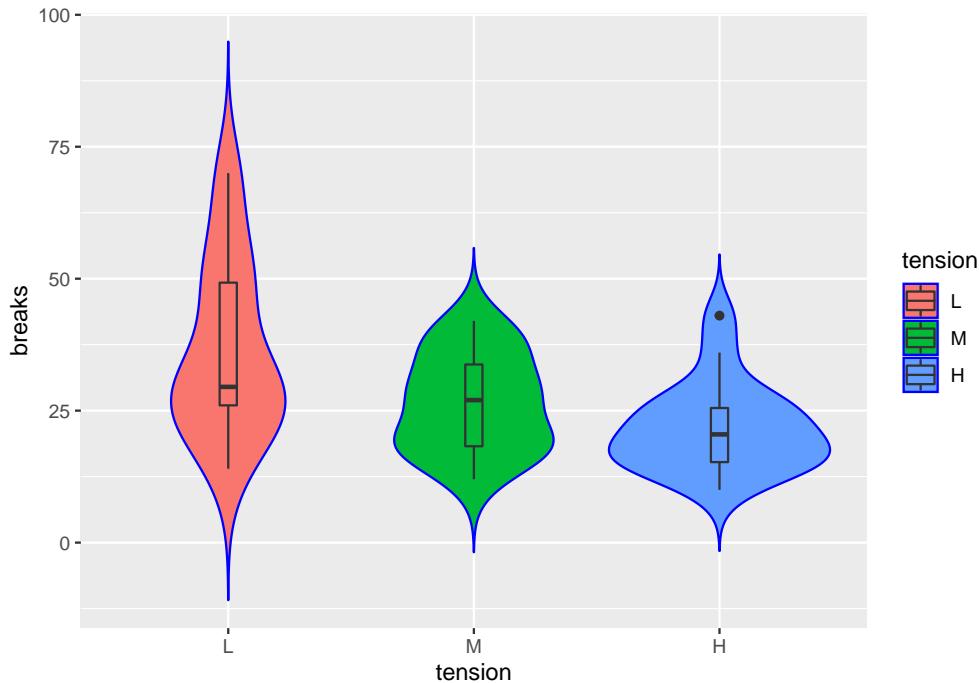


9.10.3.7 COLOR DEL BORDE

El argumento `color` de la función `geom_violin()` se puede usar para cambiar el color de los bordes.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE,
              color = "blue") +
  geom_boxplot(width = 0.07)
```



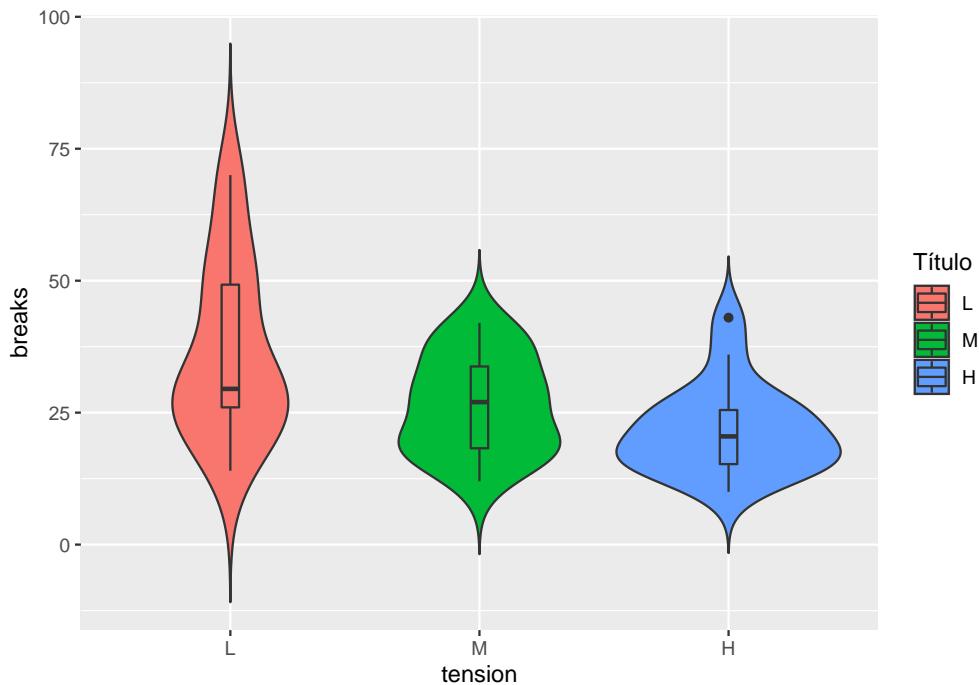
9.10.4 PERSONALIZACIÓN DE LA LEYENDA DEL GRÁFICO

9.10.4.1 TÍTULO DE LA LEYENDA

El título de la leyenda muestra el nombre de la variable categórica. Para cambiar el título por defecto se usa la función `guides()` como se muestra a continuación.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07) +
  guides(fill = guide_legend(title = "Título"))
```

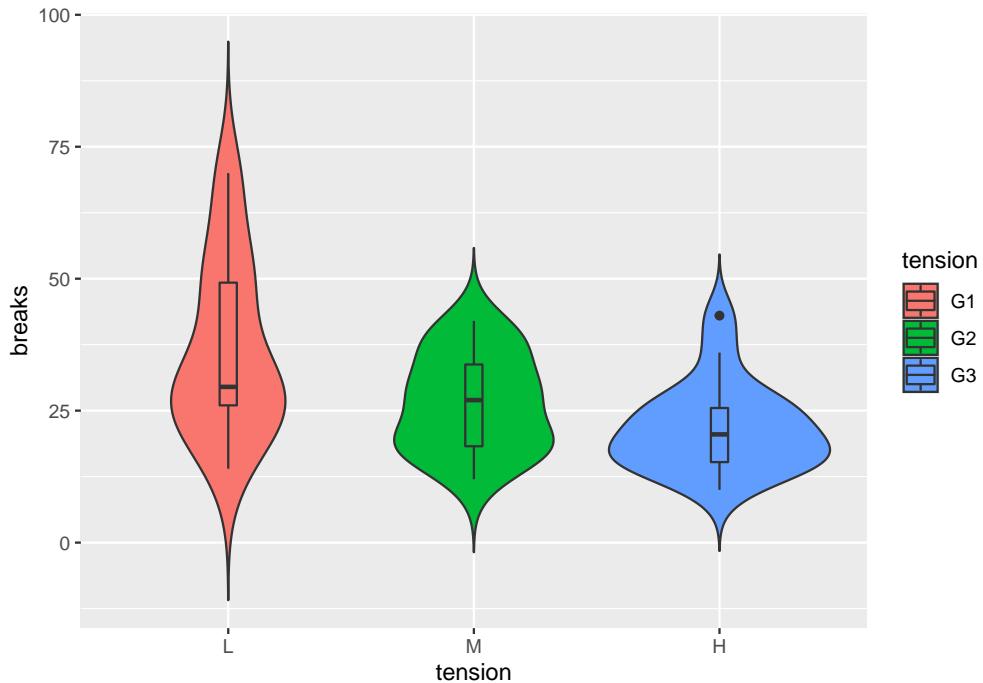


9.10.4.2 ETIQUETAS DE LA LEYENDA

Las etiquetas de la leyenda son los nombres de los grupos. Estas etiquetas se pueden cambiar con `scale_fill_manual()` si se cambian los colores de fondo o con `scale_fill_hue()` para cambiar solo las etiquetas.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07) +
  scale_fill_hue(labels = c("G1", "G2", "G3"))
```

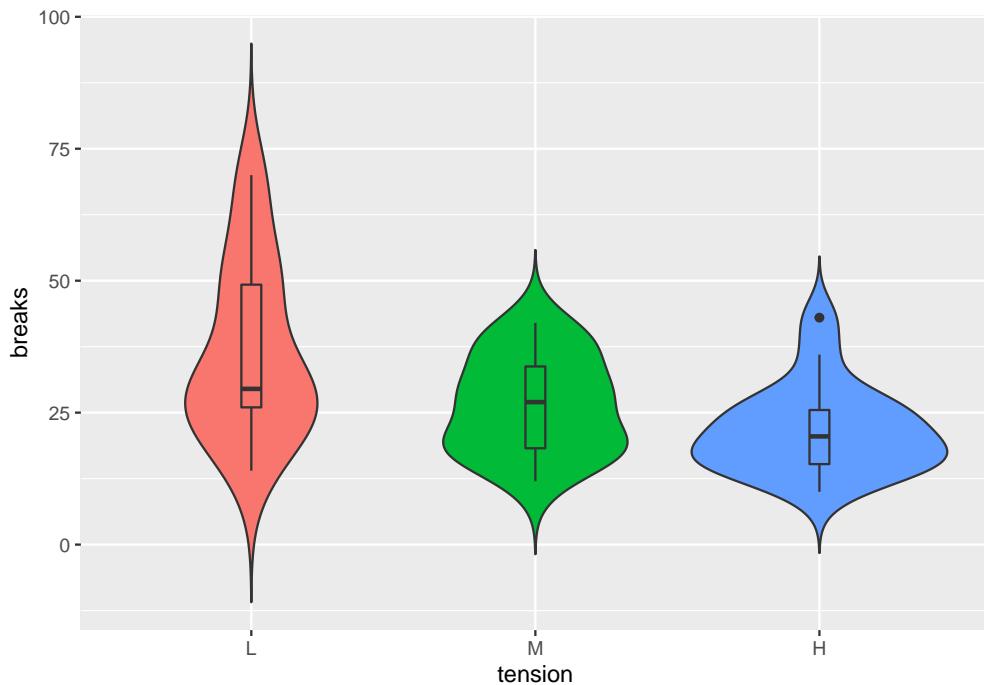


9.10.4.3 ELIMINAR LA LEYENDA DEL GRÁFICO

Por último, si se desea eliminar la leyenda se puede establecer el argumento "none" o añadir el argumento `show.legend = FALSE` a la función `geom_violin()` y `geom_boxplot()` (en el caso de que este último sea añadido dentro de los diagramas de violín).

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(warpbreaks, aes(x = tension, y = breaks, fill = tension)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.07) +
  theme(legend.position = "none")
```



9.11 RIDGELINE EN `ggplot2`

9.11.1 CONJUNTO DE DATOS DE MUESTRA

Para esta parte del manual se empleará un subconjunto del conjunto de datos del paquete `diamonds` del paquete `ggplot2`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos de muestra
df = diamonds[1:100, c("color", "depth")]

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center",
```

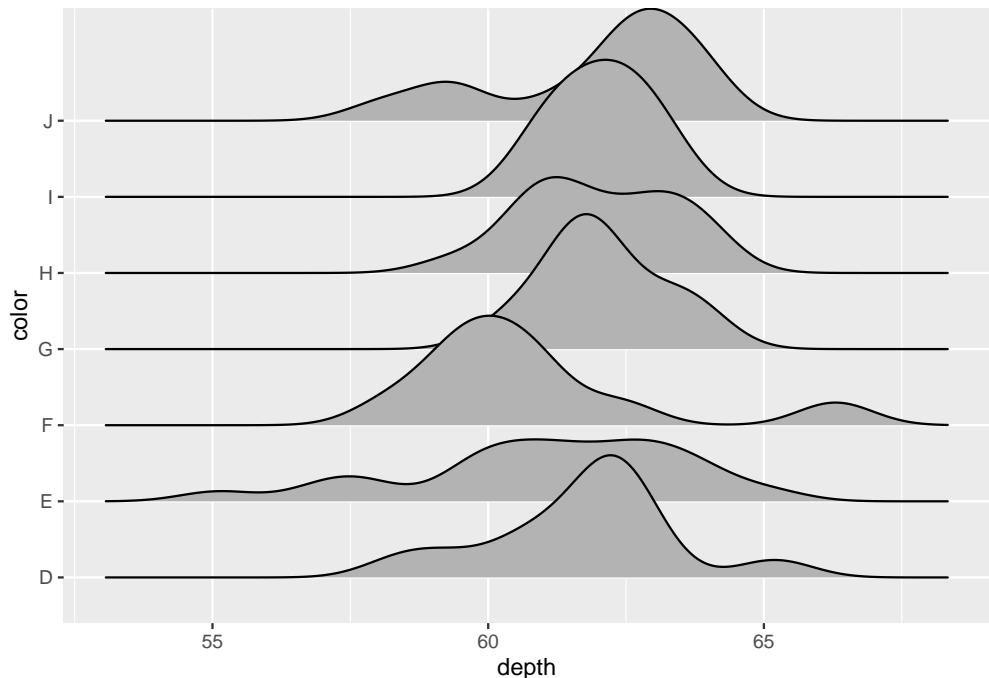
```
    full_width = FALSE  
)
```

color	depth
E	61.5
E	59.8
E	56.9
I	62.4
J	63.3
J	62.8
I	62.3
H	61.9
E	65.1
H	59.4

9.11.1.2 GRÁFICOS RIDGELINE CON LA FUNCIÓN `geom_density_ridges()`

La función `geom_density_ridges()` del paquete `ggridges` permite crear visualizaciones llamadas ridgelines o joy plots. Dada una variable numérica (`depth`) y una categoría (`color`) se calculará una estimación de densidad de los datos y se mostrará para cada grupo.

```
# install.packages("ggridges")  
library(ggridges)  
# install.packages("ggplot2")  
library(ggplot2)  
  
ggplot(df, aes(x = depth, y = color)) +  
  geom_density_ridges()
```

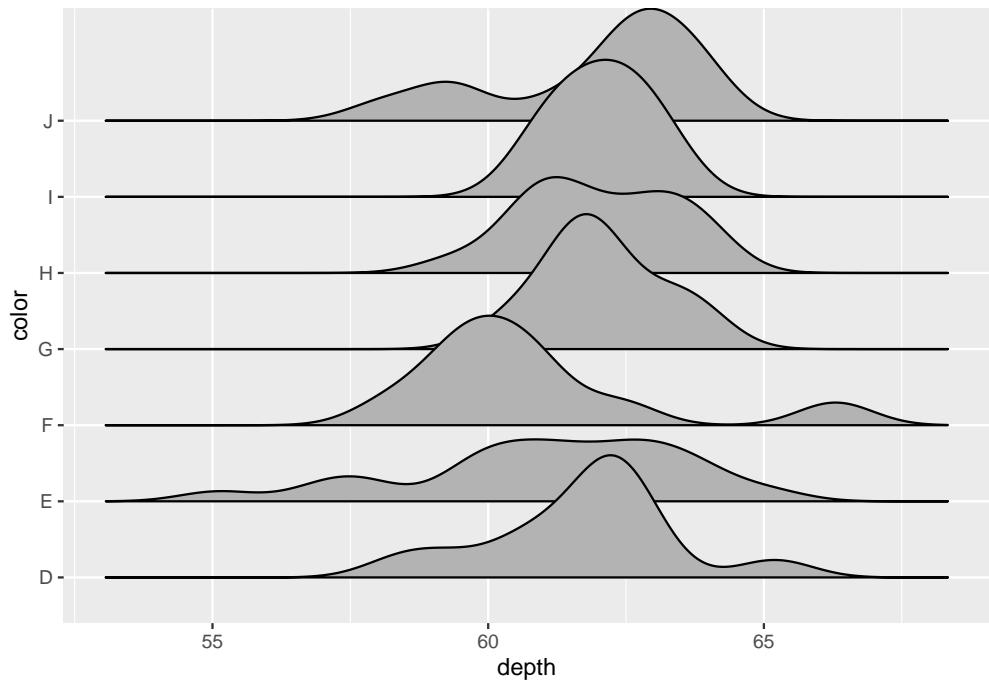


9.11.1.3 LA FUNCIÓN `geom_density_ridges2()`

Hay que tener en cuenta que existe una función equivalente a la anterior, nombrada como `geom_density_ridges2()` que usa polígonos cerrados para mostrar densidades, por lo que también aparecerá una línea en la parte inferior de cada estimación.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges2()
```

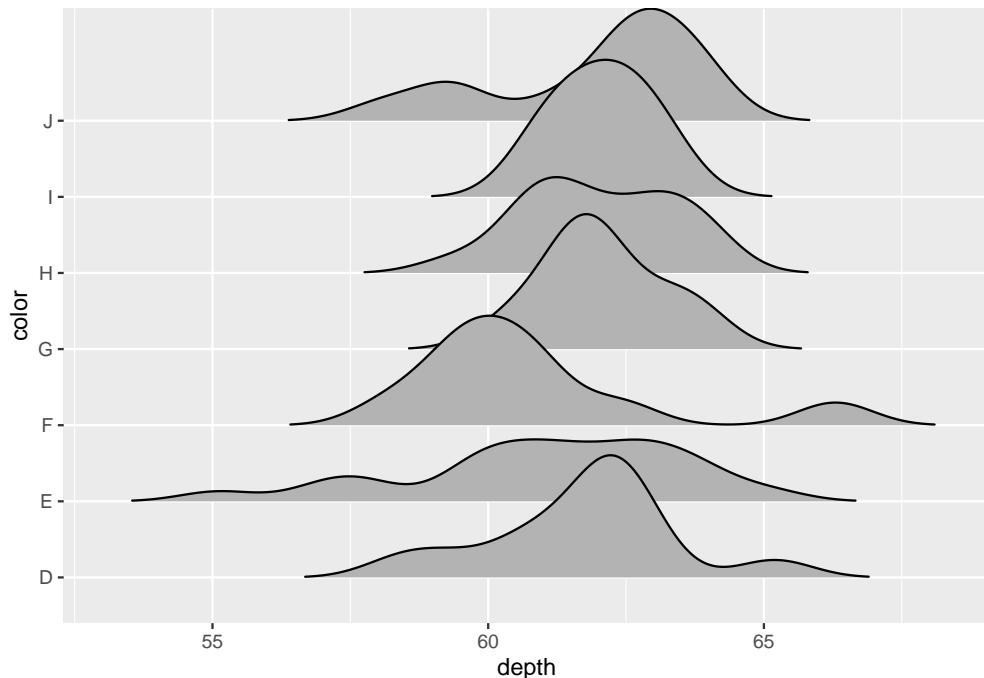


9.11.1.4 RECORTAR LAS COLAS DE LAS DISTRIBUCIONES

El argumento `rel_min_height` dentro de la función `geom_density_ridges()` se puede usar para recortar las colas de las distribuciones. Se tendrá que ajustar al valor dependiendo los datos.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges(rel_min_height = 0.005)
```

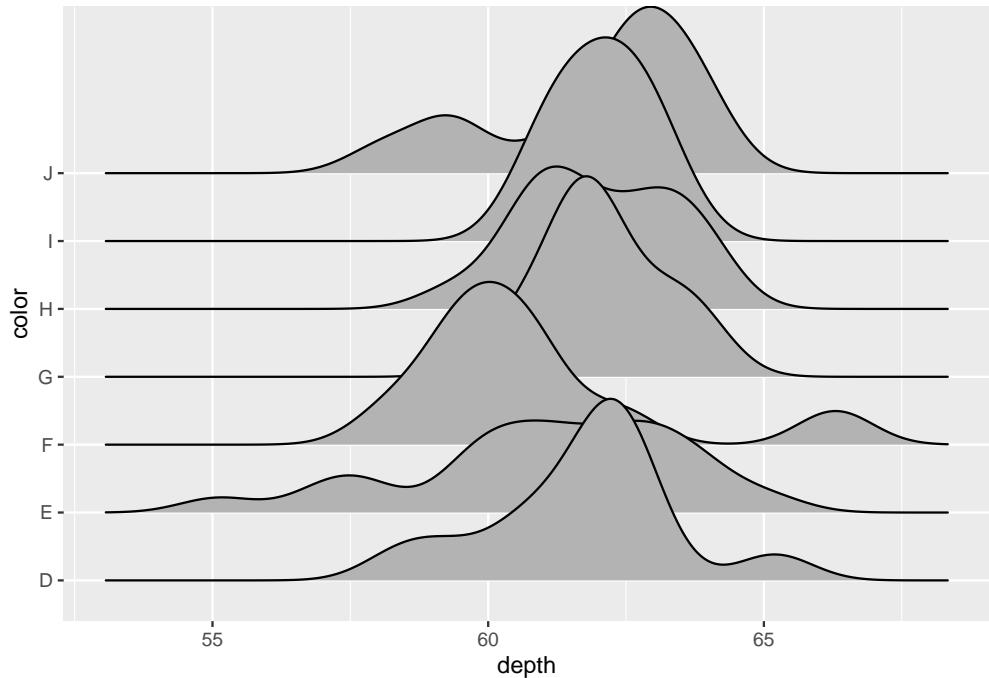


9.11.1.5 ESCALA

Además, el argumento **scale** controla la escala de los ridgelines en referencia al espacio entre ellos.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges(scale = 3)
```

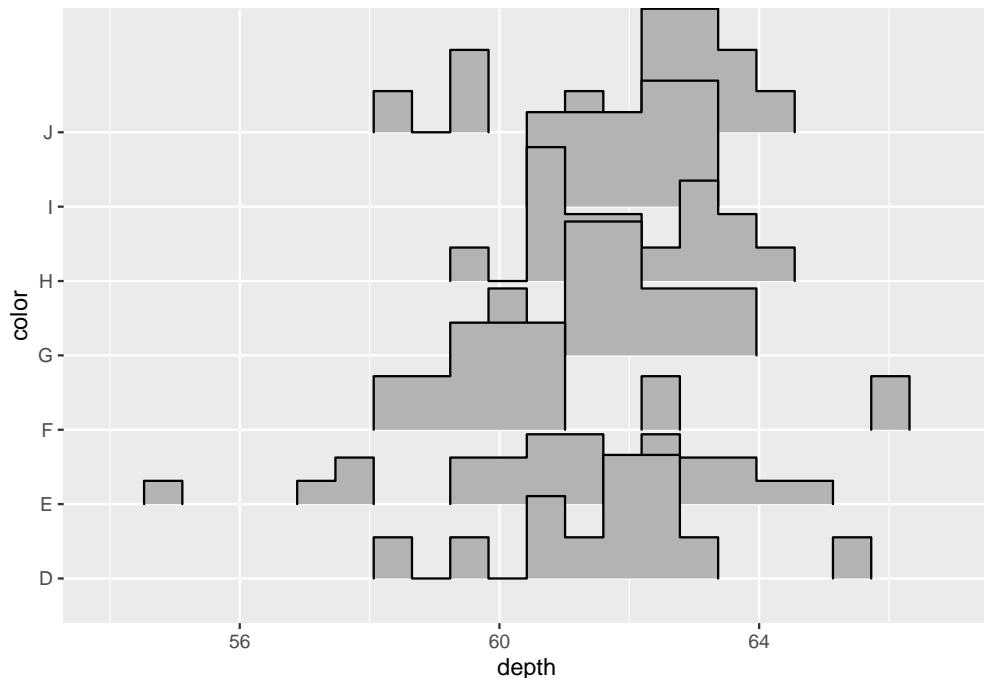


9.11.1.6 ALTERNATIVAS

El argumento **stat** se puede usar para seleccionar la transformación estadística a ser usada.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges(stat = "binline",
                      bins = 20, draw_baseline = FALSE)
```



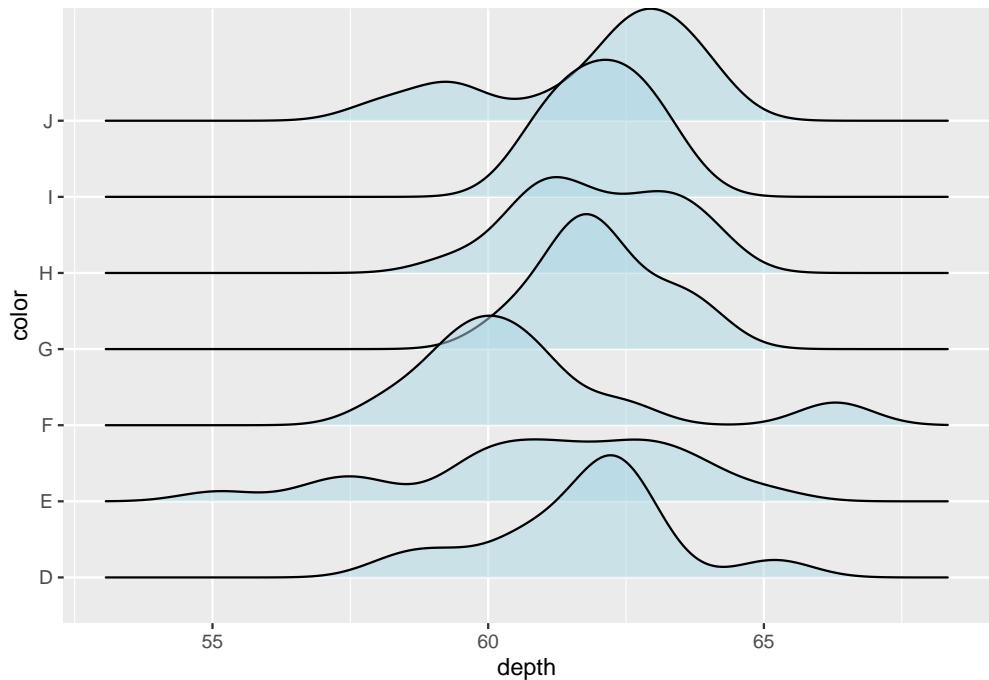
9.11.2 PERSONALIZACIÓN DEL COLOR

9.11.2.1 COLOR DE FONDO Y TRANSPARENCIA

El color gris por defecto de los ridgelines se puede cambiar con el argumento `fill` de la función `geom_density_ridges()`. Hay que tener en cuenta que también se puede especificar un nivel de transparencia con el argumento `alpha`.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges(fill = "lightblue", alpha = 0.5)
```

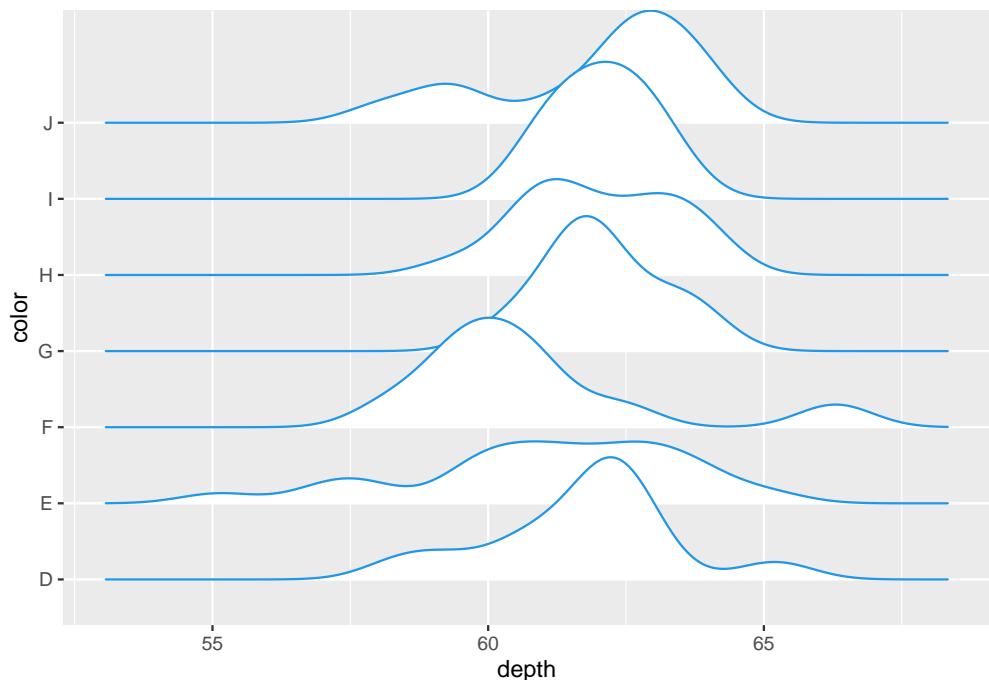


9.11.2.2 COLOR DE LOS BORDES

El argumento `color` de la función controla el color de las líneas. Al igual que en otros diagramas, también se puede cambiar el tipo de línea o el grosor.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  geom_density_ridges(fill = "white",
                      color = 4,
                      linetype = 1, # Tipo de línea
                      lwd = 0.5)    # Grosor
```

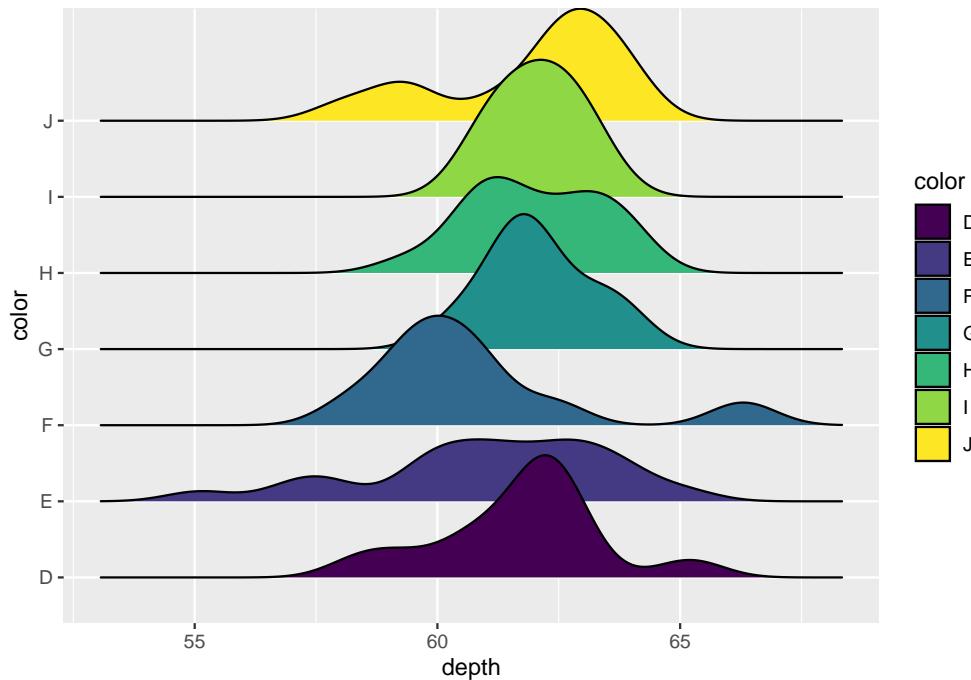


9.11.2.3 COLOR POR GRUPO

También se puede colorear las densidades con base a la variable categórica, pasándola al argumento `fill` de la función `aes()`. La paleta de color se puede cambiar, por ejemplo, con la función `scale_fill_manual()`.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = color)) +
  geom_density_ridges()
```

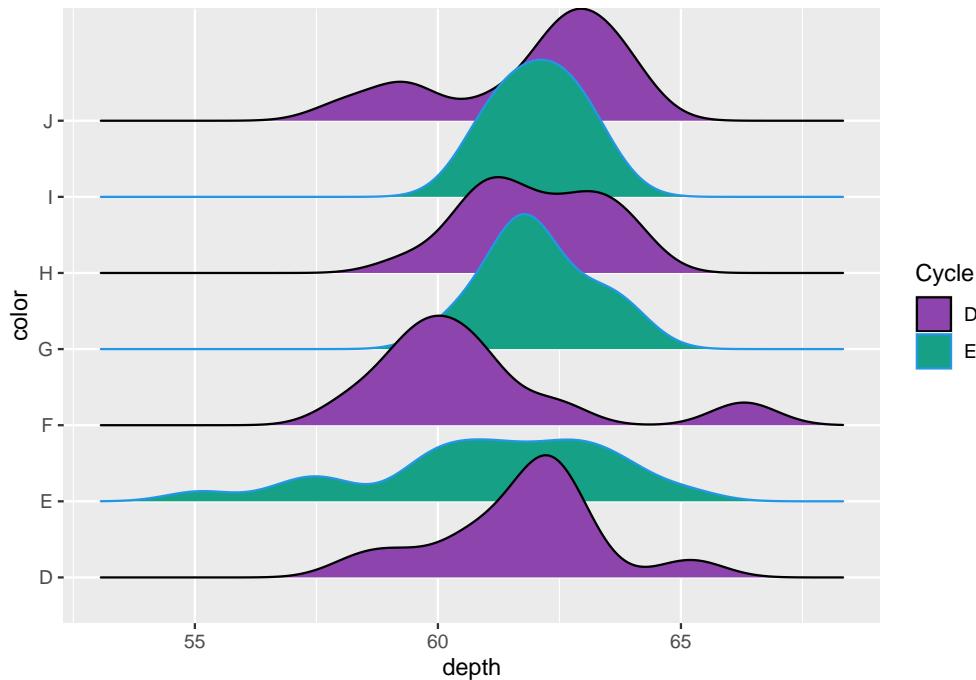


9.11.2.4 ESCALA DE COLOR CÍCLICAS

Las funciones `scale_fill_cyclical()` y `scale_color_cyclical()` se pueden usar para agregar color de fondo y de borde de manera cíclica en las estimaciones de las densidades.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = color, color = color)) +
  geom_density_ridges() +
  scale_fill_cyclical(name = "Cycle", guide = "legend",
                      values = c("#8E44AD", "#16A085")) +
  scale_color_cyclical(name = "Cycle", guide = "legend",
                      values = c(1, 4))
```

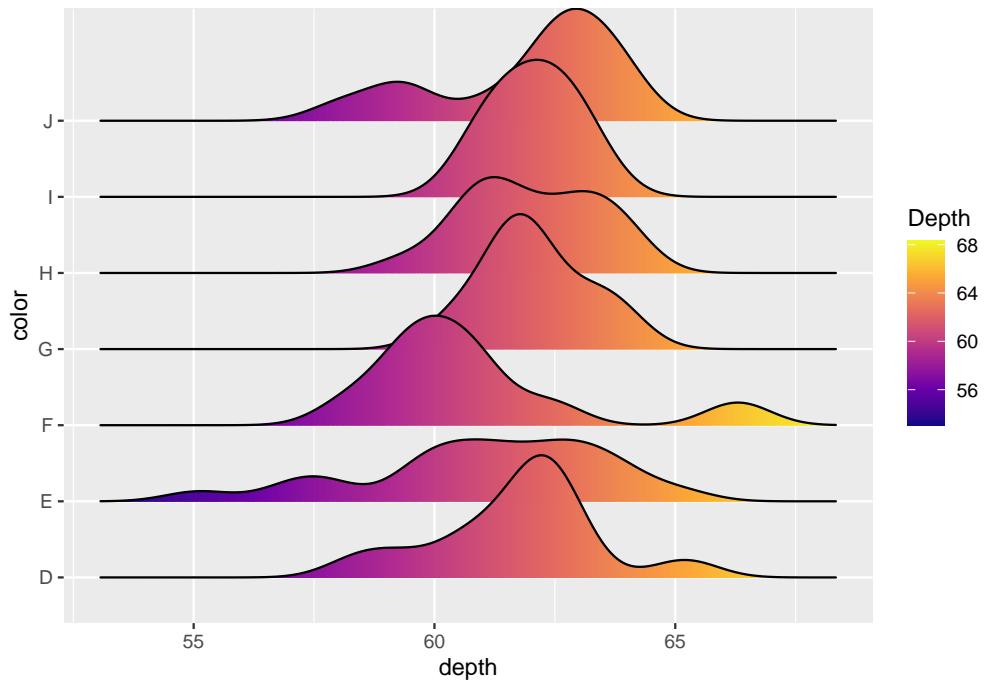


9.11.2.5 DEGRADADO

Se puede pasar la función `stat(x)` o `factor(stat(x))` al argumento `fill` de la función `aes()` y usar la función `geom_density_ridges_gradient()` y una escala de color continua para colorear cada ridgeline con un degradado.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = stat(x))) +
  geom_density_ridges_gradient() +
  scale_fill_viridis_c(name = "Depth", option = "C")
```

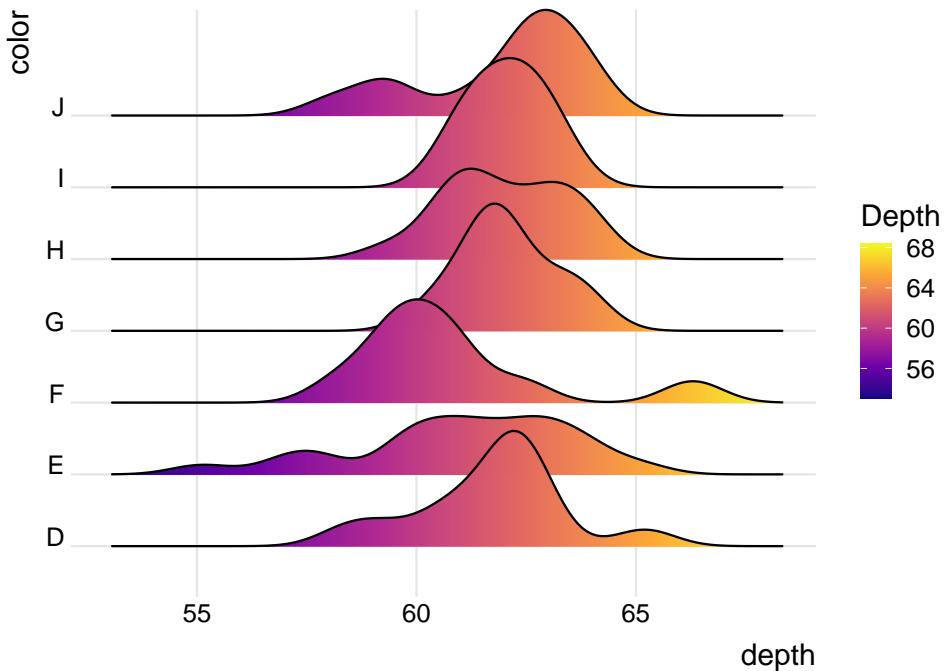


9.11.2.6 CAMBIAR EL COLOR DEL TEMA

Hay que tener en cuenta que se puede cambiar el tema de `ggplot2` para modificar la apariencia del diagrama. El paquete `ggridges` proporciona el tema `theme_ridges()`.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = stat(x))) +
  geom_density_ridges_gradient() +
  scale_fill_viridis_c(name = "Depth", option = "C") +
  coord_cartesian(clip = "off") + # Para evitar el recorte
  theme_ridges()
```

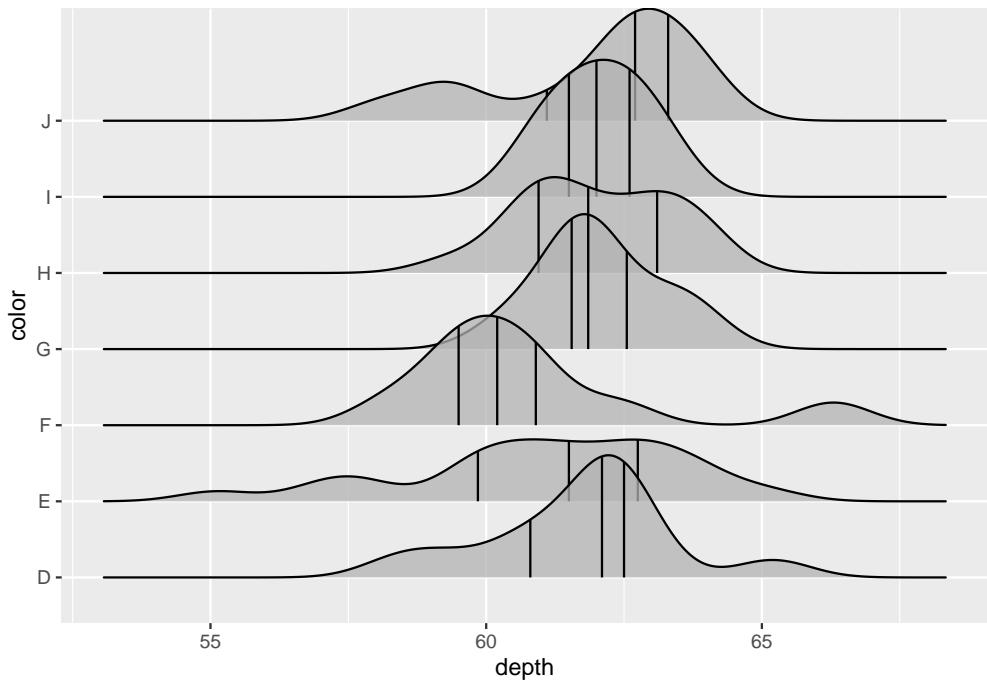


9.11.3 AGREGANDO CUANTILES Y PROBABILIDADES CON LA FUNCIÓN `stat_density_ridges()`

Si se utiliza la función `stat_density_ridges()` y se establece el argumento `quantile_lines` como TRUE, los cuantiles se mostrarán para cada densidad.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color)) +
  stat_density_ridges(quantile_lines = TRUE, alpha = 0.75)
```

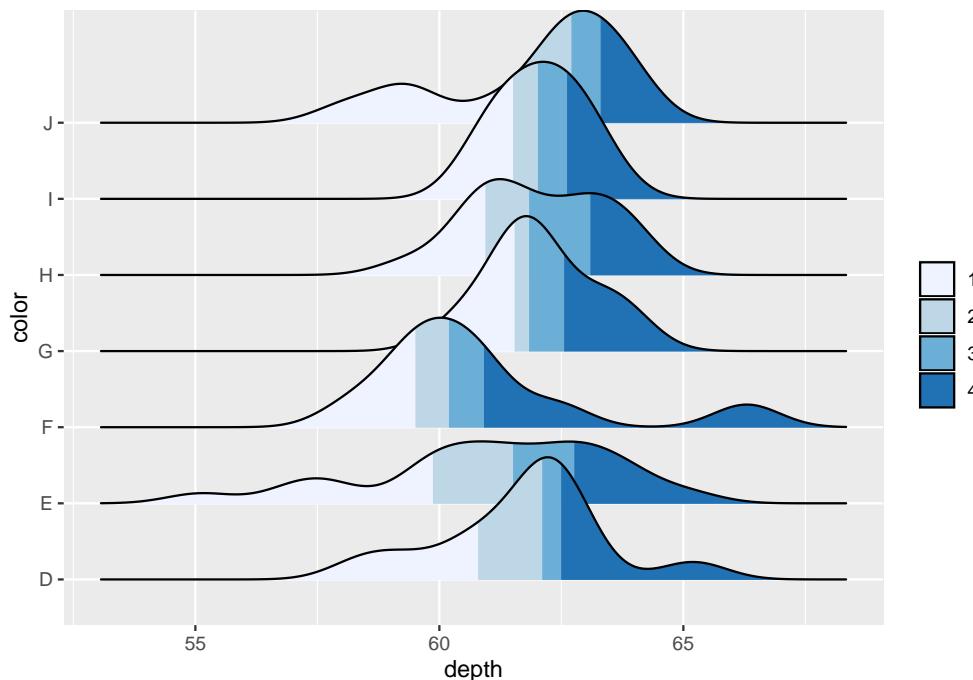


9.11.3.1 COLOR POR CUANTIL

Pasando el argumento `stat(quantile)` al argumento `fill` y, estableciendo `calc_ecdf` como `TRUE`, y el argumento `geom = "density_ridges_gradient"`, cada cuantil se coloreará de un color diferente. Hay que tener en cuenta que si se establece `quantile_lines = TRUE`, se dibujarán las líneas verticales para cada cuantil.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = stat(quantile))) +
  stat_density_ridges(quantile_lines = FALSE,
                      calc_ecdf = TRUE,
                      geom = "density_ridges_gradient") +
  scale_fill_brewer(name = "")
```

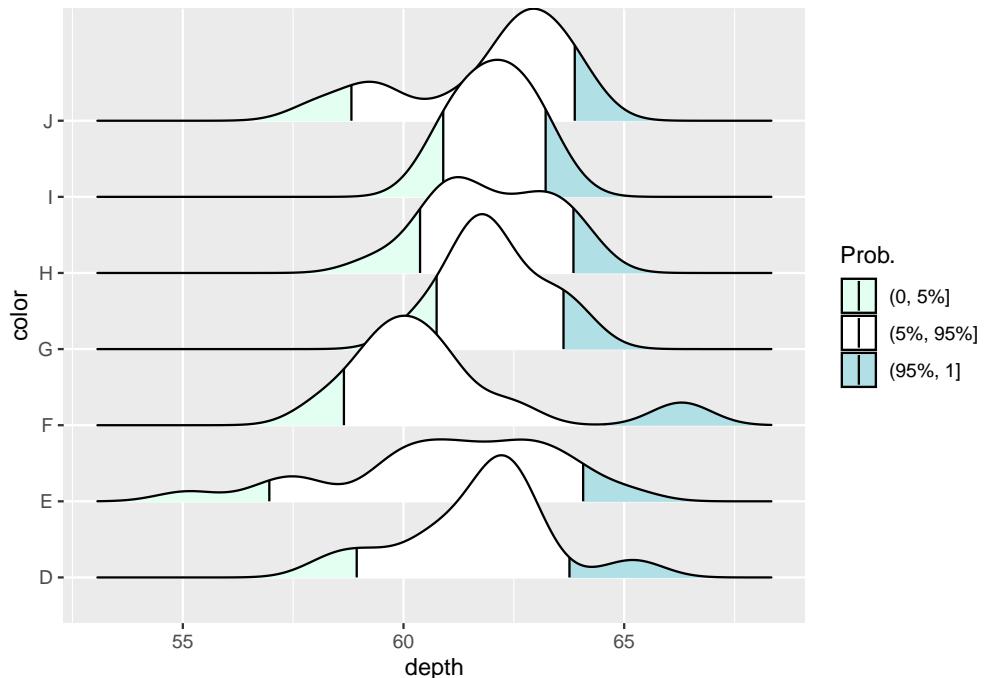


9.11.3.2 RESALTAR LAS COLAS DE LAS DISTRIBUCIONES

El procedimiento descrito antes se puede usar para destacar las colas de las distribuciones.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = depth, y = color, fill = stat(quantile))) +
  stat_density_ridges(quantile_lines = TRUE,
                      calc_ecdf = TRUE,
                      geom = "density_ridges_gradient",
                      quantiles = c(0.05, 0.95)) +
  scale_fill_manual(name = "Prob.", values = c("#E2FFF2", "white", "#BOE0E6"),
                    labels = c("(0, 5%]", "(5%, 95%]", "(95%, 1]"))
```

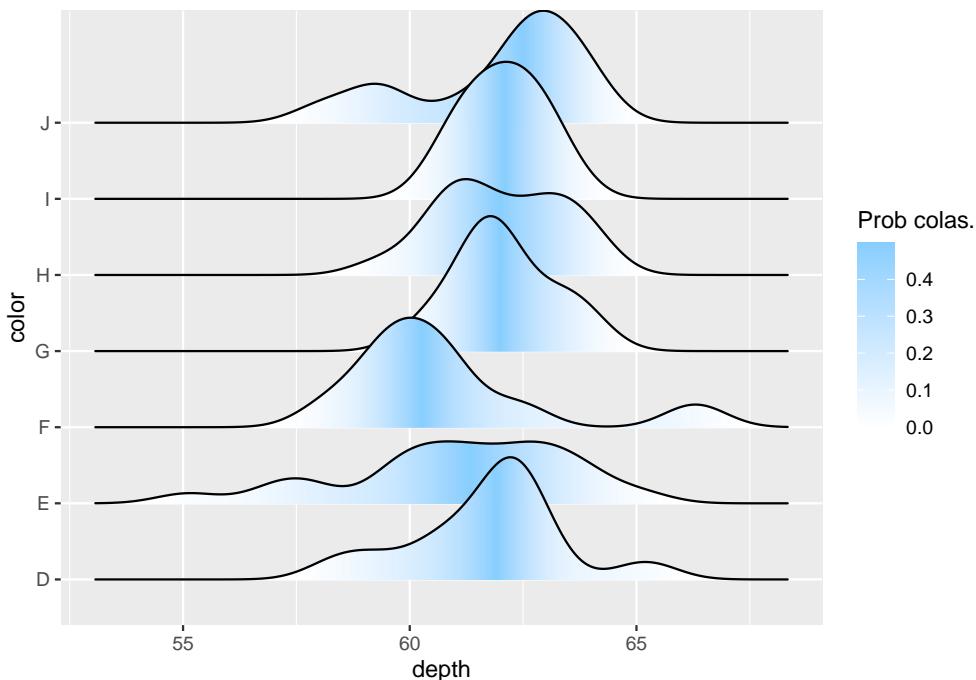


9.11.3.3 PROBABILIDADES DE LAS COLAS COMO DEGRADADO

De manera similar, usando la función `stat(ecdf)`, es posible añadir un degradado a las densidades, mostrando las probabilidades de las colas.

```
# install.packages("ggridges")
library(ggridges)
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(depth, y = color,
               fill = 0.5 - abs(0.5 - stat(ecdf)))) +
  stat_density_ridges(geom = "density_ridges_gradient", calc_ecdf = TRUE) +
  scale_fill_gradient(low = "white", high = "#87CEFF",
                      name = "Prob colas.")
```



9.12 BEESWARM EN ggplot2 CON EL PAQUETE ggbeeswarm

9.12.1 DATOS DE MUESTRA

El siguiente data frame contiene la medida de una variable para tres grupos diferentes. Usando estos datos de ejemplo dentro de este capítulo.

```
# Datos de muestra
set.seed(2020)
y = round(rnorm(500), 1)

df = data.frame(y = y,
                 grupo = sample(c("Grupo 1", "Grupo 2", "Grupo 3"),
                                size = 500,
                                replace = TRUE))

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
```

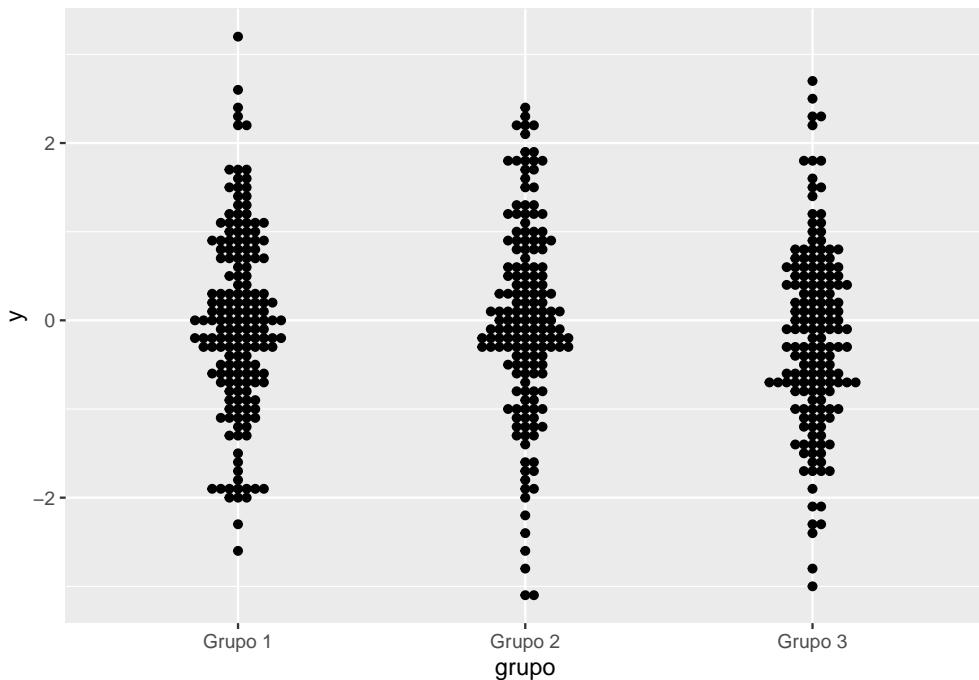
```
kable(booktabs = TRUE, format = "latex") %>%  
kable_styling(  
  latex_options = c("striped", "condensed", "HOLD_position"),  
  position = "center",  
  full_width = FALSE  
)
```

y	grupo
0.4	Grupo 1
0.3	Grupo 1
-1.1	Grupo 3
-1.1	Grupo 3
-2.8	Grupo 3
0.7	Grupo 3
0.9	Grupo 1
-0.2	Grupo 1
1.8	Grupo 3
0.1	Grupo 1

9.12.2 BEESWARMS EN ggplot2 CON geom_beeswarm()

El paquete `ggbeeswarm` contiene una función llamada `geom_beeswarm()`, que puede ser usada para crear un beeswarm en `ggplot2`.

```
# install.packages("ggplot2")  
library(ggplot2)  
# install.packages("ggbeeswarm")  
library(ggbeeswarm)  
  
# Beeswarm básico en ggplot2  
ggplot(df, aes(x = grupo, y = y)) +  
  geom_beeswarm()
```

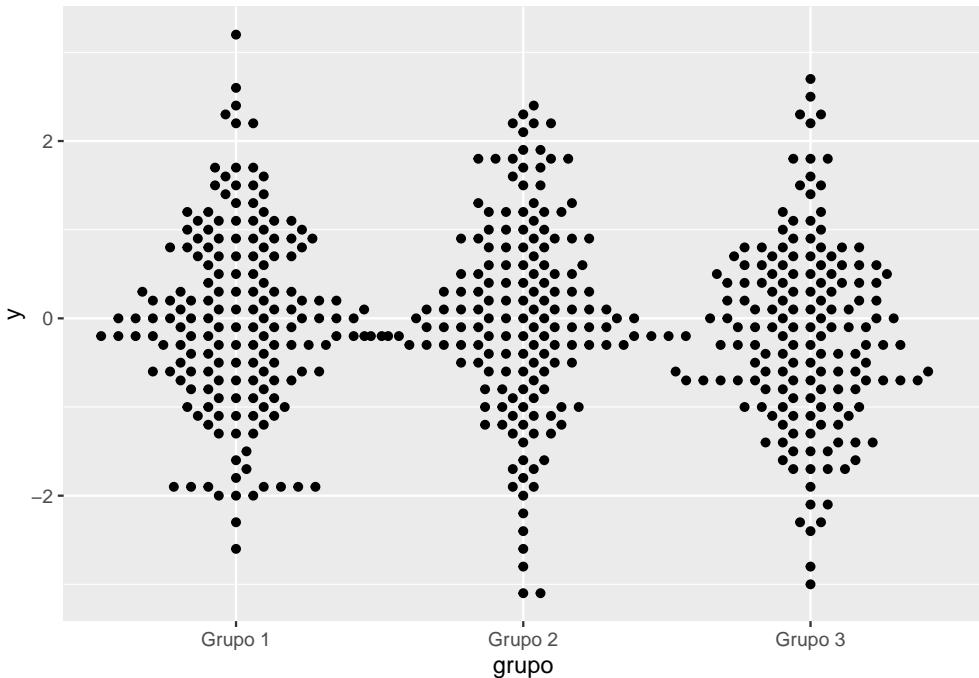


9.12.2.1 TAMAÑO Y ESCALA

Por defecto las observaciones se muestran muy cerca una de otras. Se puede usar el argumento `cex` para incrementar el espacio entre ellas y `size` para incrementar el tamaño de los puntos.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("ggbeeswarm")
library(ggbeeswarm)

# Beeswarm en ggplot2
ggplot(df, aes(x = grupo, y = y)) +
  geom_beeswarm(cex = 2)
```

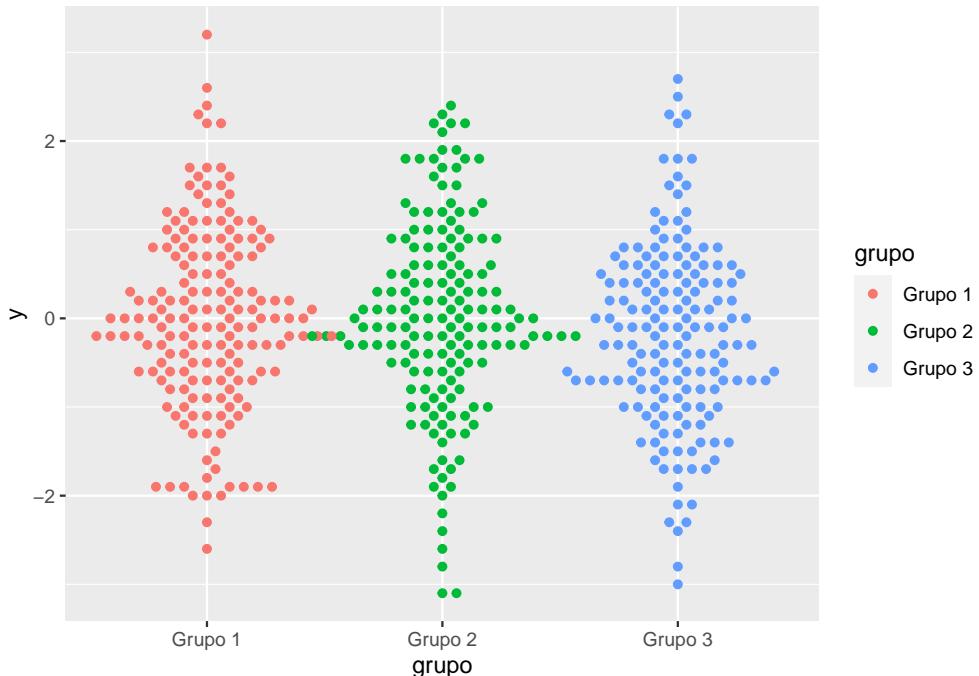


9.12.2.2 COLOR POR GRUPO

Al igual que sucede con otros diagramas con `ggplot2`, si se desea colorear las observaciones por grupo se puede pasar la variable categórica al argumento `color` o `colour` de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("ggbeeswarm")
library(ggbeeswarm)

# Beeswarm en ggplot2
ggplot(df, aes(x = grupo, y = y, color = grupo)) +
  geom_beeswarm(cex = 2)
```

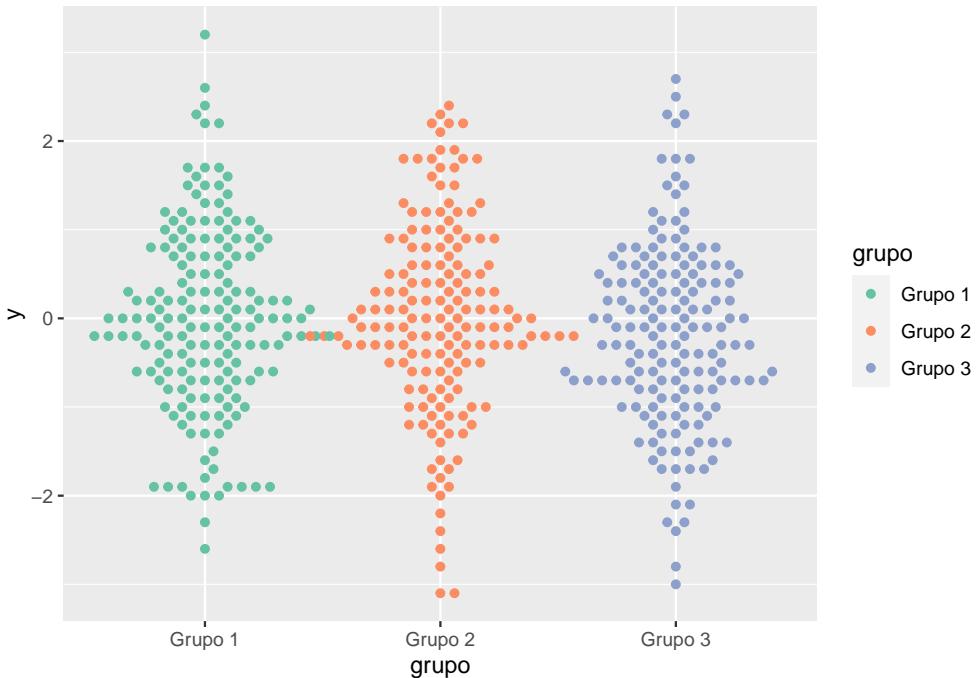


9.12.2.3 CAMBIAR LOS COLORES

Si se colorean las observaciones por grupo se usará una escala de colores por defecto. Se pueden cambiar los colores con la función `scale_color_brewer()`, `scale_color_manual()` o una escala de color equivalente.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("ggbeeswarm")
library(ggbeeswarm)

# Beeswarm en ggplot2
ggplot(df, aes(x = grupo, y = y, color = grupo)) +
  geom_beeswarm(cex = 2) +
  scale_color_brewer(palette = "Set2")
```

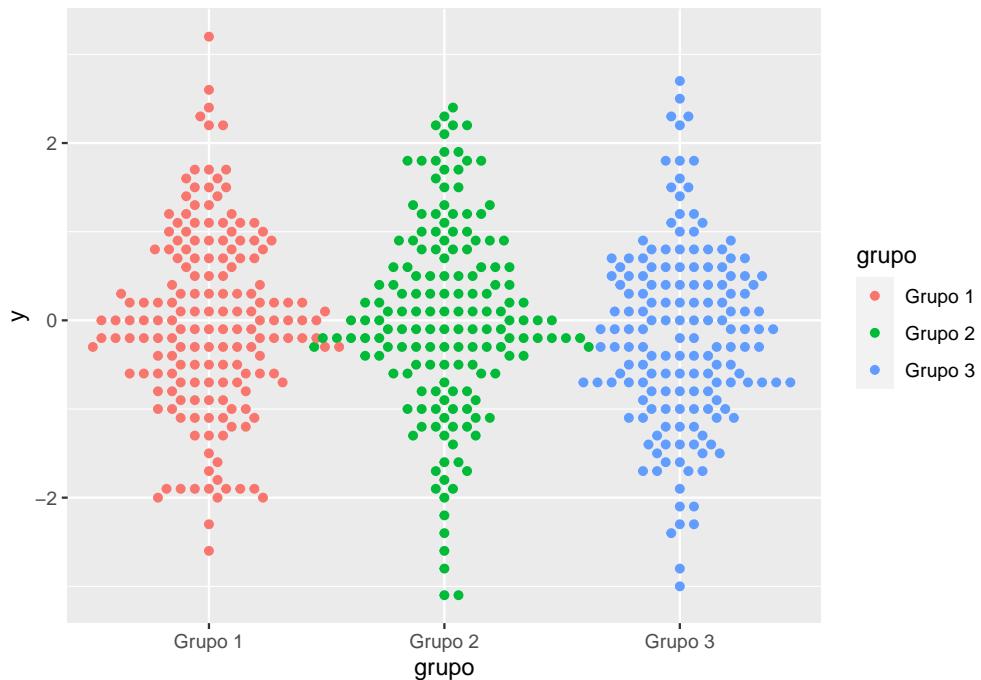


9.12.2.4 MÉTODOS DE PRIORIDAD

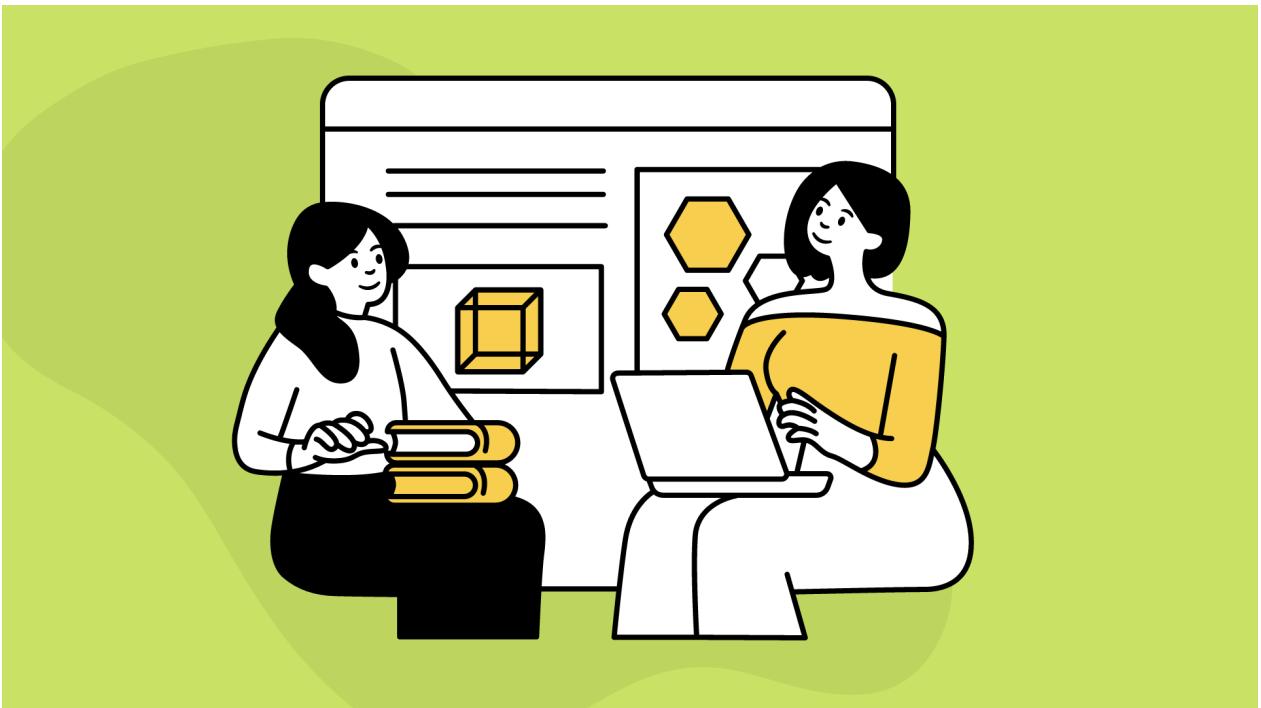
Para que se realicen los cálculos necesarios para crear el beeswarm es necesario elegir un método de prioridad. Por defecto se utiliza el método "ascending", pero se puede cambiar con el argumento `priority`. Las otras posibles opciones son "descending", "density", "random" y "none".

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("ggbeeswarm")
library(ggbeeswarm)

# Beeswarm en ggplot2
ggplot(df, aes(x = grupo, y = y, color = grupo)) +
  geom_beeswarm(cex = 2,
                priority = "density")
```



CAPÍTULO 10: DIAGRAMAS DE CORRELACIÓN CON `ggplot2`



Los diagramas de correlación son útiles para visualizar la relación entre dos o más variables.

10.1 GRÁFICO DE DISPERSIÓN POR GRUPO EN `ggplot2`

10.1.1 DATOS

El siguiente dataframe contiene dos variables numéricas y una variable categórica que representa grupos. Este `data frame` será usado dentro de los siguientes ejemplos dentro de este subcapítulo.

```

set.seed(999)

# Simulación de datos
x = runif(600)
y = 5 * x ^ 2 + rnorm(length(x), sd = 2)
grupo <- ifelse(x < 0.4, "Grupo 1",
                 ifelse(x > 0.8, "Grupo 3", "Grupo 2"))
x = x + runif(length(x), -0.2, 0.2)

# Data frame
df = data.frame(x = x, y = y, grupo = grupo)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center",
    full_width = FALSE
  )

```

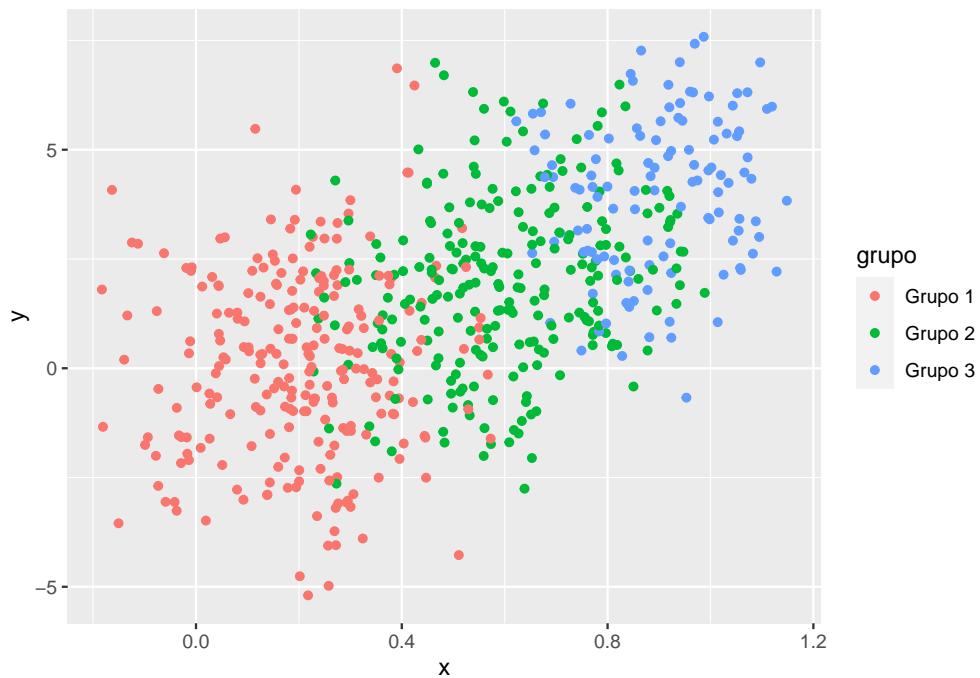
x	y	grupo
0.5172935	3.205086	Grupo 1
0.5424679	1.294659	Grupo 2
-0.0097314	2.226582	Grupo 1
0.6705957	5.857887	Grupo 3
0.6355820	3.234543	Grupo 2
0.2931532	-1.435535	Grupo 1
0.4888600	3.082968	Grupo 2
0.2497989	2.967532	Grupo 1
0.2484183	1.890612	Grupo 1
0.5182422	2.863419	Grupo 2

10.1.2 DIAGRAMA DE DISPERSIÓN POR GRUPO CON `geom_point()`

Para crear un diagrama de dispersión con colores por grupo con el paquete `ggplot2`, sólo se necesita especificar las variables numéricas y pasar la variable categórica al argumento `color` (o `colour`) dentro de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Gráfico de dispersión por grupo
ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point()
```



10.1.2.1 CAMBIAR LOS COLORES

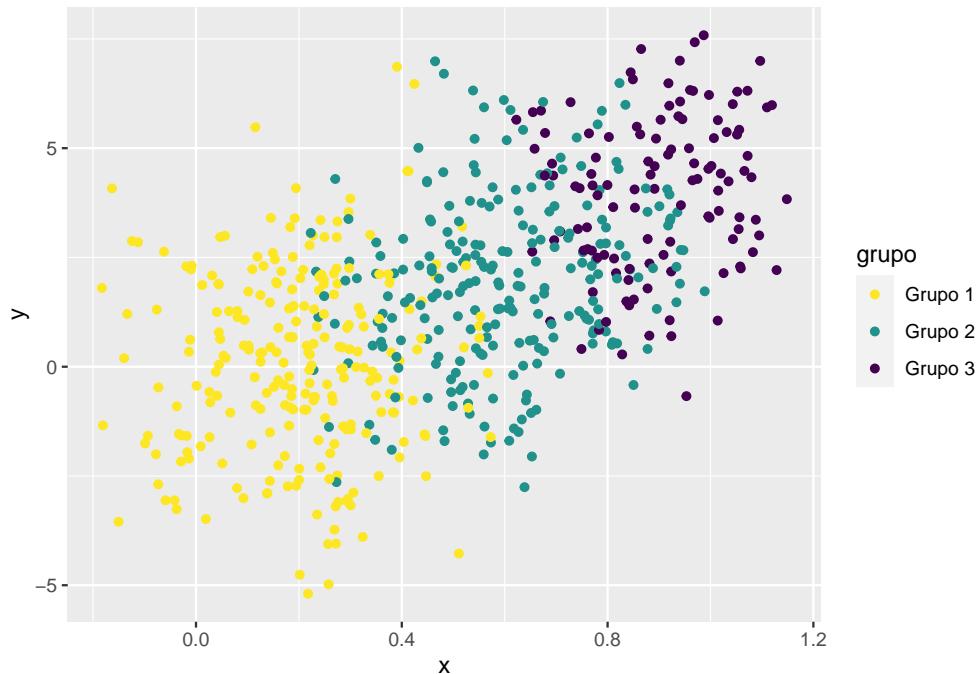
La paleta de colores por defecto se puede personalizar de varias maneras. Se puede usar la función `scale_color_manual()` y pasar un vector de colores al argumento `values` o elegir una paleta de colores predefinida, como `scale_color_viridis()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Vector de colores estilo 'viridis'
```

```
cols = c("#fde725", "#21918c", "#440154")

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point() +
  scale_color_manual(values = cols)
```



10.1.2.2 CAMBIAR LA FORMA Y EL TAMAÑO

La forma por defecto de las observaciones son círculos, pero se puede elegir cualquier símbolo con el argumento `shape`. El tamaño de los símbolos se pueden personalizar haciendo uso del argumento `size`, todo esto dentro de la función `geom_point()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point(shape = 21, size = 2)
```



10.1.2.3 FORMAS POR GRUPO

Si se desea que cada grupo tenga un símbolo propio y diferente, se pasa la variable categórica al argumento `shape` de la función `aes()`. Hay que tener en cuenta que se puede especificar este argumento sin tener que especificar el argumento `color`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo,
               shape = grupo)) +
  geom_point(size = 2, alpha = 0.5)
```



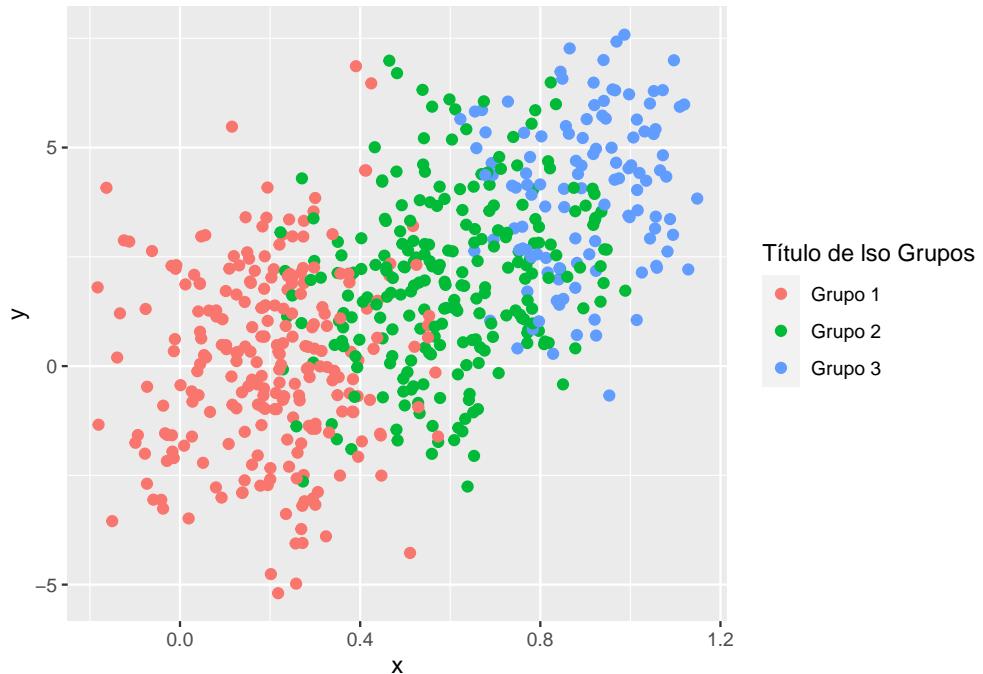
10.1.3 PERSONALIZACIÓN DE LA LEYENDA

10.1.3.1 TÍTULO DE LA LEYENDA

Cuando se crea un diagrama de dispersión se crea una leyenda de manera automática basada en la variable categórica. El título por defecto de la leyenda es el nombre de la variable, pero se puede sobre escribir con el siguiente código.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point(size = 2) +
  guides(colour = guide_legend(title = "Título de los Grupos"))
```

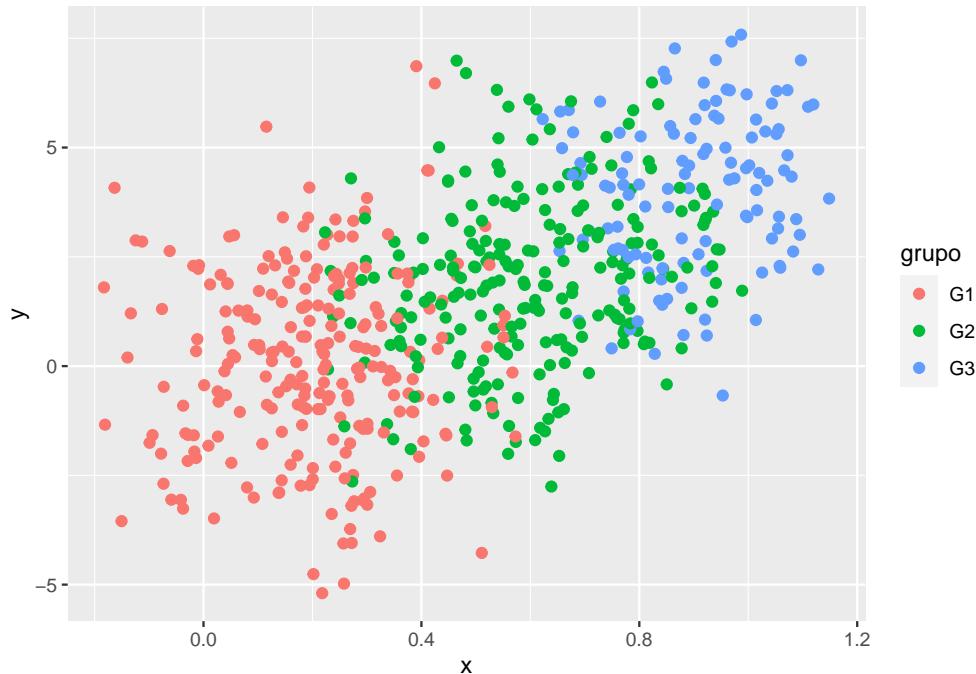


10.1.3.2 ETIQUETAS

Las etiquetas también se pueden personalizar. Se puede hacer uso del argumento `labels` de la función `scale_color_discrete()` para cambiarlas.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point(size = 2) +
  scale_color_discrete(labels = c("G1", "G2", "G3"))
```

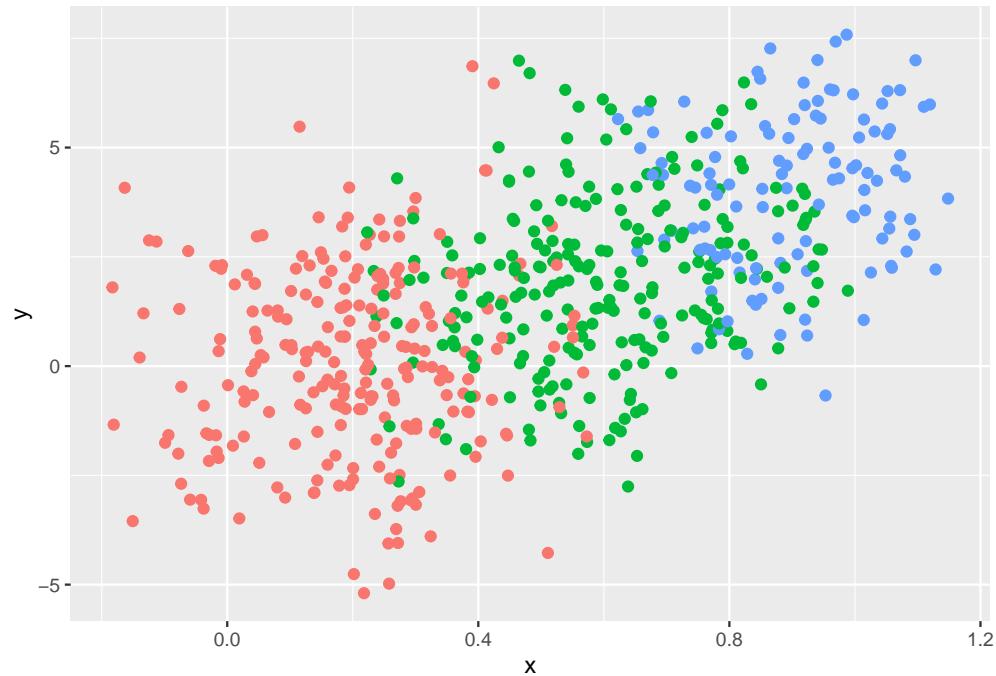


10.1.3.2 ELIMINAR LA LEYENDA

En caso de que se quiera uno deshacer de la leyenda, sólamente se establece el argumento `legend.position = "none"` dentro de la función `theme()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point(size = 2) +
  theme(legend.position = "none")
```



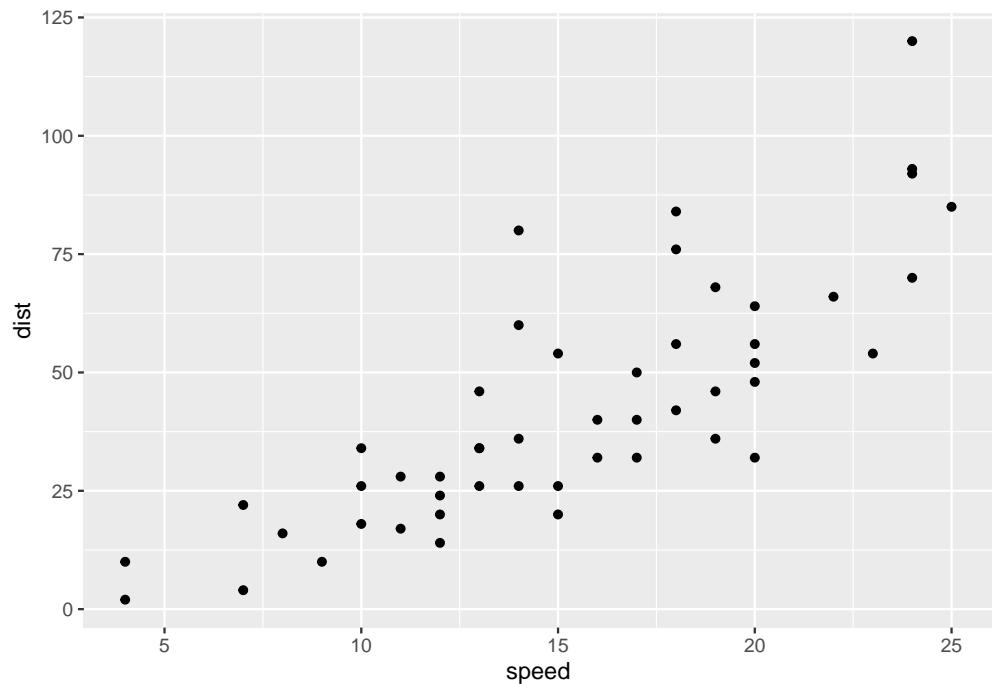
10.2 GRÁFICO DE DIPERSIÓN EN `ggplot2`

10.2.1 DIAGRAMA DE DISPERSIÓN EN `ggplot2` CON LA FUNCIÓN `geom_point()`

La función `geom_point()` se puede utilizar para crear un diagrama de dispersión con el paquete de `ggplot2` (también conocido como gráfico de dispersión o una nube de puntos). Utilizando el conjunto de datos `cars`, se puede crear la siguiente visualización.

```
# install.packages("ggplot2")
library(ggplot2)

# Diagrama básico de dispersión
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()
```

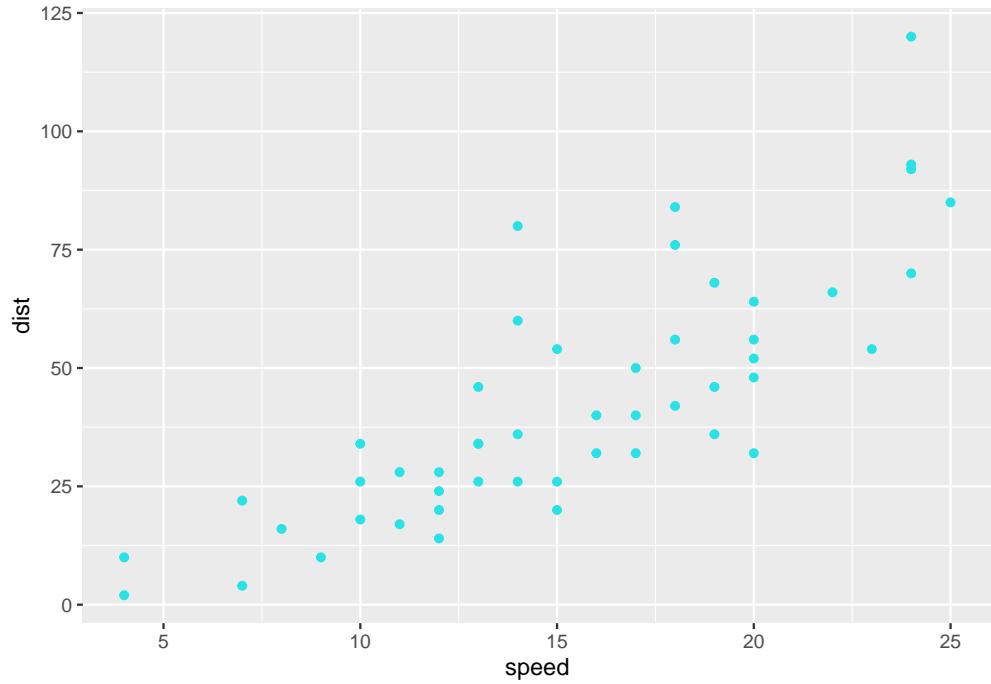


10.2.1.1 COLOR

El argumento `colour` (o `color`) de la función `geom_point()`, permite personalizar el color de los puntos.

```
# install.packages("ggplot2")
library(ggplot2)
```

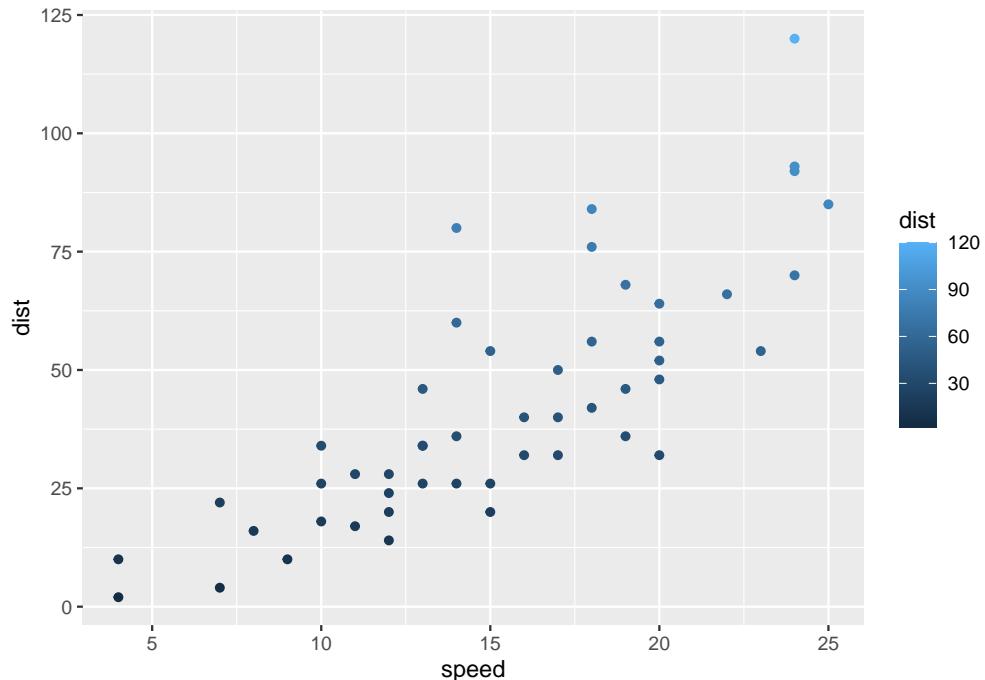
```
ggplot(cars, aes(x = speed, y = dist)) +  
  geom_point(colour = 5)
```



10.2.1.2 DEGRADADO

Si se desea colorear las observaciones con base a una variable, se puede pasar la variable al argumento `color` dentro de la función `aes()`.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
ggplot(cars, aes(x = speed, y = dist, color = dist)) +  
  geom_point()
```



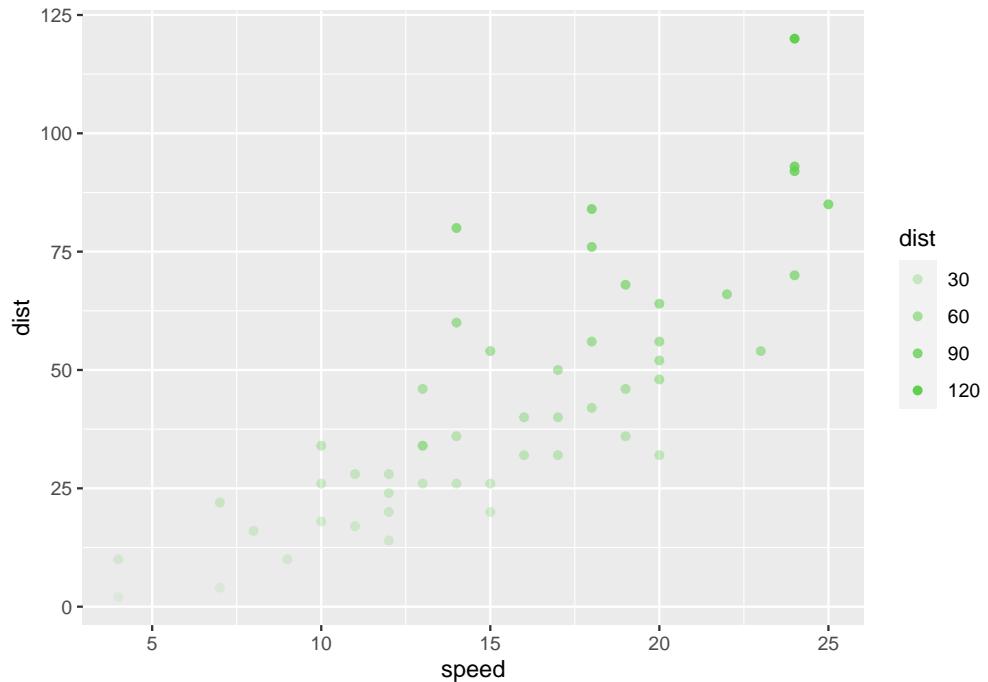
10.2.1.3 TRANSPARENCIA

La transparencia de los puntos también se pueden personalizar con el argumento `alpha`. Se puede pasar un valor o una variable, de modo que la transparencia se basará en dicha variable, como en el siguiente ejemplo.¹⁷

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(cars, aes(x = speed, y = dist, alpha = dist)) +
  geom_point(colour = 3)
```

¹⁷Estableciendo el argumento `show.legend = FALSE` dentro de la función `geom_point()`, se eliminará la leyenda creada de manera automática cuando se especifica `colour`, `alpha` o ambos.

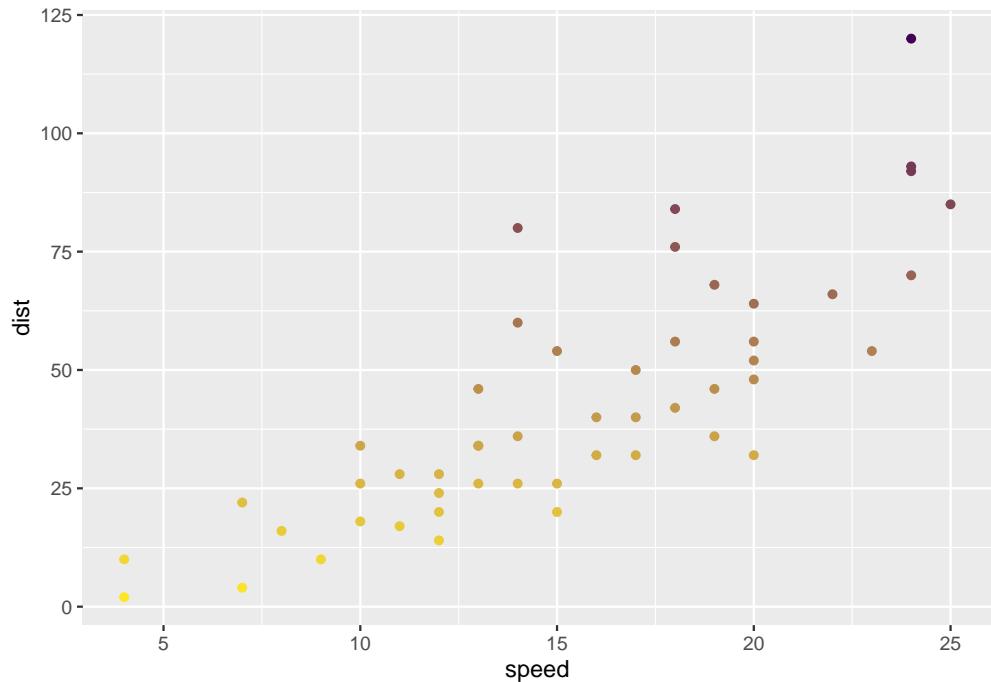


10.2.1.4 ESCALA DE COLOR

Si se pasa una variable al argumento `colour` dentro de la función `aes()`, se puede personalizar la escala de color `scale_color_gradient()` (o una función similar), estableciendo el color para el valor más bajo (`low`) y para el color con el valor más alto (`high`).

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(cars, aes(x = speed, y = dist,
                  colour = dist)) +
  geom_point(show.legend = FALSE) +
  scale_color_gradient(low = "#fde725", high = "#440154")
```

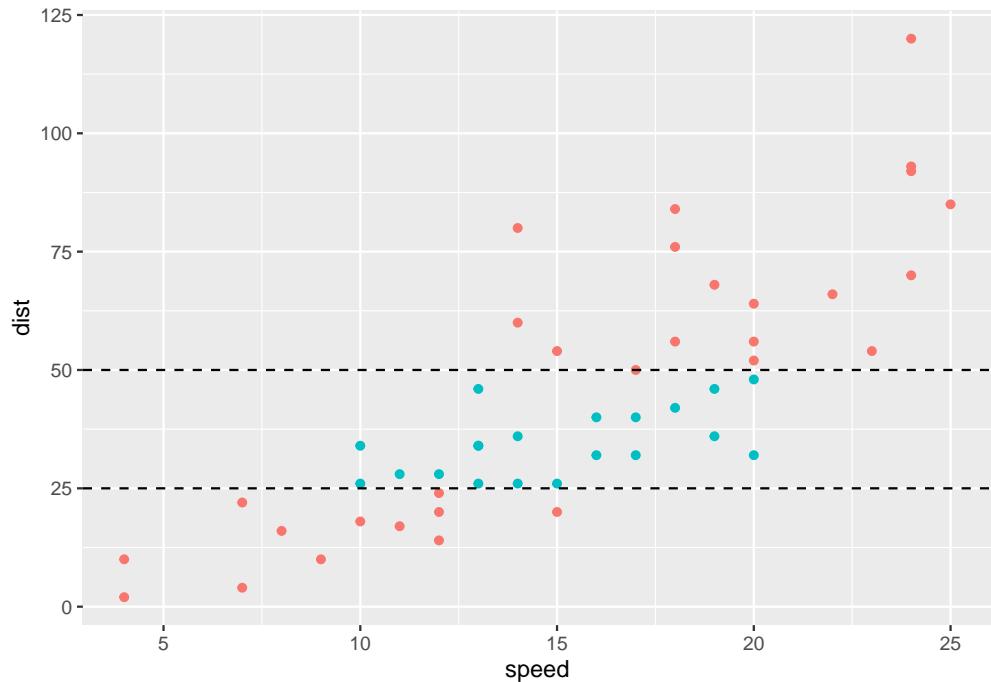


10.2.1.5 COLOR BASADO EN VALORES

La función `geom_point()` permite colorear los puntos basados en ciertas condiciones. Hay que tener en cuenta que se pueden personalizar los colores con la función `scale_color_manual()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(cars, aes(x = speed, y = dist)) +
  geom_point(aes(colour = dist > 25 & dist < 50),
             show.legend = FALSE) +
  geom_hline(yintercept = 25, linetype = "dashed") +
  geom_hline(yintercept = 50, linetype = "dashed")
```



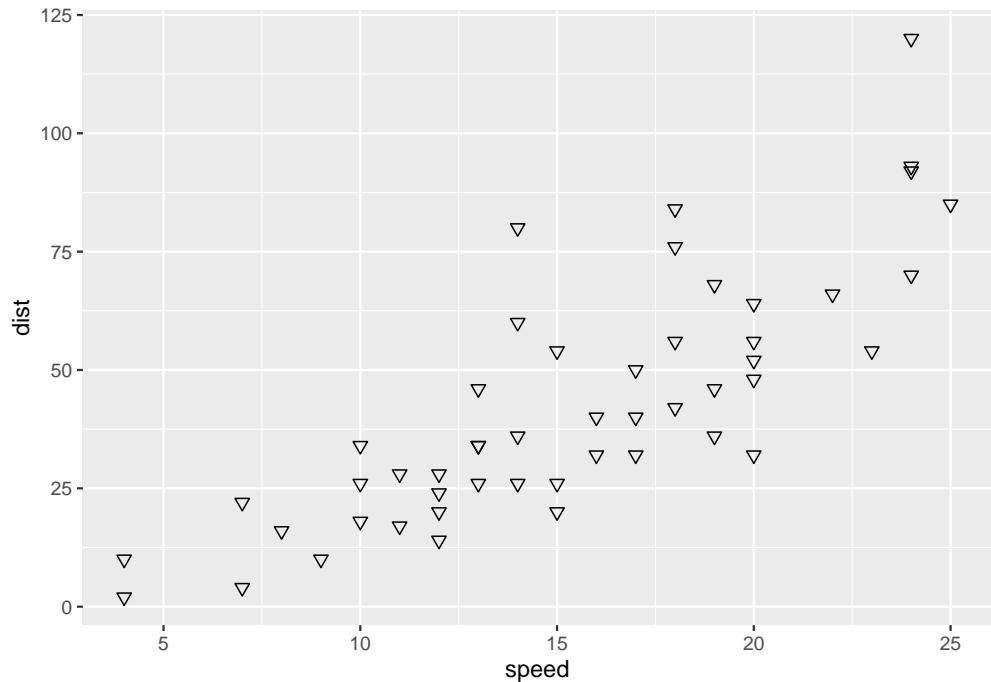
10.2.2 FORMA Y TAMAÑO

10.2.2.1 TAMAÑO Y FORMA DE LOS PUNTOS

La forma y el tamaño de los puntos se puede cambiar con los argumentos `size` y `shape`, respectivamente.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(cars, aes(x = speed, y = dist)) +
  geom_point(size = 2, shape = 6)
```



10.3 GRÁFICO DE DIPERSIÓN CON HISTOGRAMAS MARGINALES EN ggplot2

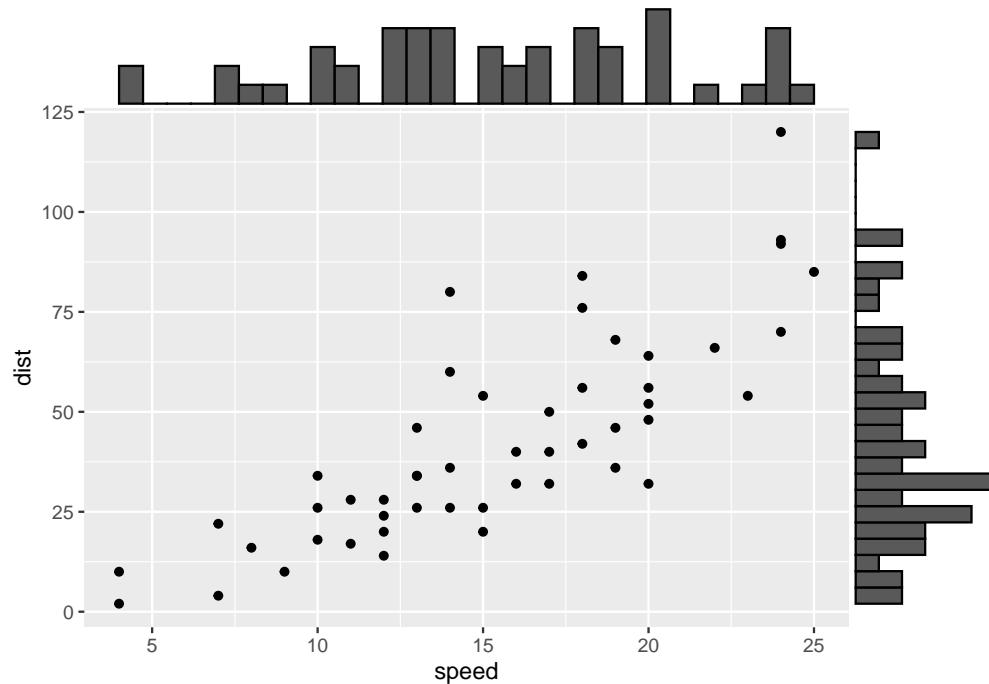
10.3.1 AGREGANDO HISTOGRAMAS MARGINALES CON ggExtra

La función `ggMarginal` del paquete `ggExtra`, permite añadir histogramas marginales a un gráfico de dispersión ya generado. Para tal propósito se tendrá que guardar el diagrama de dispersión que se realizó con el paquete `ggplot2` dentro de una variable. Posteriormente se pasa la función `ggMarginal`, especificando dentro el argumento `type = "histogram"`.¹⁸

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Crea el gráfico con histogramas marginales
ggMarginal(p, type = "histogram")
```



¹⁸Cabe mencionar que entre los tipos de gráficos que se le pueden añadir se encuentran: `c("density", "histogram", "boxplot", "violin", "densigram")`.

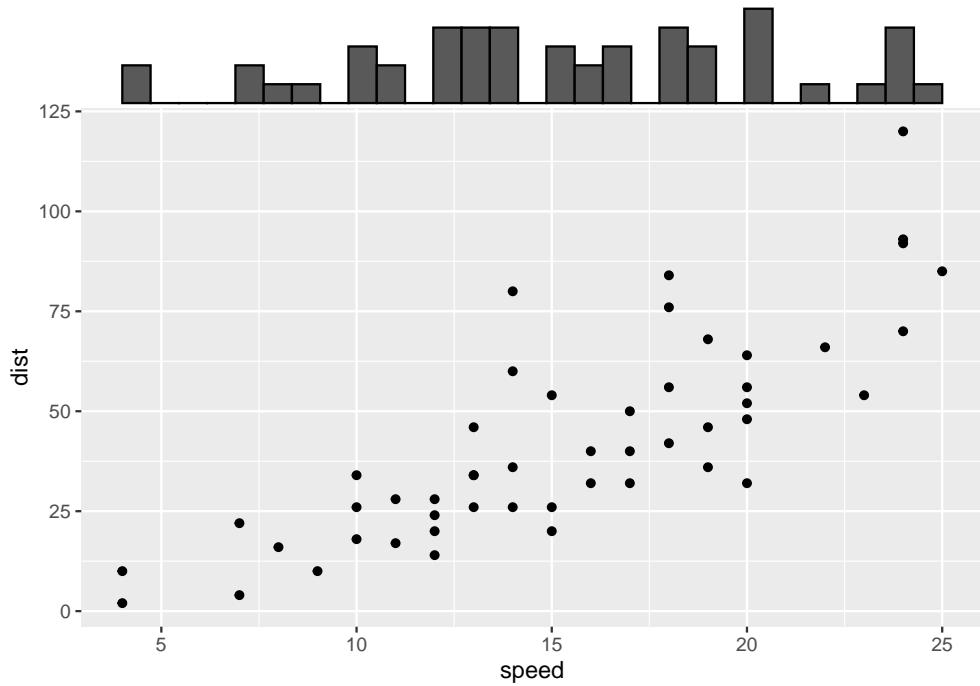
10.3.1.1 HISTOGRAMA SÓLO EN EL EJE X

El argumento `margins` se puede usar para añadir sólo uno de los histogramas marginales, en este caso el del eje X.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Histograma marginal horizontal
ggMarginal(p, type = "histogram",
            margins = "x")
```



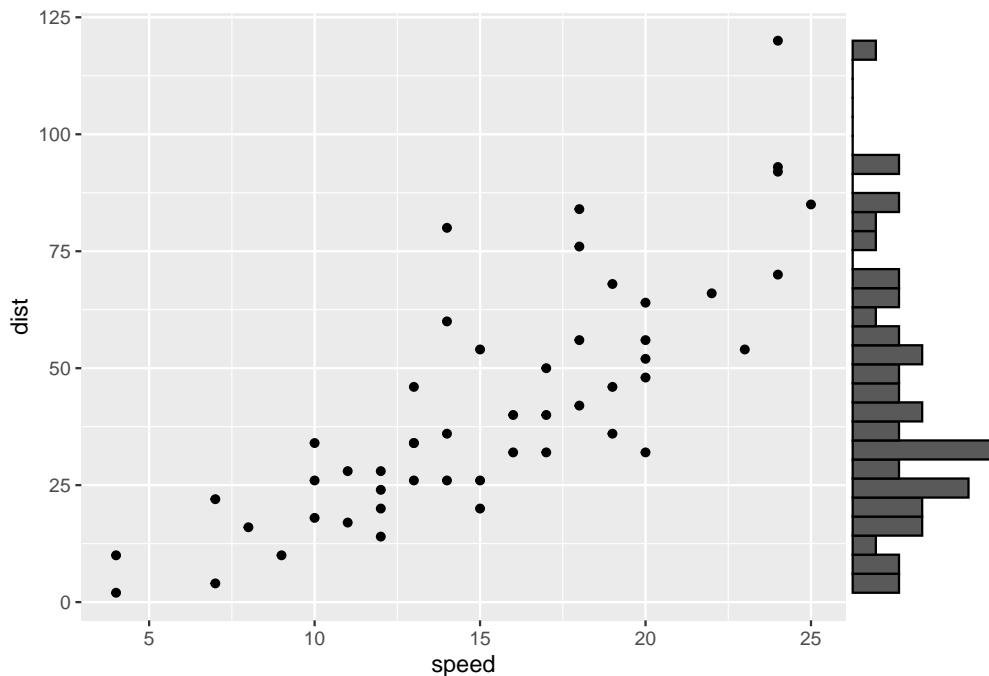
10.3.1.2 HISTOGRAMA SOLO EN EL EJE VERTICAL

Si se prefiere mostrar únicamente en el eje Y el histograma, se establece el argumento `margins = "y"`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Histograma marginal vertical
ggMarginal(p, type = "histogram",
            margins = "y")
```



10.3.1.3 TAMAÑO RELATIVO

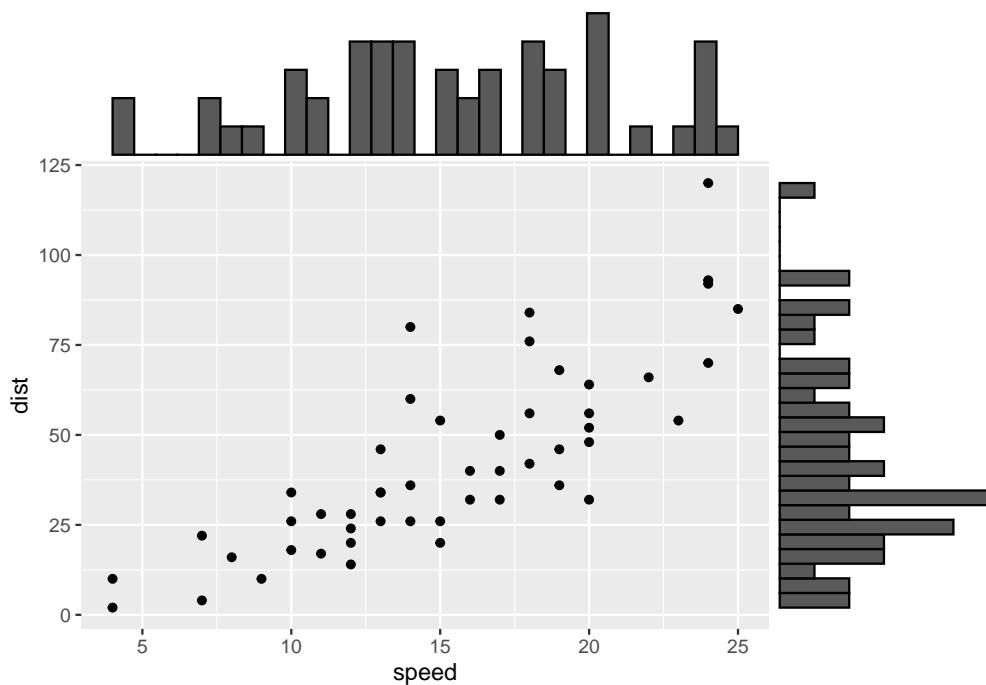
El argumento `size` modifica el tamaño relativo entre los histogramas y el gráfico de dispersión. El valor por defecto es 5 (el gráfico de dispersión es 5 veces más grande que los histogramas).

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
```

```
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Cambiando el tamaño relativo
ggMarginal(p, type = "histogram",
            size = 3)
```



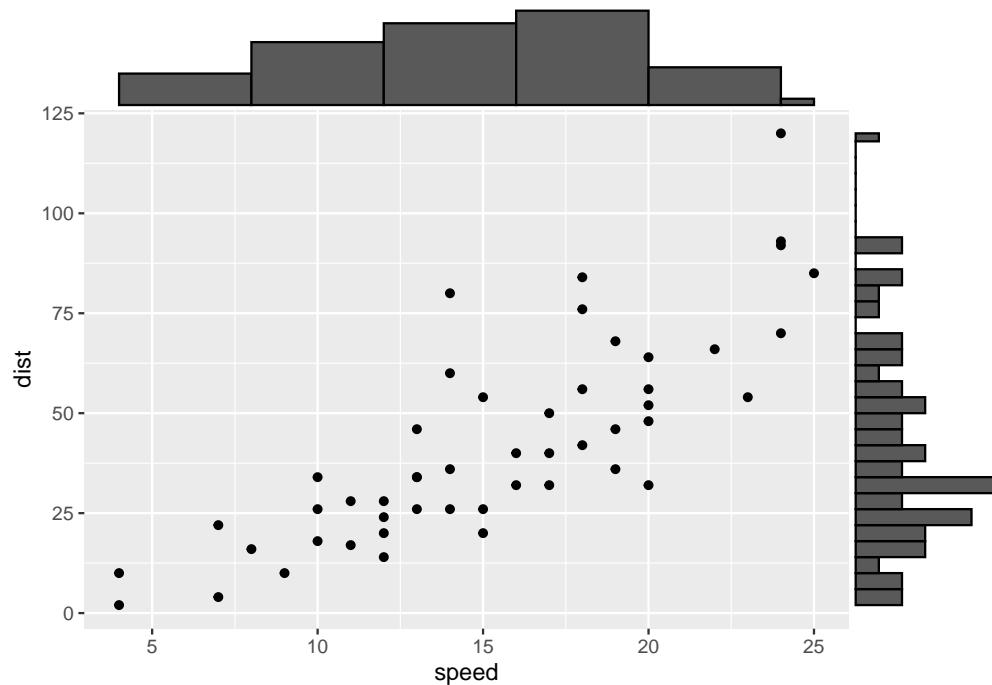
10.3.1.4 TAMAÑO DE LAS BARRAS DEL HISTOGRAMA

Las barras del histograma pueden cambiar, teniendo diferentes rangos con el argumento `binwidth`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Personalización barras
ggMarginal(p, type = "histogram",
            binwidth = 4)
```



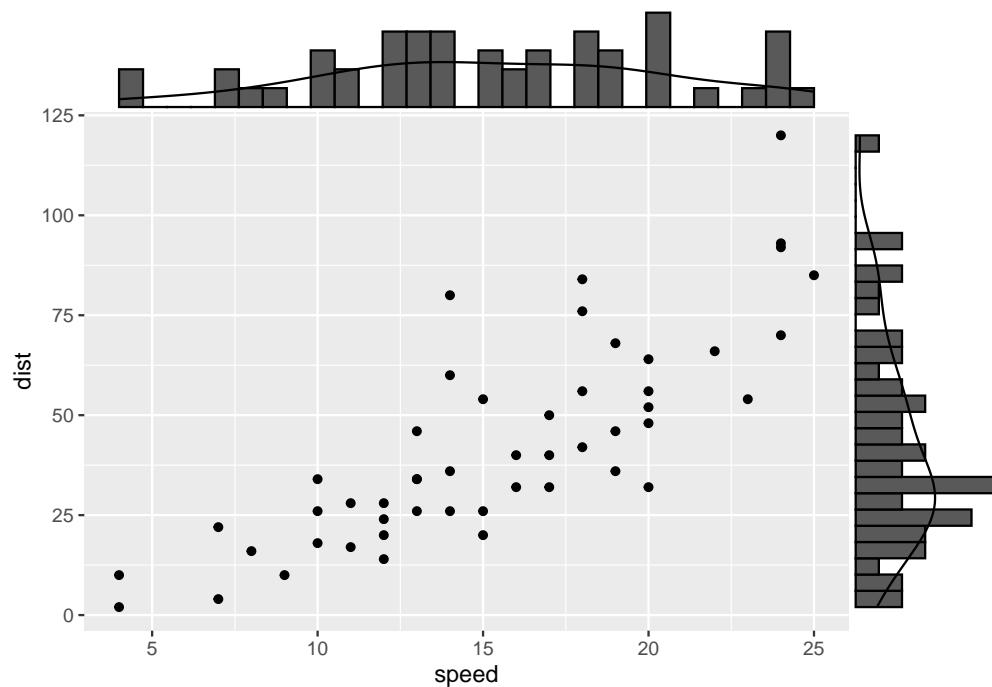
10.3.1.5 HISTOGRAMA CON CURVAS DE DENSIDAD

Si se quiere superponer las curvas de densidad de los histogramas, se usa el argumento `type = "densigram"`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Densograma
ggMarginal(p, type = "densigram")
```



10.3.2 PERSONALIZACIÓN DEL COLOR

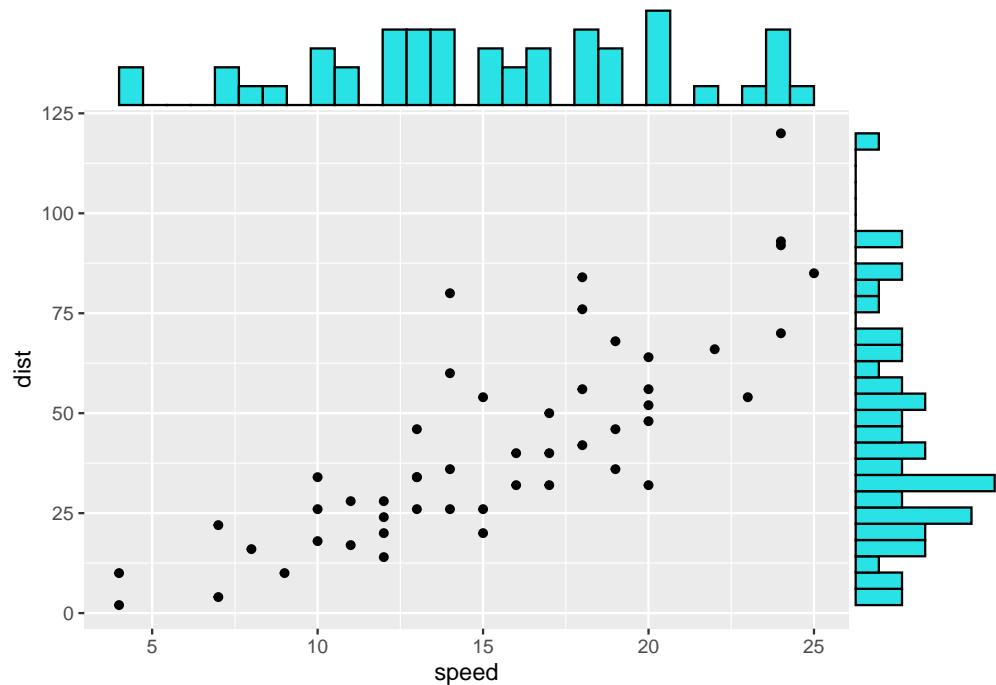
10.3.2.1 COLOR DE FONDO

Cambiar el color de fondo de los histogramas con el argumento `fill`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p <- ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Cambiando el color de fondo
ggMarginal(p, type = "histogram",
           fill = 5)
```



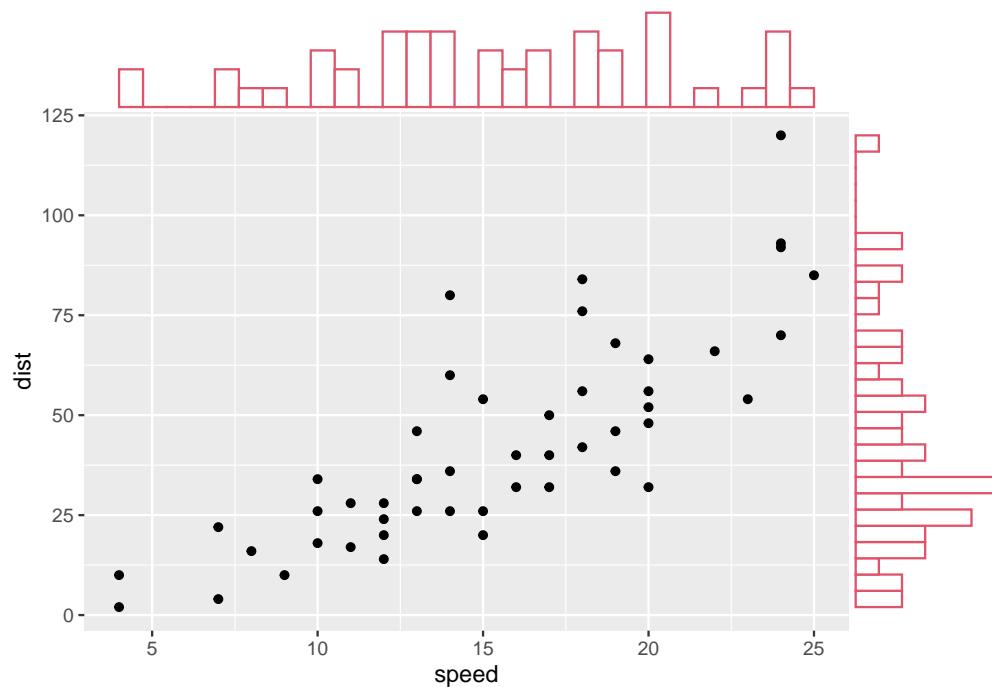
10.3.2.2 COLOR DEL BORDE

También se puede personalizar el color del borde con el argumento `col`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Color de fondo y de borde de los histogramas marginales
ggMarginal(p, type = "histogram",
            fill = "white",
            col = 2)
```



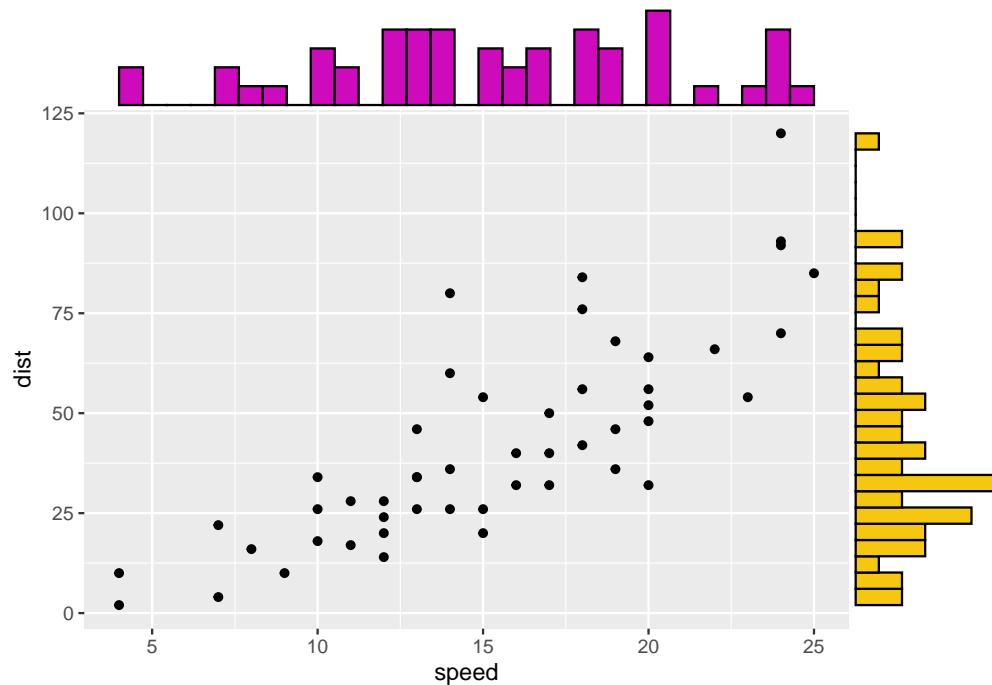
10.3.2.3 ARGUMENTOS PARA CADA HISTOGRAMA

Si se desea personalizar cada histograma por separado, se pasa una lista de argumentos para cada eje con `xparams` e `yparams`.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()

# Argumentos para cada histograma marginal
ggMarginal(p, type = "histogram",
            xparams = list(fill = 6),
            yparams = list(fill = 7))
```



10.3.2.4 HISTOGRAMAS POR GRUPO

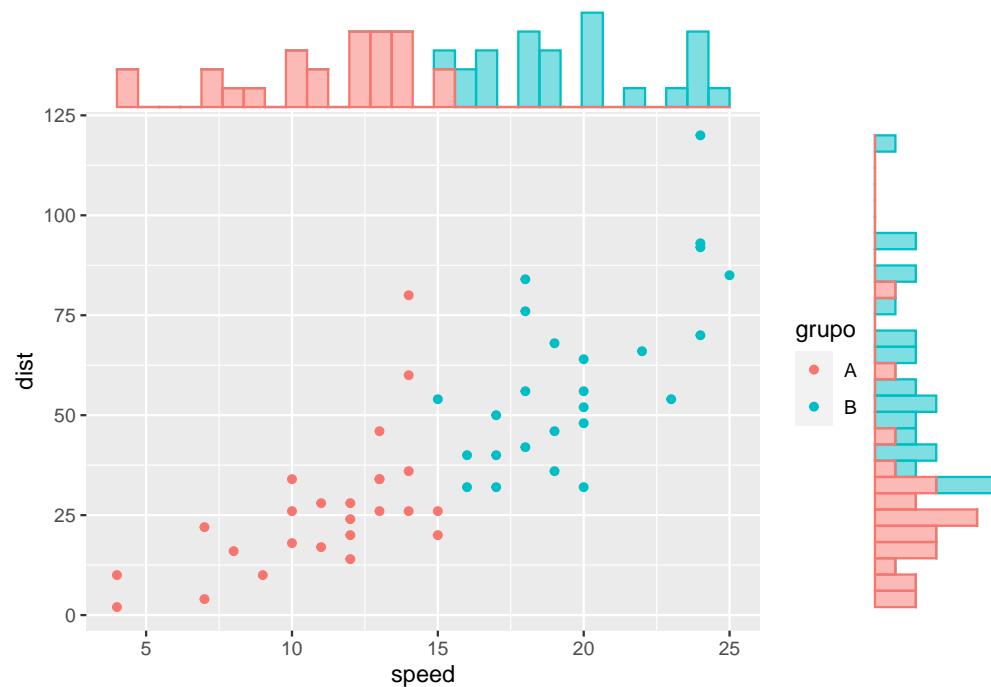
Por último, si el conjunto de datos contiene una variable categórica que representa grupos, se puede crear histogramas para cada grupo en cada margen.

```
# install.packages("ggplot2")
# install.packages("ggExtra")
library(ggplot2)
library(ggExtra)

# Variable categórica de ejemplo
cars$grupo <- c(rep("A", 25), rep("B", 25))

# Guarda el gráfico de dispersión en una variable
p = ggplot(cars, aes(x = speed, y = dist, color = grupo)) +
  geom_point()

# Histogramas marginales por grupo
ggMarginal(p, type = "histogram",
            groupColour = TRUE,
            groupFill = TRUE)
```



10.4 GRÁFICO DE DISPERSIÓN CONECTADO EN ggplot2

10.4.1 GRÁFICO DE DISPERSIÓN CONECTADO CON LA FUNCIÓN geom_path()

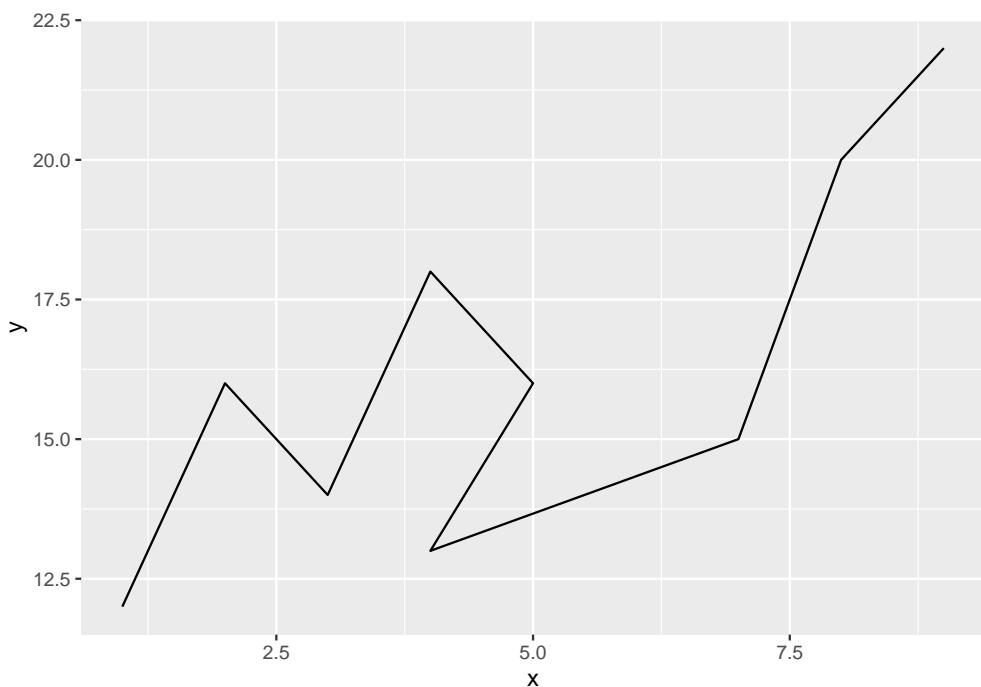
10.4.1.1 LA FUNCIÓN geom_path()

Dadas dos variables en un data frame, se puede dibujar el camino que recorren con `geom_path()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
x = c(1, 2, 3, 4, 5, 4, 7, 8, 9)
y = c(12, 16, 14, 18, 16, 13, 15, 20, 22)
df = data.frame(x, y)

# Gráfico dispersión conectado
ggplot(df, aes(x = x, y = y)) +
  geom_path()
```



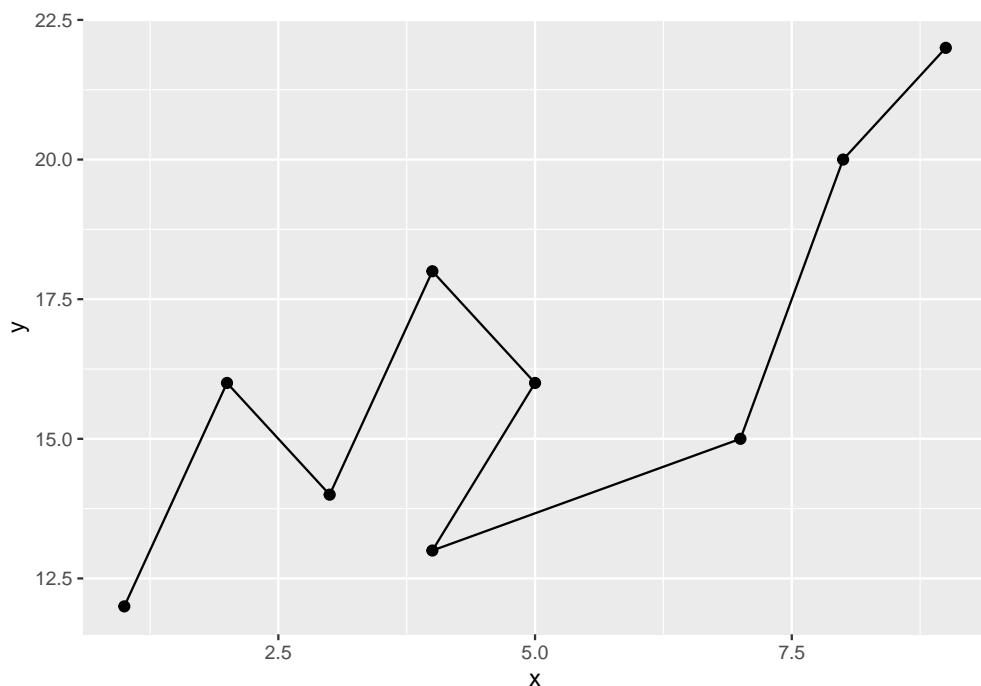
10.4.1.2 OBSERVACIONES

Se pueden resaltar los valores con la función `geom_point()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
x = c(1, 2, 3, 4, 5, 4, 7, 8, 9)
y = c(12, 16, 14, 18, 16, 13, 15, 20, 22)
df = data.frame(x, y)

# Gráfico dispersión conectado
ggplot(df, aes(x = x, y = y)) +
  geom_path() +
  geom_point(size = 2)
```



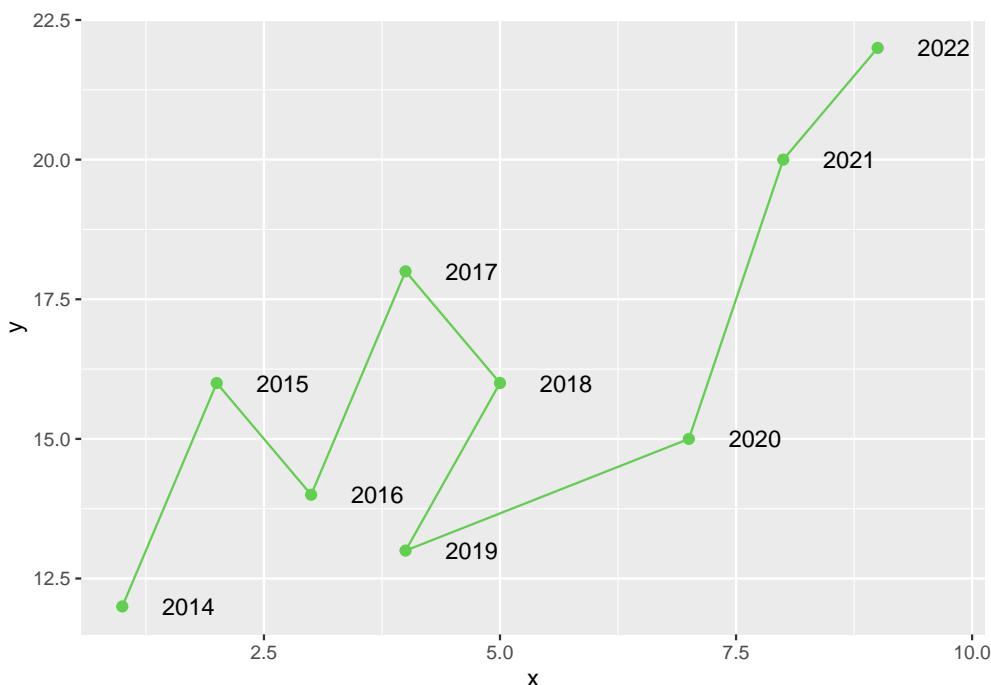
10.4.1.3 ETIQUETANDO OBSERVACIONES

Se puede etiquetar cada observación haciendo uso de la función `geom_text()`. En este ejemplo se añade unas fechas.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
# Datos
x = c(1, 2, 3, 4, 5, 4, 7, 8, 9)
y = c(12, 16, 14, 18, 16, 13, 15, 20, 22)
etiquetas = 2014:2022
df = data.frame(x, y, labels = etiquetas)

# Gráfico dispersión conectado
ggplot(df, aes(x = x, y = y)) +
  geom_path(color = 3) +
  geom_point(size = 2, color = 3) +
  geom_text(aes(label = etiquetas, x = x + 0.7, y = y))
```



10.4.2 AGREGAR FLECHAS

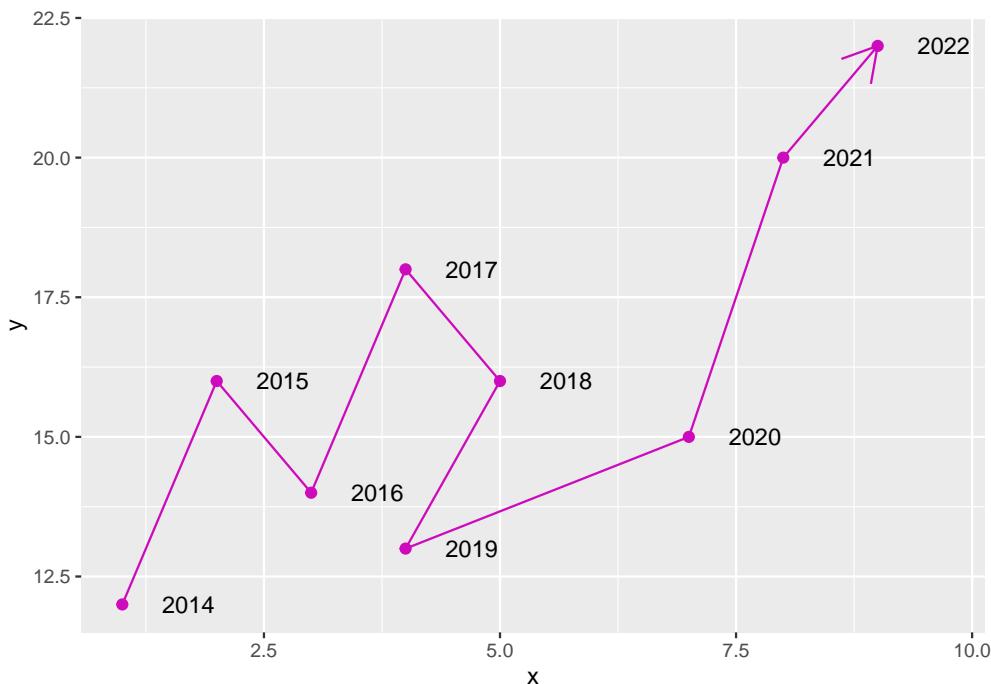
Se puede pasar la función `arrow()` al argumento `arrow` de la función `geom_path()` para añadir una punta de flecha al final de la línea.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
x = c(1, 2, 3, 4, 5, 4, 7, 8, 9)
y = c(12, 16, 14, 18, 16, 13, 15, 20, 22)
etiquetas = 2014:2022
```

```
df = data.frame(x, y, labels = etiquetas)

# Gráfico dispersión conectado
ggplot(df, aes(x = x, y = y)) +
  geom_path(color = 6, arrow = arrow()) +
  geom_point(size = 2, color = 6) +
  geom_text(aes(label = etiquetas, x = x + 0.7, y = y))
```



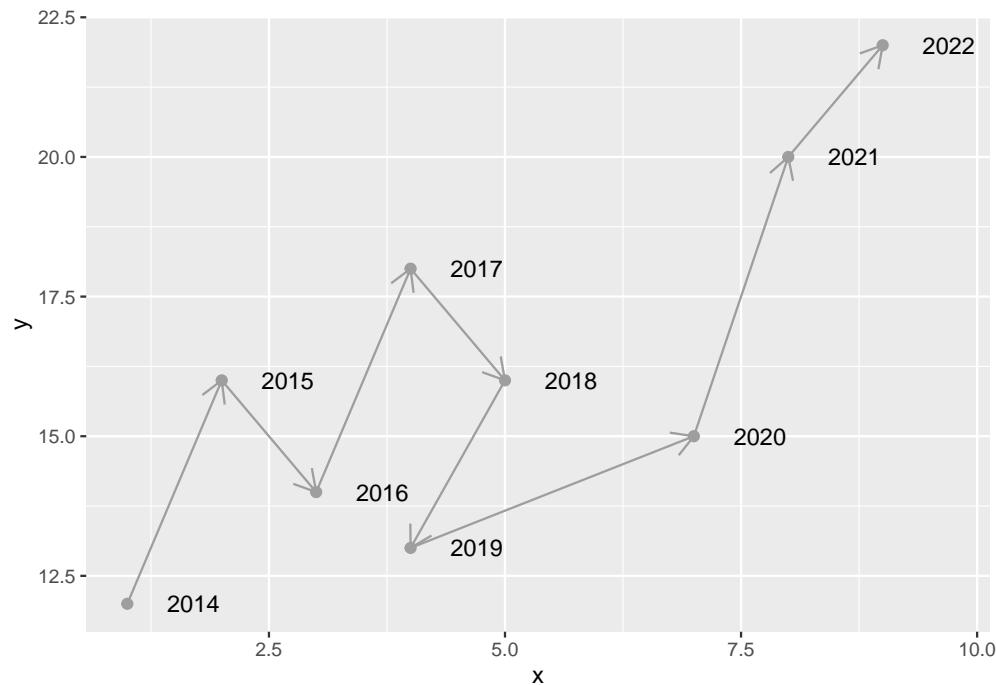
Sin embargo, si se usa la función `geom_segment()`, como se indica a continuación, se puede añadir una flecha entre cada par de observaciones.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
x = c(1, 2, 3, 4, 5, 4, 7, 8, 9)
y = c(12, 16, 14, 18, 16, 13, 15, 20, 22)
etiquetas = 2014:2022
df = data.frame(x, y, labels = etiquetas)

# Gráfico dispersión conectado
ggplot(df, aes(x = x, y = y)) +
  geom_segment(aes(xend = c(tail(x, n = -1), NA),
                   yend = c(tail(y, n = -1), NA)),
```

```
arrow = arrow(length = unit(0.4, "cm")),
color = 8) +
geom_point(size = 2, color = 8) +
geom_text(aes(label = etiquetas, x = x + 0.7, y = y))
```



10.5 GRÁFICO DE DISPERSIÓN CON ELIPSSES EN ggplot2

En este capítulo se considera el siguiente `data frame` para este tutorial, que consiste en dos variables numéricas y una categórica con tres grupos distintos.

```
# Se fija una semilla para que los datos no queden aleatorios
set.seed(999)

# Simulación de datos
x = runif(400)
y = 4 * x ^ 2 + rnorm(length(x), sd = 4)
grupo = ifelse(x < 0.4, yes = "A",
               no = ifelse(x > 0.8, yes = "C",
                           no = "B"))
x = x + runif(length(x), -0.15, 0.15)

# Data frame
df = data.frame(x = x, y = y, grupo = grupo)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center",
    full_width = FALSE
  )
```

x	y	grupo
0.4695968	-5.323157	A
0.4802473	7.444958	B
-0.0395795	1.644628	A
0.9340850	4.783526	C
0.8189403	6.155228	B
-0.0085278	3.114494	A
0.4978638	1.988797	B
-0.0083126	-2.719502	A
0.3275615	2.173894	A
0.5758622	-1.361788	B

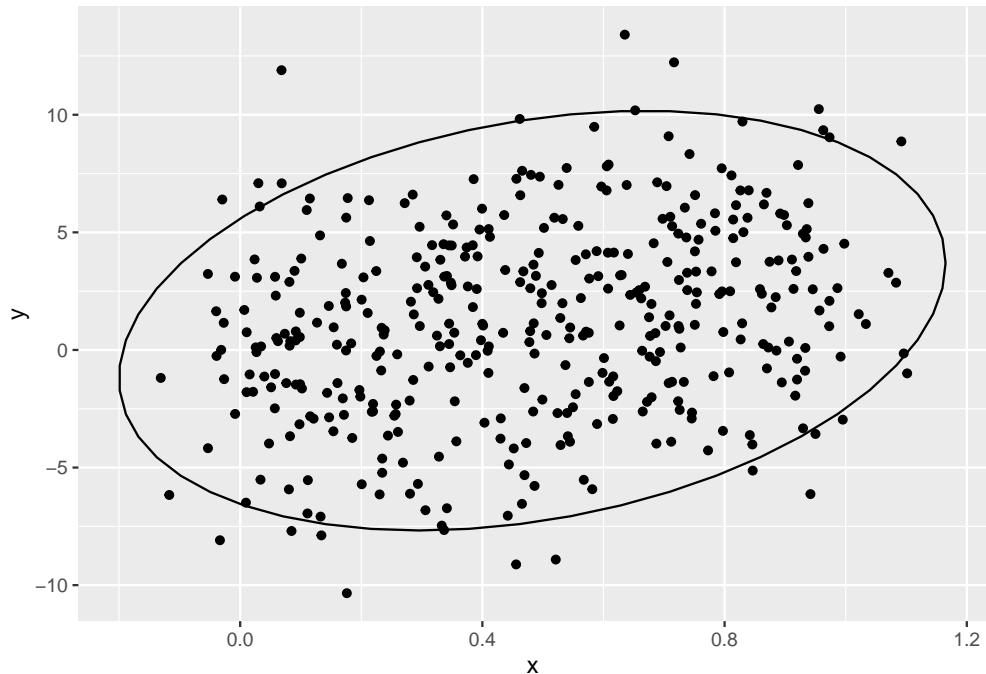
10.5.1 AGREGANDO UNA ELIPSE CON LA FUNCIÓN `stat_ellipse()`

10.5.1.1 ELIPSE POR DEFECTO

Se puede agregar una elipse al diagrama de dispersión añadiendo la capa `stat_ellipse()`, tal y como se muestra en el siguiente ejemplo.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  stat_ellipse()
```

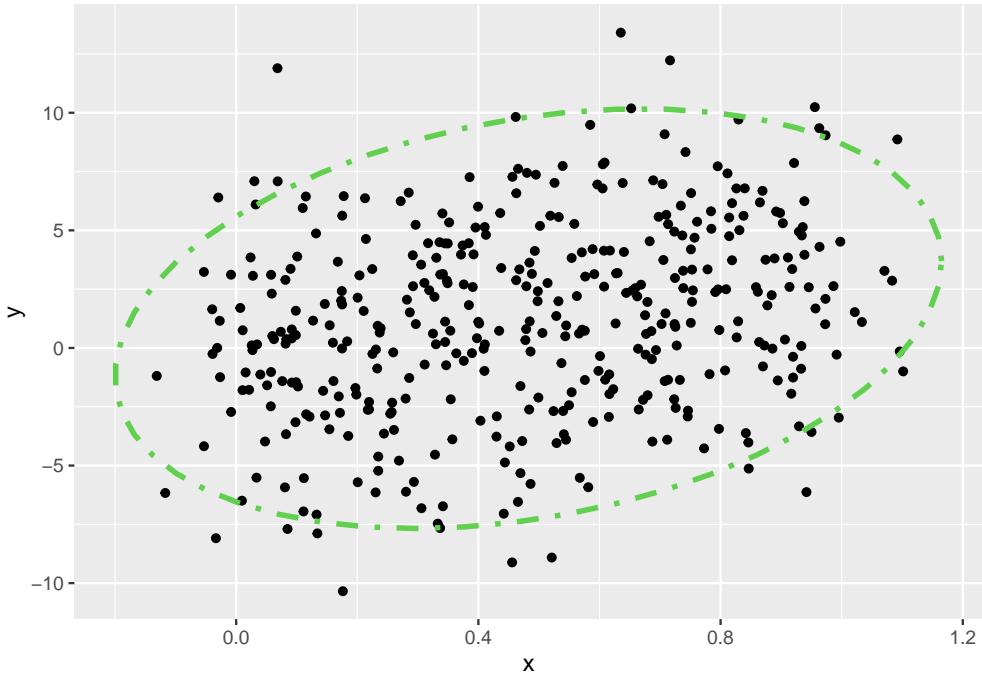


10.5.1.2 PERSONALIZACIÓN

Con el color, el tipo y ancho de línea de las elipses se puede personalizar con los argumentos `color`, `linetype` y `lwd`, respectivamente.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  stat_ellipse(color = 3,
               linetype = 4,
               lwd = 1.2)
```

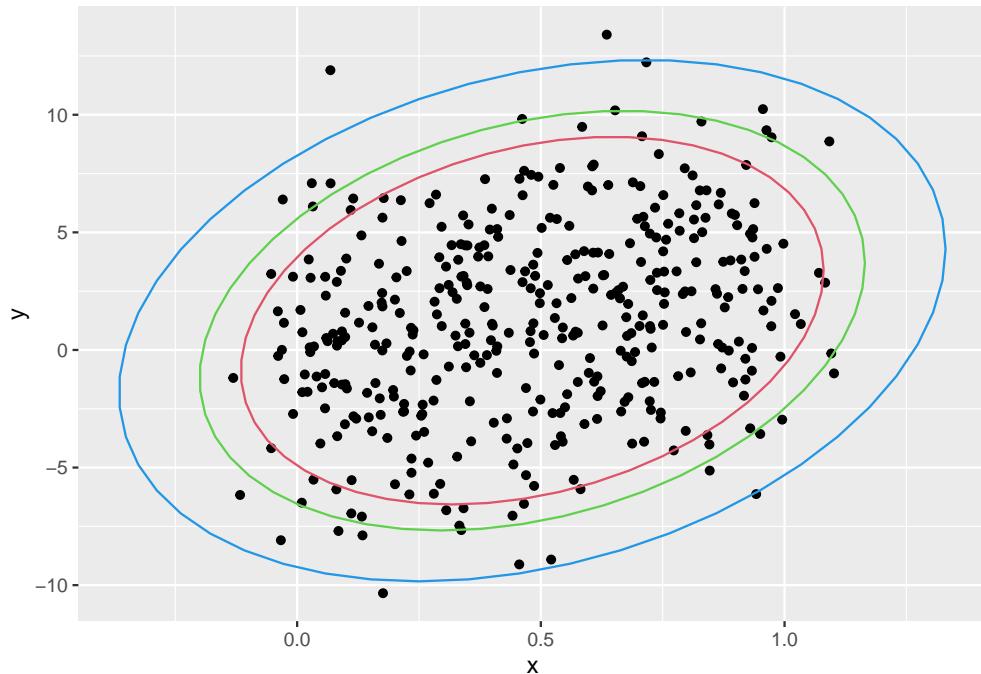


10.5.1.3 INTERVALOS DE CONFIANZA

Por defecto, la función `stat_ellipse()` dibuja un intervalo de confianza del 95 % para una distribución *t-student* multivariante. Se puede modificar este nivel con el argumento `level`. En este caso se presentarán los intervalos de confianza del 99 %, 95 % y 90 %.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  stat_ellipse(level = 0.90, color = 2) +
  stat_ellipse(level = 0.95, color = 3) +
  stat_ellipse(level = 0.99, color = 4)
```

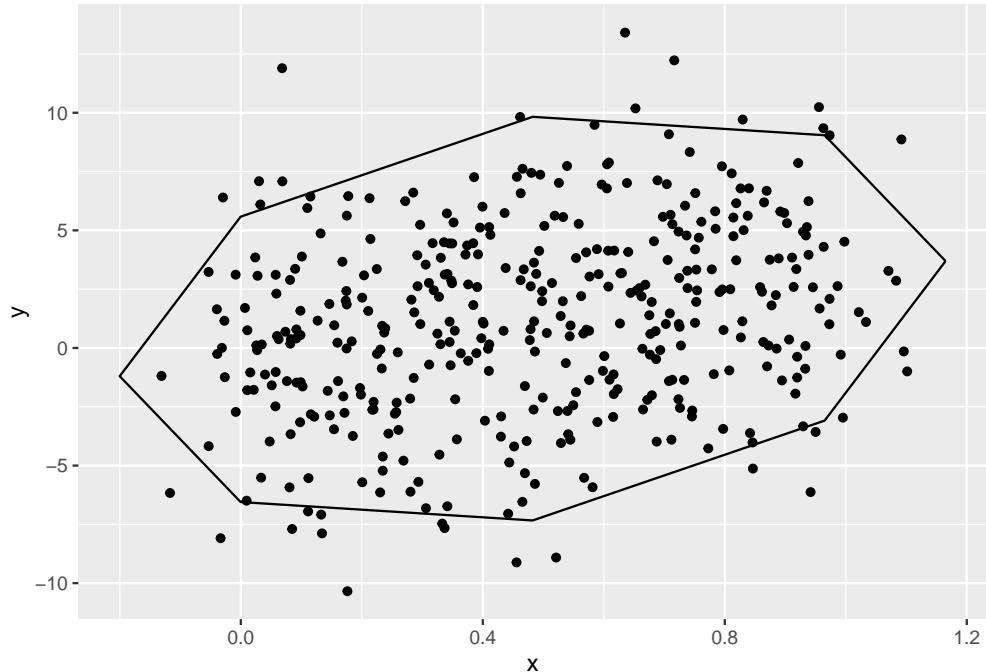


10.5.1.4 SEGMENTOS

El número por defecto de segmentos que se utilizan para crar la elipse es 51, pero se puede modificar este número con el argumento `segments`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  stat_ellipse(segments = 8)
```

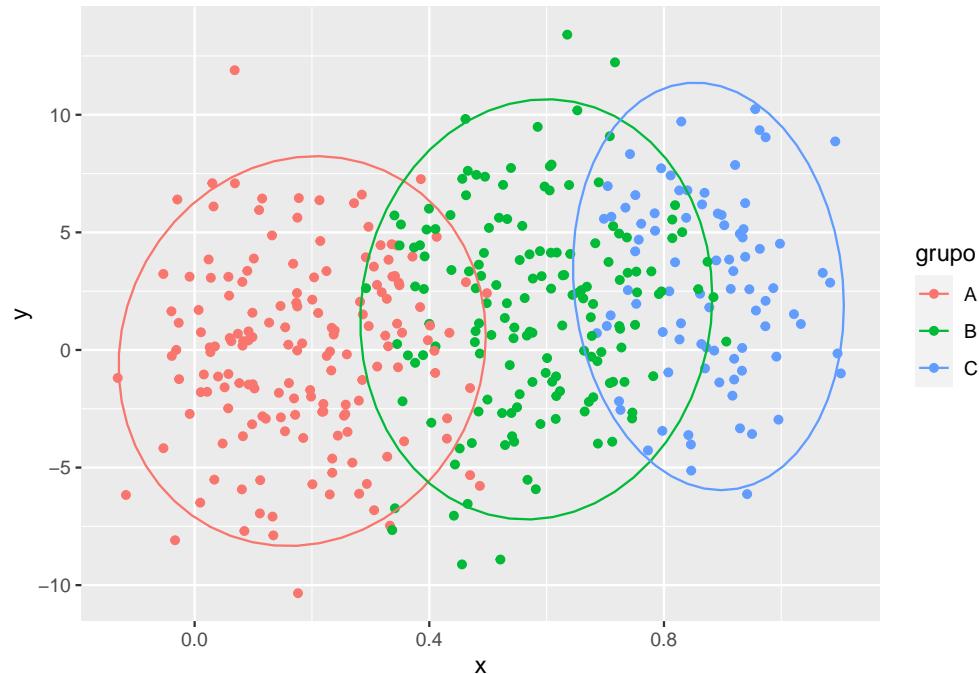


10.5.1.5 ELIPSES POR GRUPO

Cuando se crea un diagrama de dispersión por grupo, las elipses se crean para cada grupo.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point() +
  stat_ellipse()
```

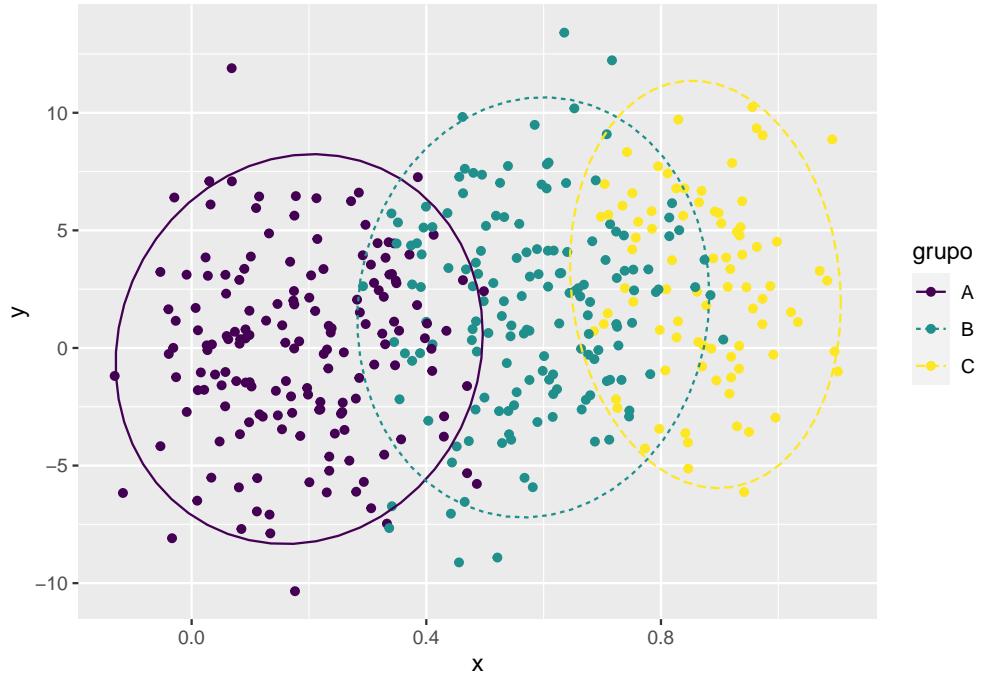


10.5.1.6 TIPO DE LÍNEA POR GRUPO

También se puede cambiar el tipo de línea de las elipses basándose en el grupo, pasando la variable categórica al grupo `linetype` de la función `aes()`.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("viridis")
library(viridis)

ggplot(df, aes(x = x, y = y, color = grupo,
               linetype = grupo)) +
  geom_point() +
  stat_ellipse() +
  scale_color_viridis(discrete = TRUE)
```



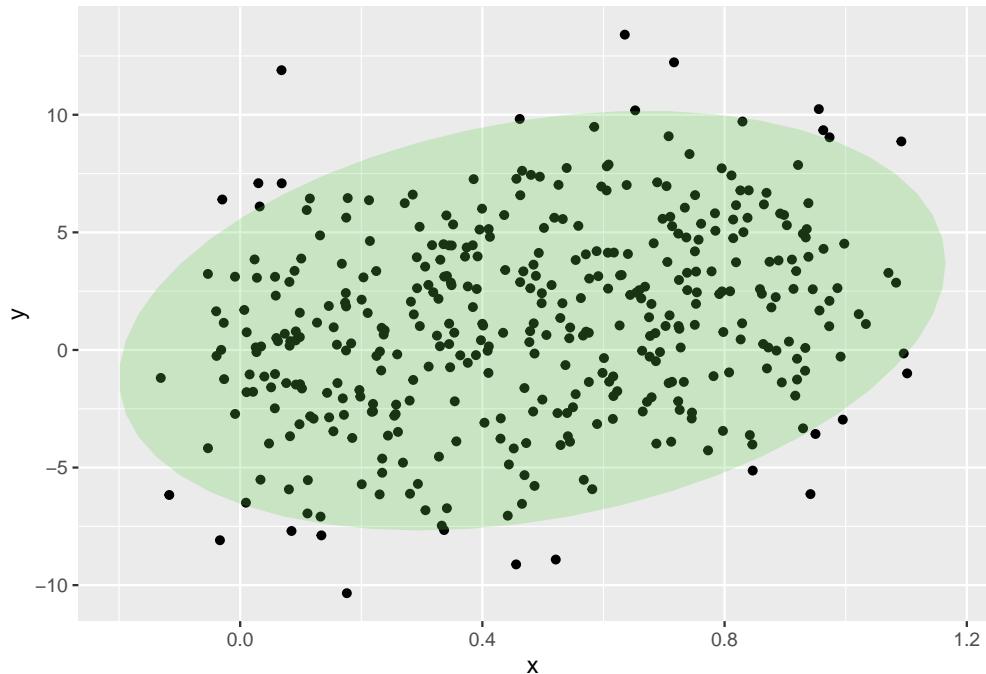
10.5.2 COLOR DEL ÁREA DE LAS ELIPSES

10.5.2.1 COLOR DEL ÁREA

La función `stat_ellipse()` usa `geom_path` por defecto para crear las elipses, pero si se establece el argumento `geom = "polygon"` se creará un polígono con área. Hay que tener en cuenta que se puede cambiar el grado de transparencia con el argumento `alpha`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  stat_ellipse(geom = "polygon",
               fill = 3, alpha = 0.25)
```

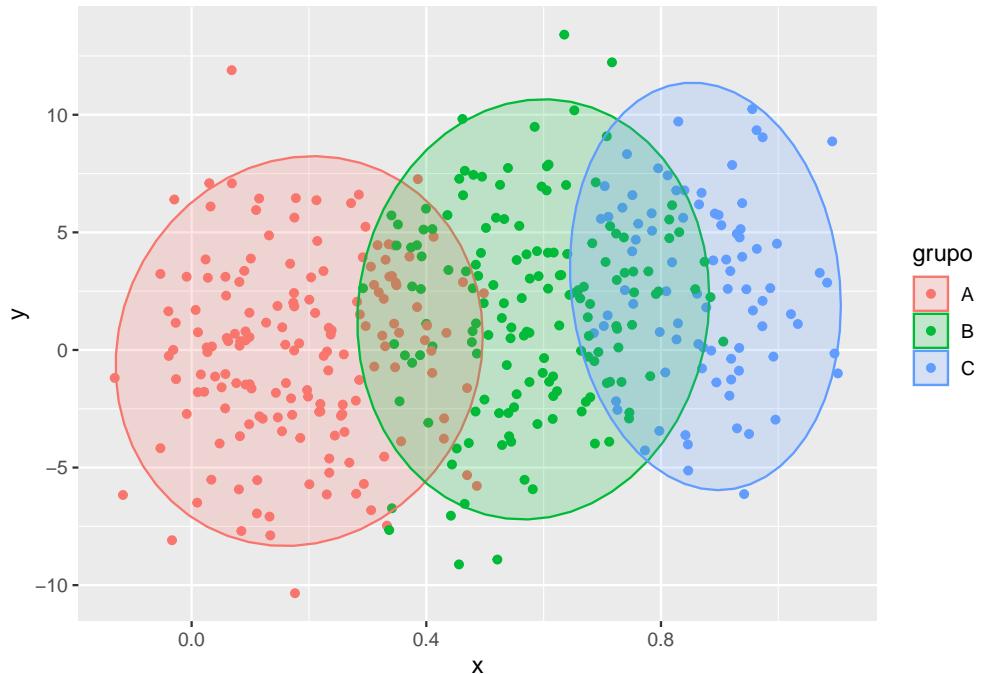


10.5.2.2 COLOR Y TRANSPARENCIA DEL ÁREA POR GRUPO

Tal y como se mostró en uno de los ejemplos anteriores, el argumento `alpha` se puede utilizar para establecer el grado de transparencia de las áreas.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point() +
  stat_ellipse(geom = "polygon",
              aes(fill = grupo),
              alpha = 0.20)
```



10.5.3 TIPOS DE ELIPSES

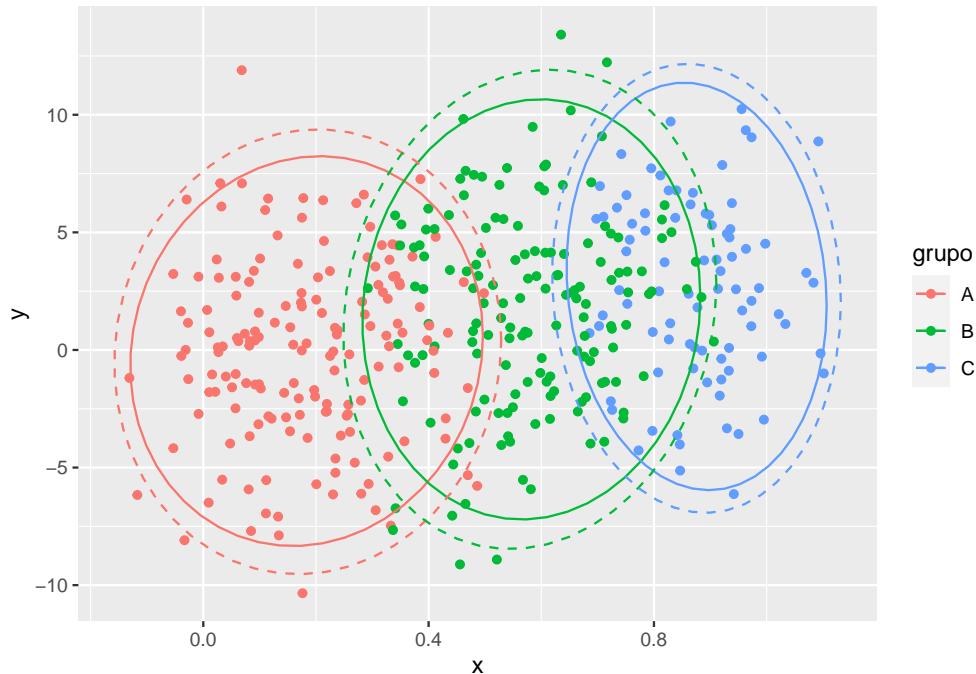
10.5.3.1 ELIPSE NORMAL

Por defecto, la función `stat_ellipse()` asume una distribución *t – student* multivariante (`type = "t"`). Sin embargo, se puede establecer el argumento `type = "norm"` para asumir una distribución normal multivariante.¹⁹

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point() +
  stat_ellipse(type = "t") +
  stat_ellipse(type = "norm", linetype = 2)
```

¹⁹Hay que tener en cuenta que se tienen como tipo: "t", "norm" y "euclid".

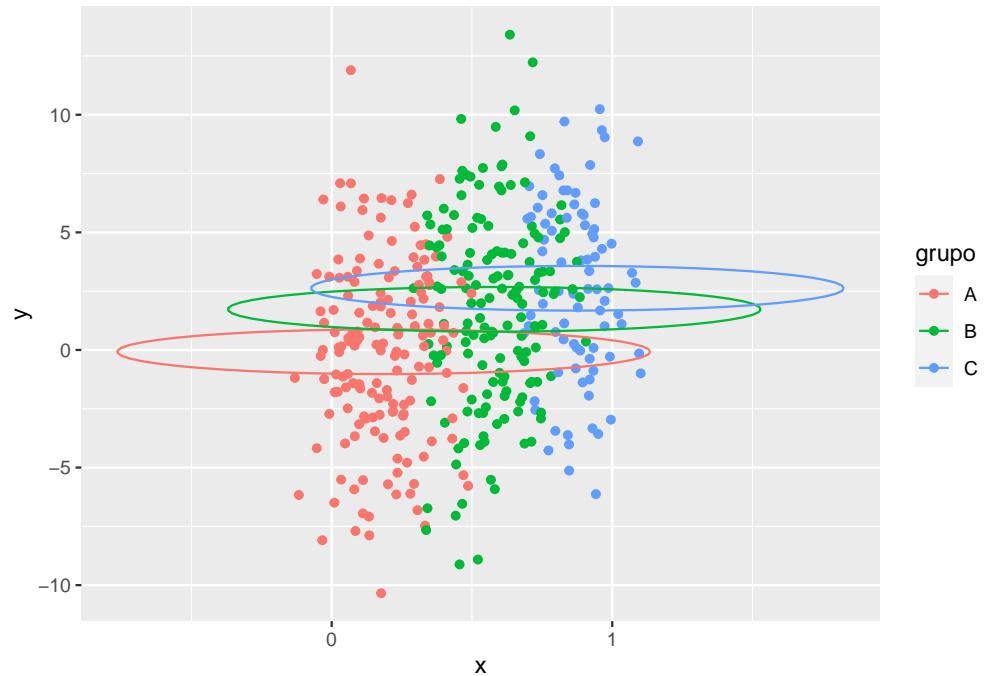


10.5.3.2 ELIPSE EUCLÍDEA

La opción restante es establecer `type = "euclid"` para dibujar una elipse euclídea o euclídeana. Hay que tener en cuenta que la elipse no se verá circular salvo que se establezca la función `coord_flixed()`. En este escenario, si se establece un intervalo con el argumento `level`, el nivel será el radio del círculo dibujado.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, color = grupo)) +
  geom_point() +
  stat_ellipse(type = "euclid")
```



10.6 GRÁFICO HEXBIN EN `ggplot2`

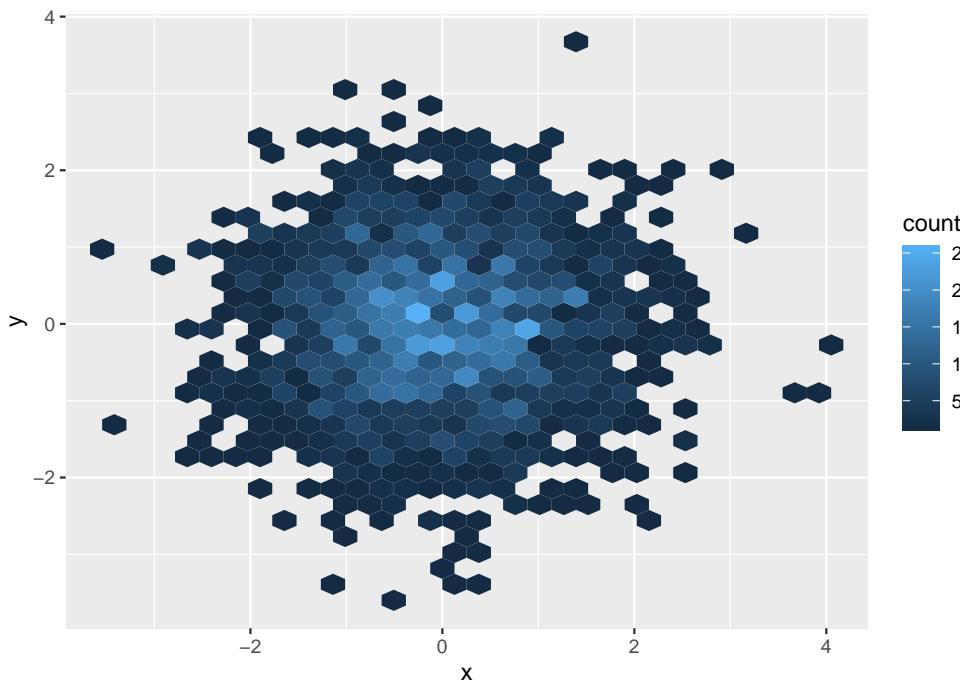
10.6.1 LA FUNCIÓN `geom_hex()`

Dado un data frame con variables numéricas, se puede crear un diagrama hexbin con el paquete `ggplot2` haciendo uso de la función `geom_hex()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex()
```



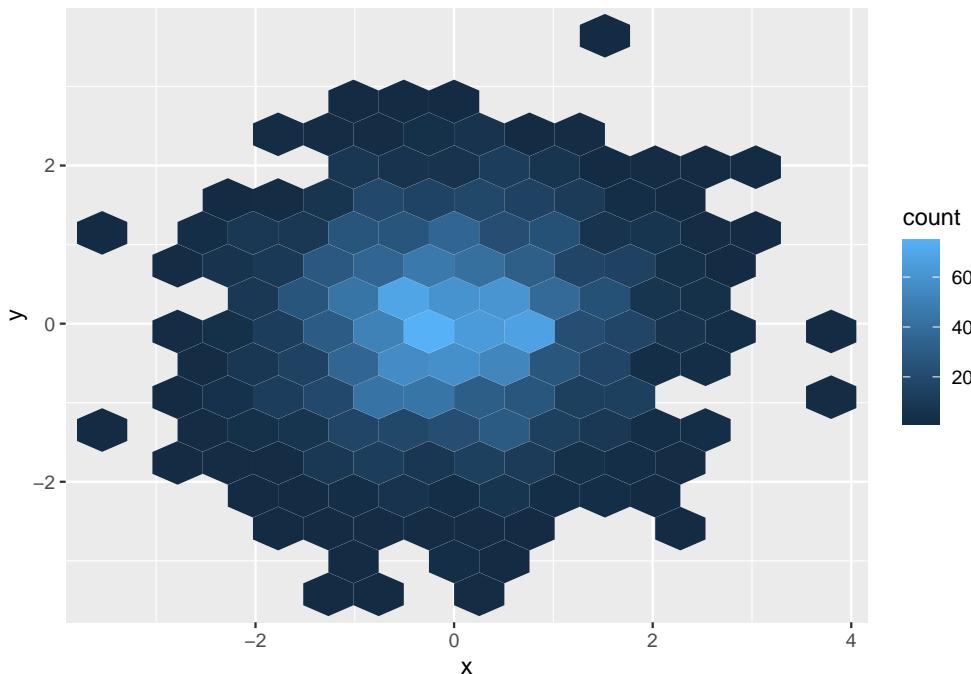
10.6.1.1 POCAS CLASES

Se puede controlar el número de clases en ambas direcciones con el argumento `bins`. El valor por defecto es 30.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex(bins = 15)
```



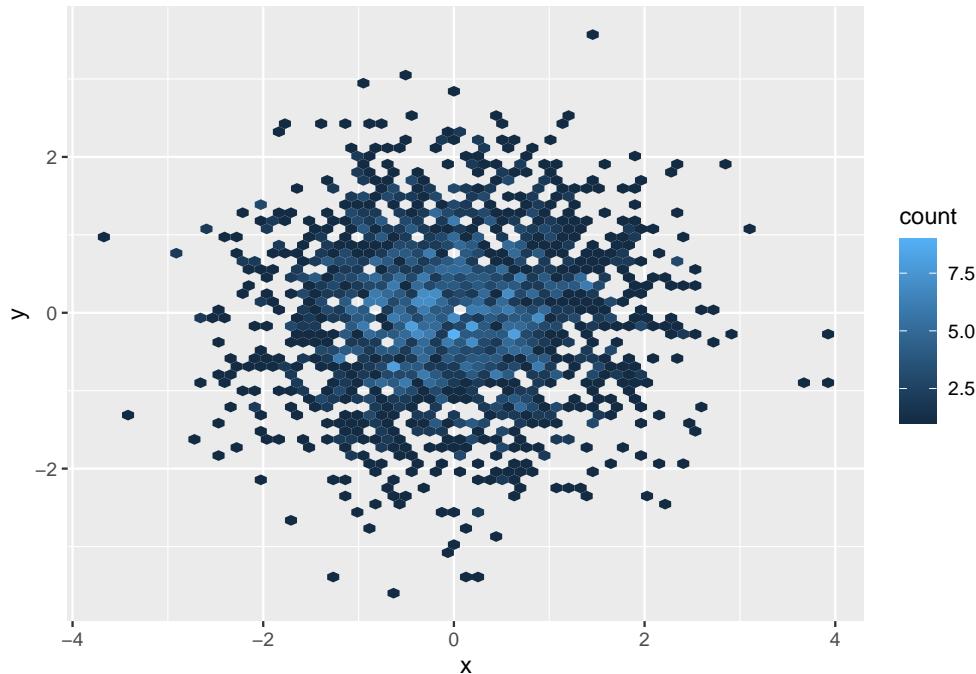
10.6.1.2 MUCHAS CLASES

Hay que tener en cuenta que si se establecen demasiadas clases, el diagrama hexbin tendrá el aspecto de un gráfico de dispersión.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex(bins = 60)
```



10.6.2 PERSONALIZACIÓN DEL COLOR

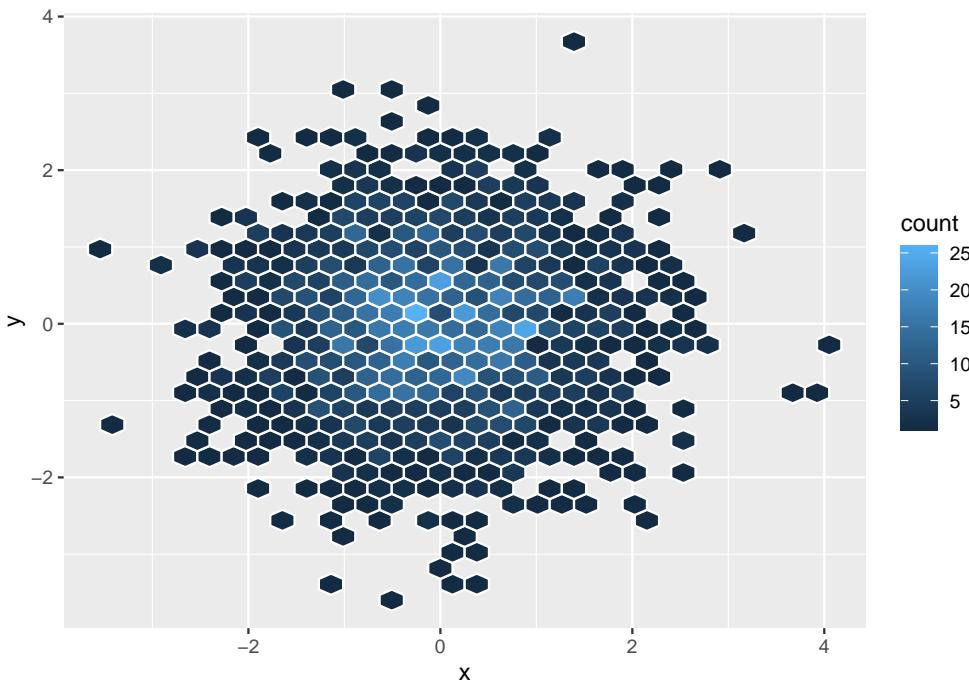
10.6.2.1 COLOR DEL BORDE

El color del borde de los hexágonos se puede personalizar con el argumento `color` de la función `geom_hex()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex(color = "white")
```



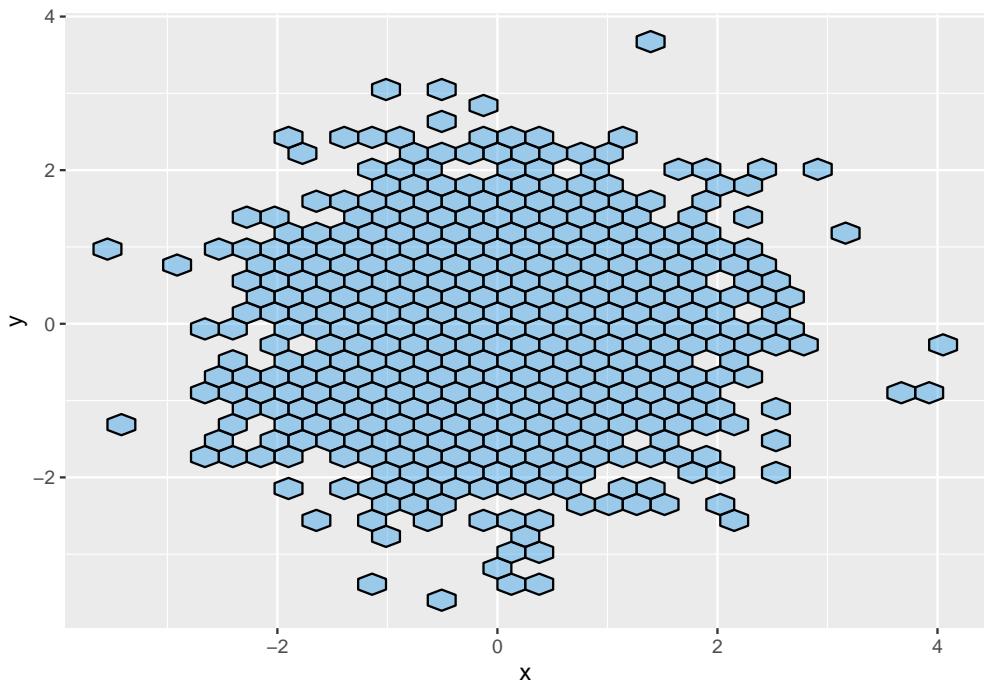
10.6.2.2 COLOR DE FONDO Y TRANSPARENCIA

También se puede establecer un color para el fondo de los hexágonos con el argumento `fill` de la función `'aes()'` y controlar su grado de transparencia con el argumento `alpha`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex(color = 1, fill = 4, alpha = 0.4)
```



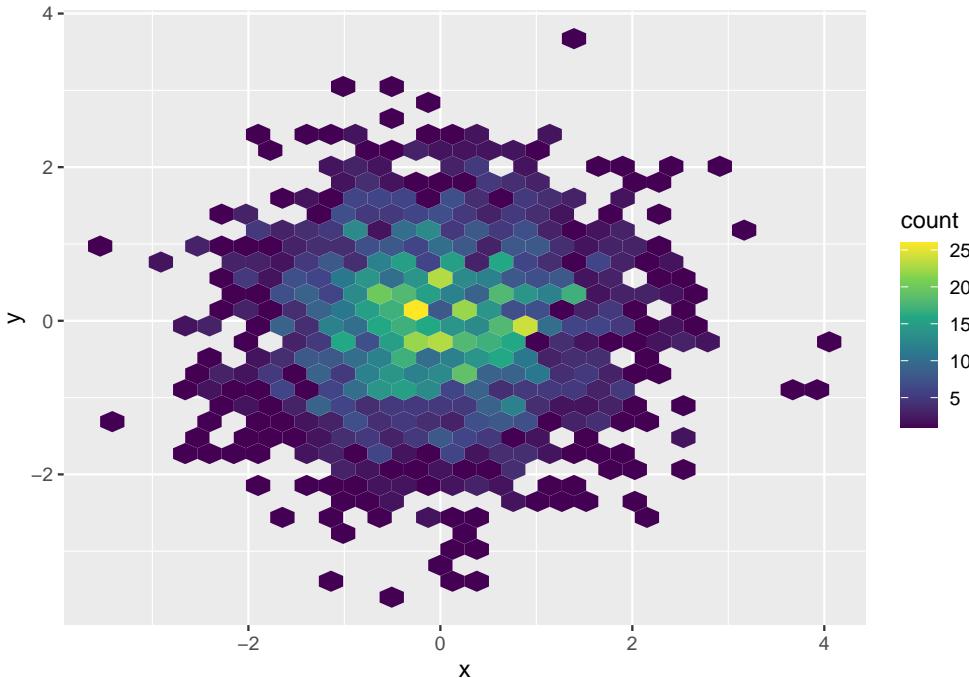
10.6.2.3 PALETA DE COLORES

Si se quiere tener una paleta de colores degradada que represente los valores, se puede cambiar los colores por defecto con las funciones `scale_fill_viridis_c()`, o `scale_fill_gradiente()` o una función familiar.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex() +
  scale_fill_viridis_c()
```



10.6.3 PERSONALIZACIÓN DE LA LEYENDA

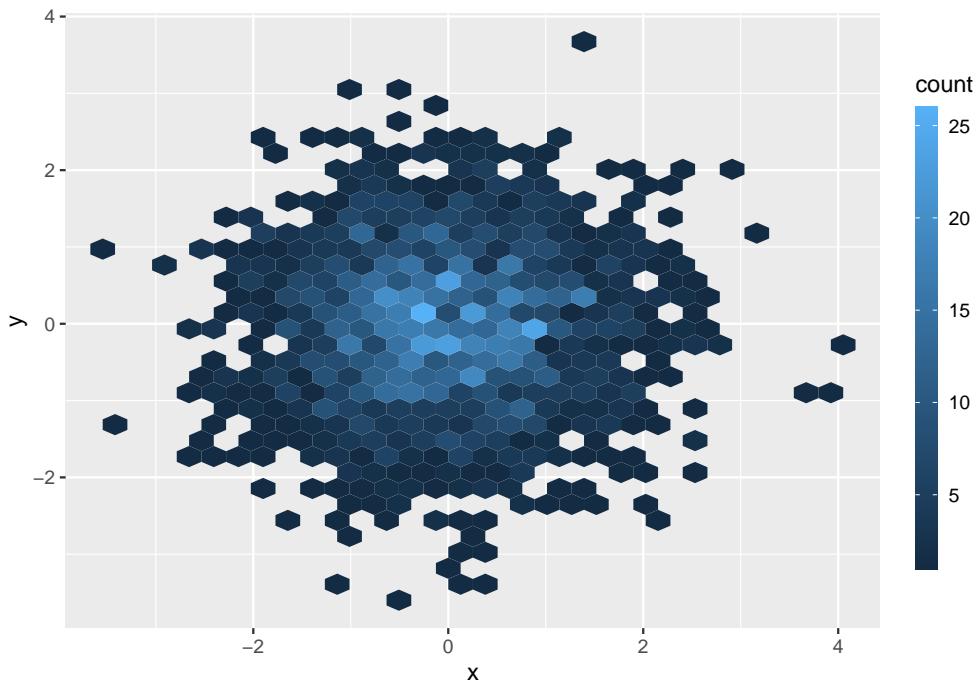
10.6.3.1 ANCHO Y ALTO

Cuando se crea un diagrama hexbin aparecerá una barra de color a modo de leyenda de manera automática. Se puede controlar el ancho y la altura de la barra con los argumentos `barwidth` y `barheight`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex() +
  guides(fill = guide_colourbar(barwidth = 0.7,
                                barheight = 15))
```



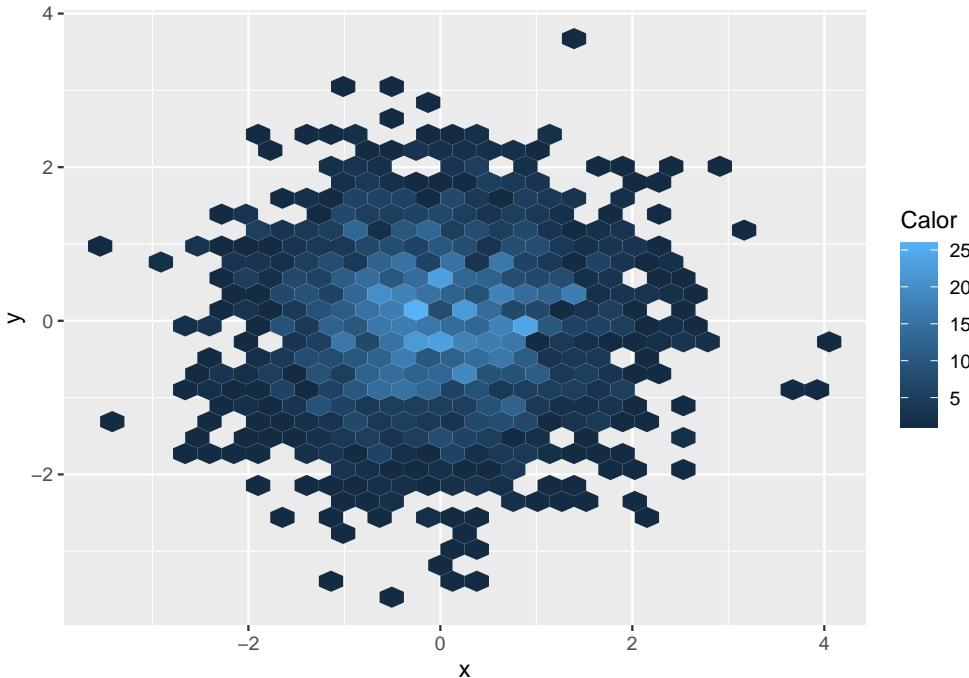
10.6.3.2 CAMBIAR EL TÍTULO

El título por defecto de la leyenda es “*count*”. Se puede cambiar con el argumento `title` de la función `guide_colourbar`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex() +
  guides(fill = guide_colourbar(title = "Calor"))
```



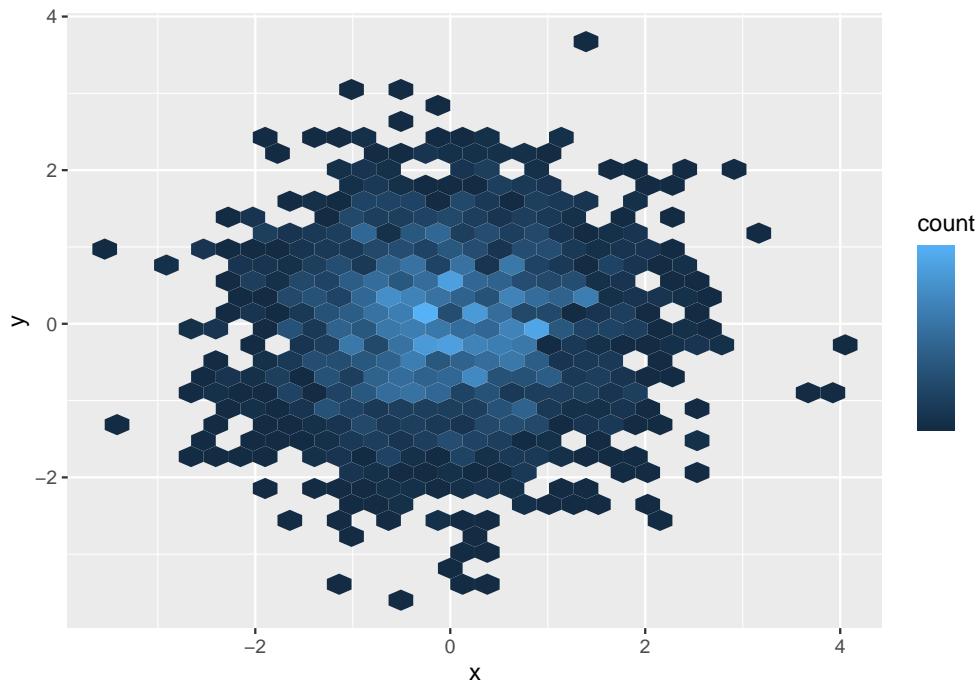
10.6.3.3 ELIMINAR LAS ETIQUETAS Y LAS MARCAS

Hay que tener en cuenta que se puede eliminar las marcas (ticks) y/o las etiquetas estableciendo los argumentos `label` y `ticks` como `FALSE`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex() +
  guides(fill = guide_colourbar(label = FALSE,
                                 ticks = FALSE))
```



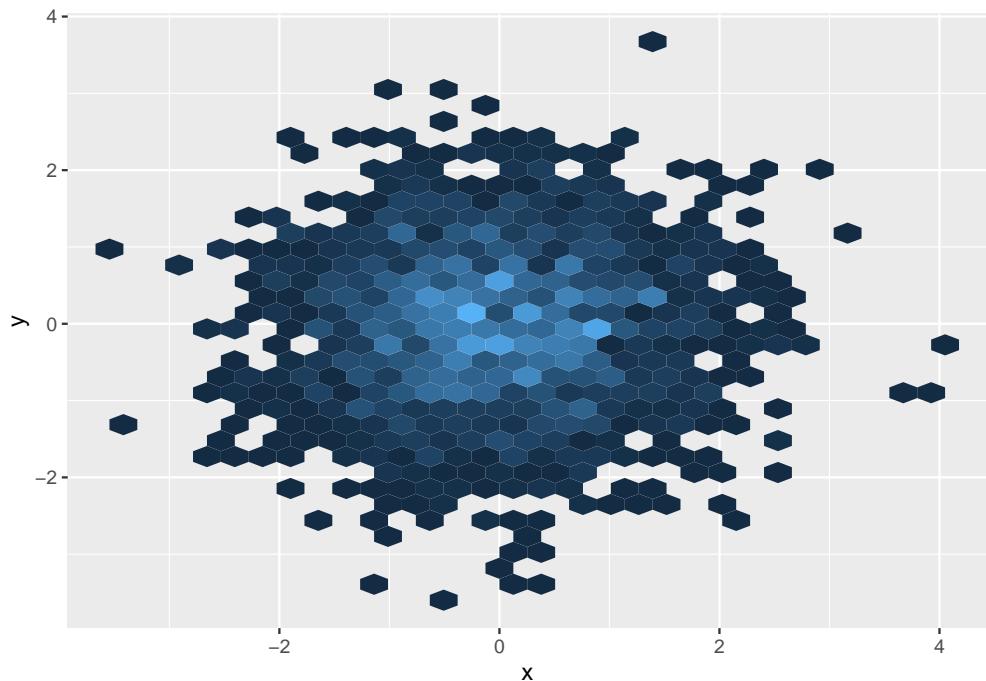
10.6.3.4 ELIMINAR LA LEYENDA

Por último, si se desea deshacer la leyenda, se puede establecer su posición como ““none”“.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(85)
df = data.frame(x = rnorm(2000), y = rnorm(2000))

ggplot(df, aes(x = x, y = y)) +
  geom_hex() +
  theme(legend.position = "none")
```



10.7 CONTOURS EN ggplot2

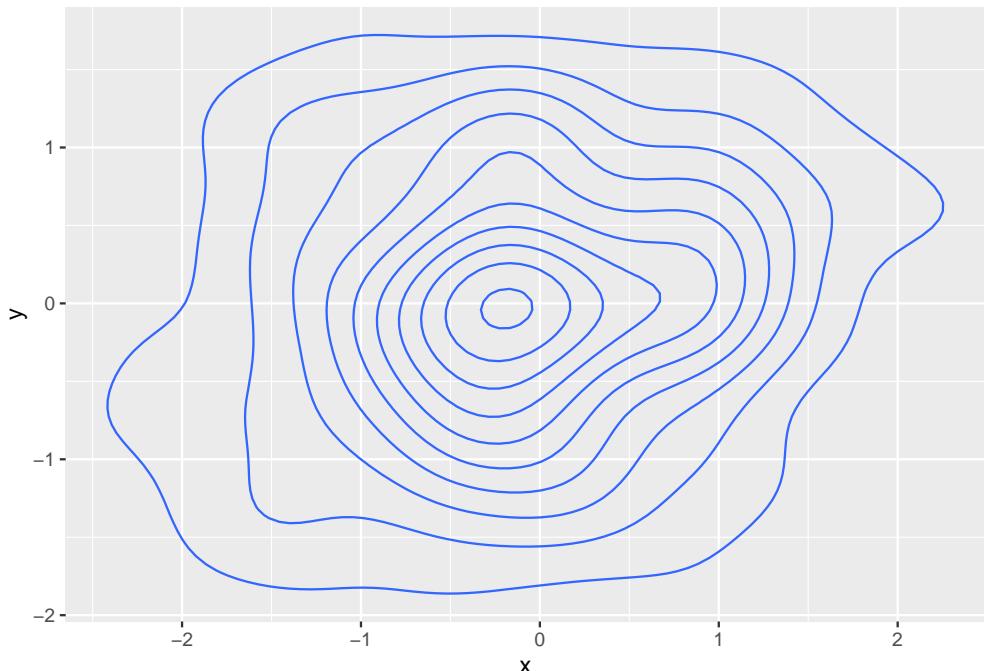
10.7.1 LA FUNCIÓN geom_density_2d()

Se puede crear un gráfico de contorno para una densidad en 2D usando la función `geom_density_2d()` dentro del paquete de `ggplot2`. Para ello solo es necesario pasarle un data frame e indicar dentro de la función `aes()` cuáles son las variables `x` e `y`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(90)
df = data.frame(x = rnorm(200), y = rnorm(200))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d()
```



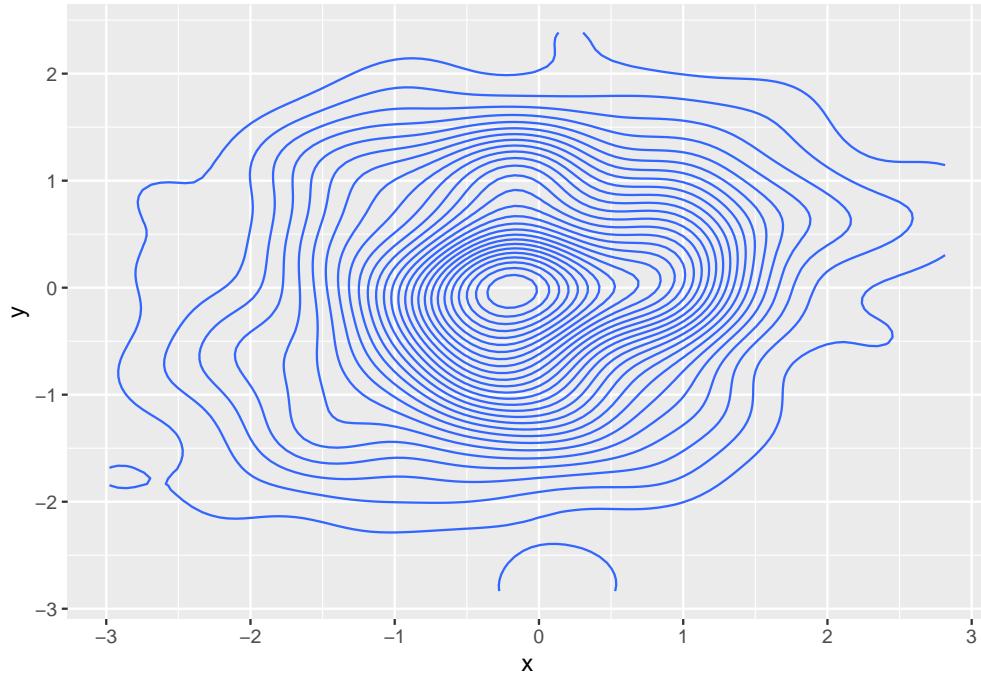
10.7.1.1 MODIFICAR EL NÚMERO DE NIVELES

Es posible incrementar o disminuir el número de niveles con el argumento `bins`.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
# Datos
set.seed(90)
df = data.frame(x = rnorm(200), y = rnorm(200))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d(bins = 30)
```



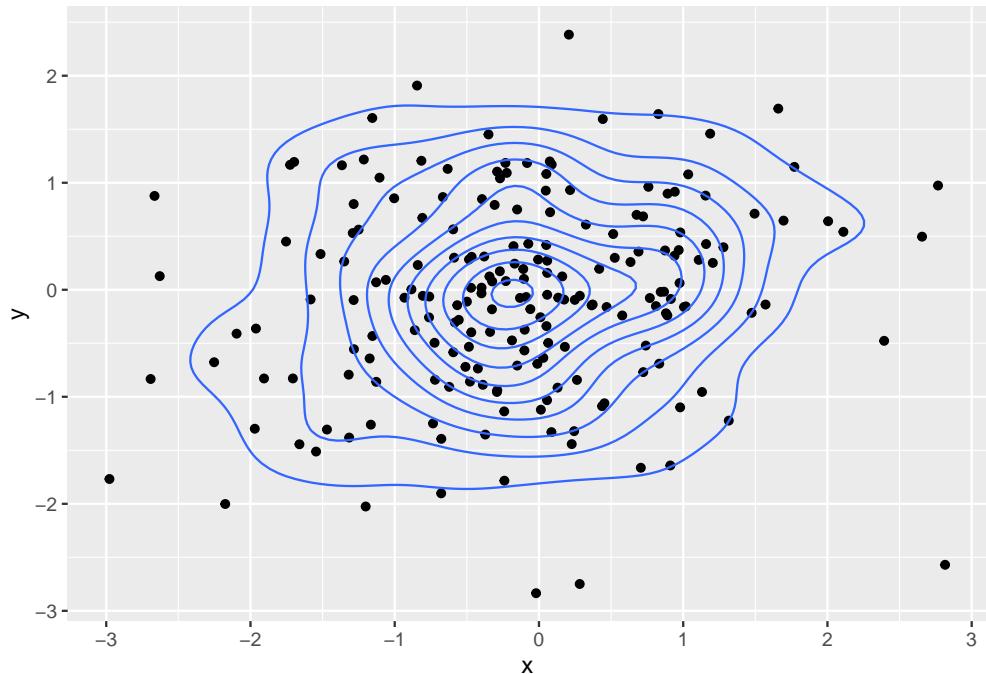
10.7.1.2 GRÁFICO DE DISPERSIÓN CON CURVAS DE NIVEL

También se pueden representar los puntos con la función `geom_point()` y después usar la función `geom_density_2d()` para añadir al gráfico de dispersión sus correspondientes curvas de nivel.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(90)
df = data.frame(x = rnorm(200), y = rnorm(200))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_density_2d()
```



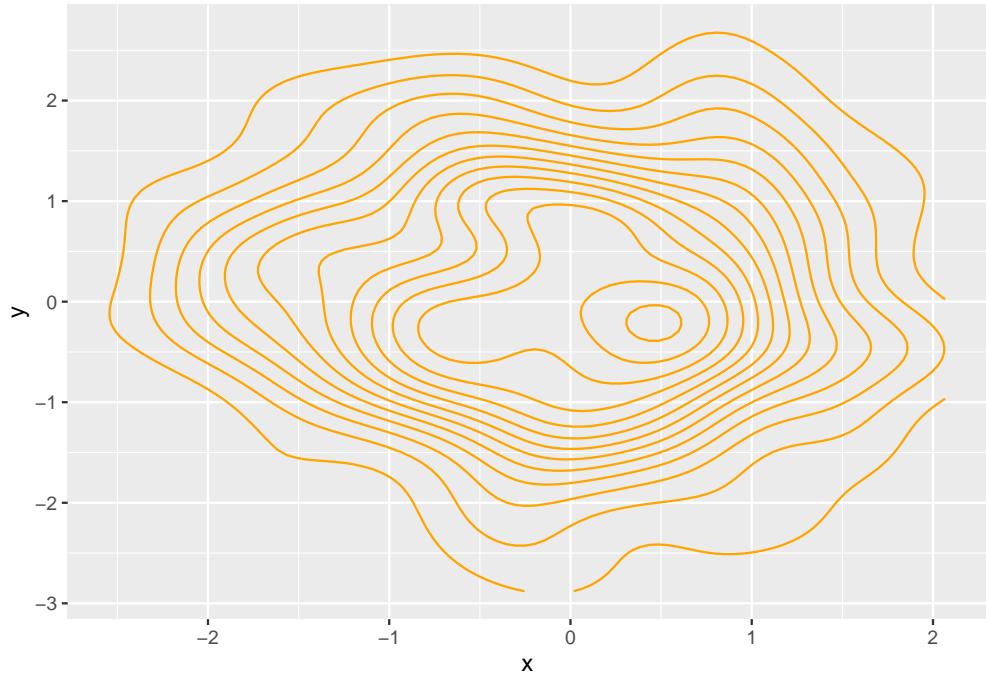
10.7.1.3 PERSONALIZACIÓN DE LAS LÍNEAS

Se pueden personalizar las líneas de contorno de diversas maneras. Al igual que en otro tipo de diagramas, es posible cambiar el color, el grosor o el tipo de línea.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(56)
df = data.frame(x = rnorm(300), y = rnorm(300))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d(color = "orange")
```



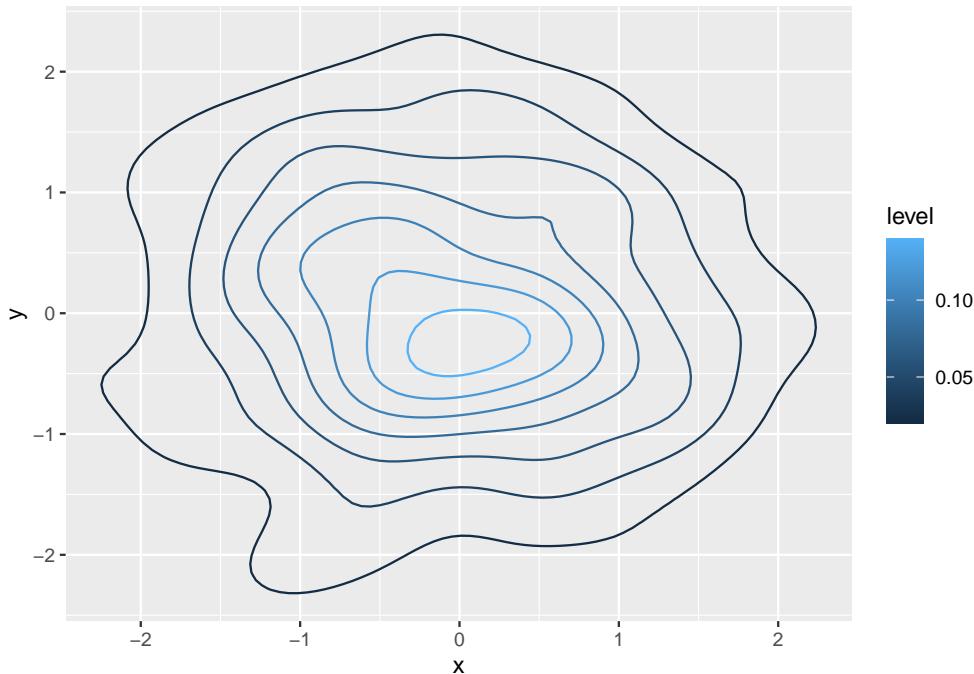
10.7.1.4 COLOR EN FUNCIÓN DEL NIVEL

Pasando el argumento `..level..` dentro del argumento `color` de la función `aes()` se colorearán las curvas en función del nivel. Así se resaltarán zonas de mayor densidad con colores más claros.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(99)
df = data.frame(x = rnorm(450), y = rnorm(450))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d(aes(color = ..level..))
```



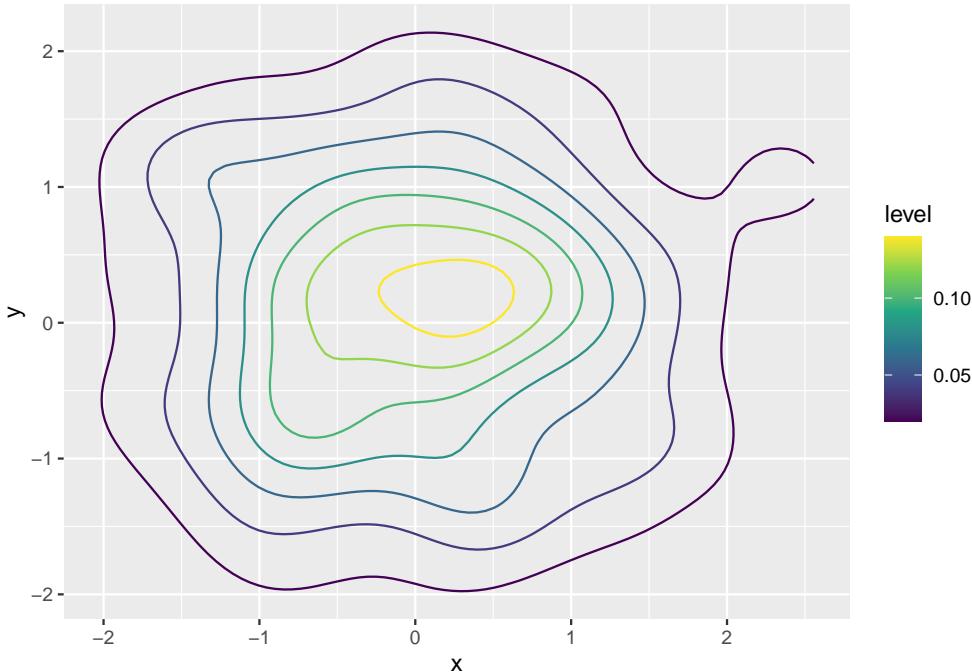
10.7.1.5 CAMBIAR LA PALETA DE COLORES

Si las líneas se van a pintar en función del nivel, se puede cambiar el color por defecto por una paleta de colores continua. Por ejemplo, para usar la paleta famosa **viridis** basta con añadir la función `scale_color_viridis_c()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(88)
df = data.frame(x = rnorm(199), y = rnorm(199))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d(aes(color = ..level..)) +
  scale_color_viridis_c()
```



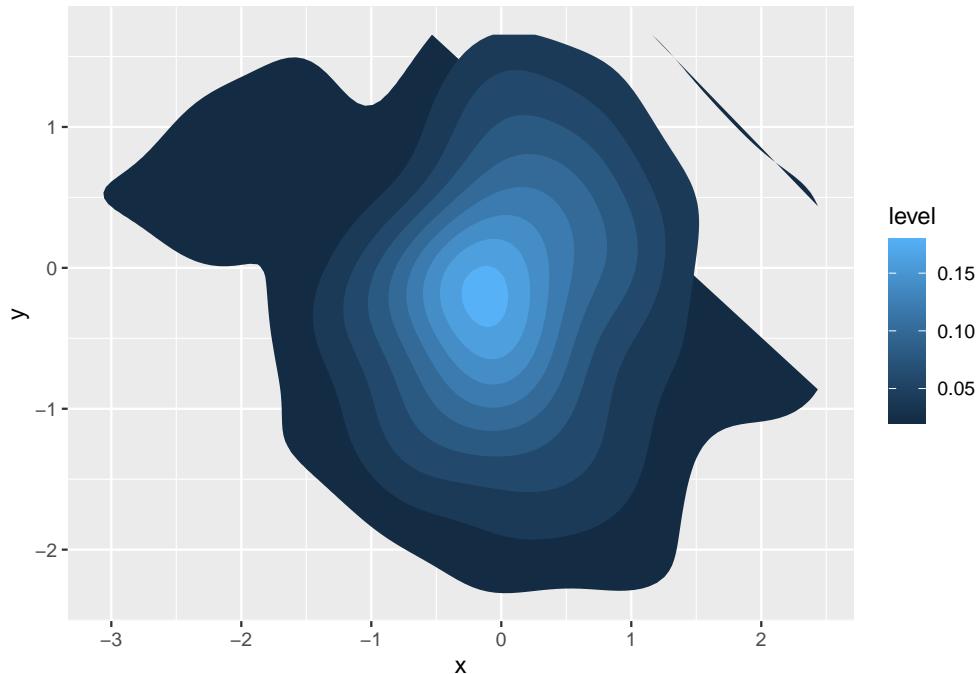
10.7.1.6 COLOREAR EL CONTOUR CON `stat_density_2d()`

Existe una función similar llamada `stat_density_2d()` que permite cambiar el geom. Si se establece el argumento `geom = "polygon"` se puede colorear el contorno como se muestra a continuación.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(77)
df = data.frame(x = rnorm(99), y = rnorm(99))

ggplot(df, aes(x = x, y = y, fill = ..level..)) +
  stat_density_2d(geom = "polygon")
```



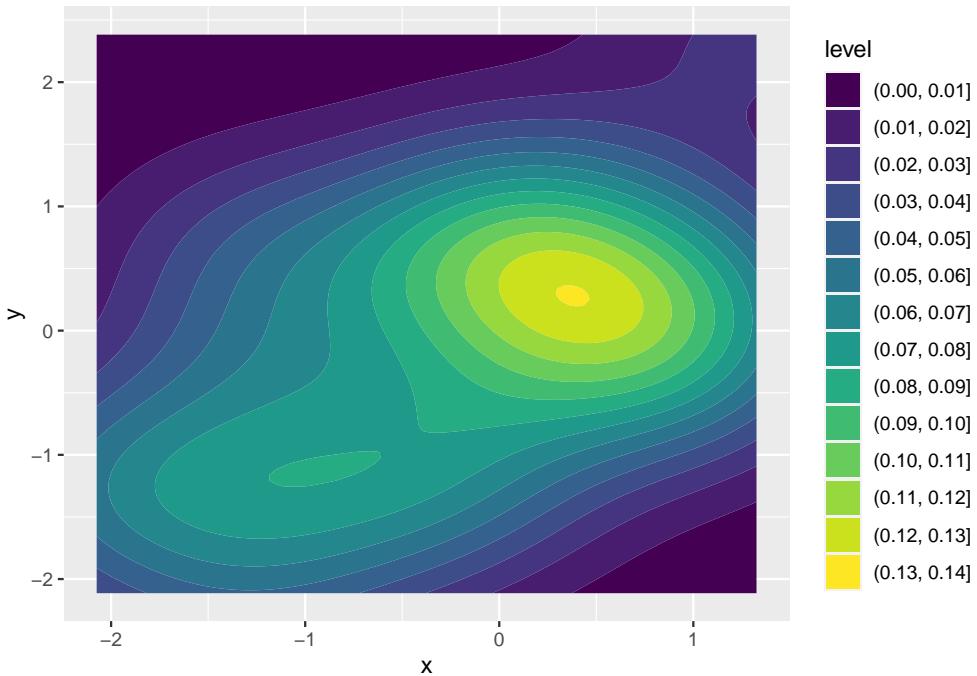
10.7.2 LA FUNCIÓN `geom_density_2d_filled()`

Para representar los niveles mediante áreas en lugar de líneas se puede usar la función `geom_density_2d_filled()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(999)
df = data.frame(x = rnorm(20), y = rnorm(20))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d_filled()
```



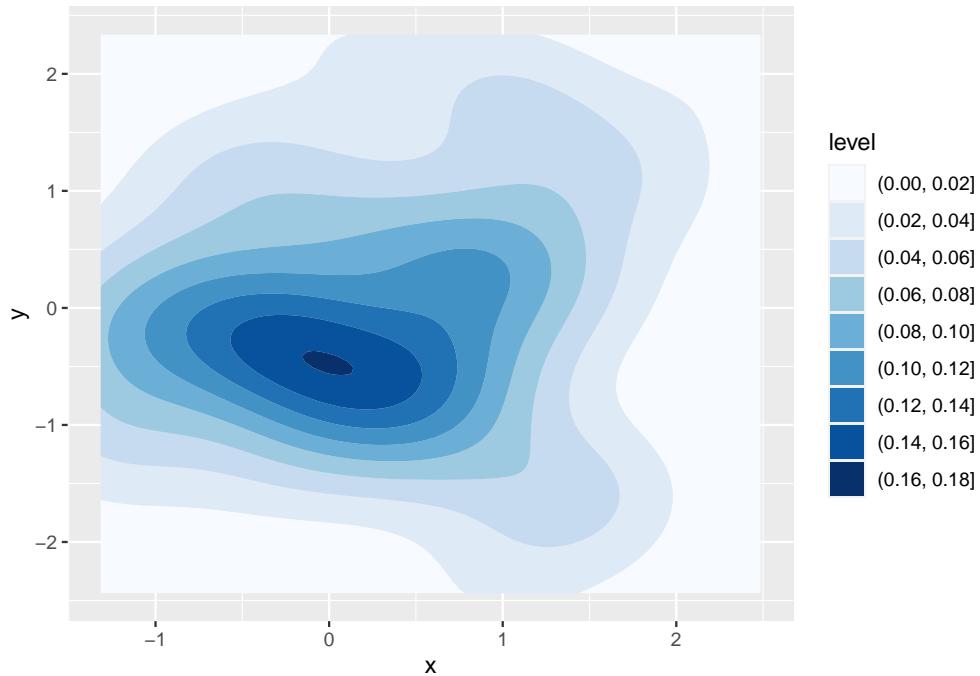
10.7.2.1 PALETA DE COLORES

De nuevo, se pueden cambiar los colores de relleno usando una paleta de colores discreta como `scale_fill_brewer()` o la función `scale_fill_manual()`, esta última permite elegir los colores personalizados.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(15)
df = data.frame(x = rnorm(50), y = rnorm(50))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d_filled() +
  scale_fill_brewer()
```



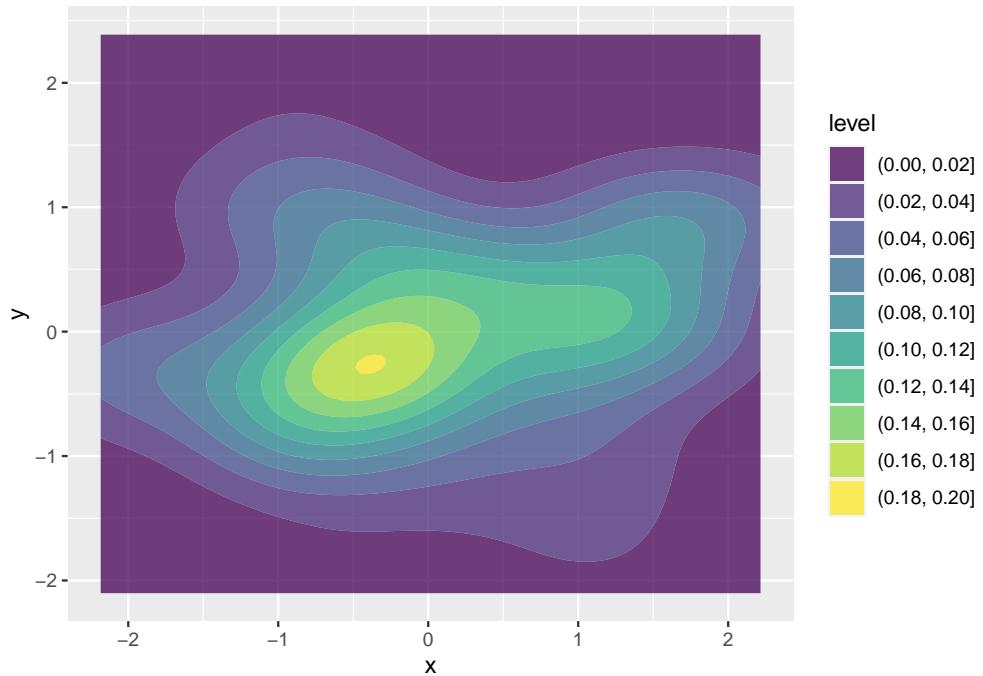
10.7.2.2 TRANSPARENCIA

El argumento `alpha` permite modificar la transparencia de las áreas coloreadas.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(5)
df = data.frame(x = rnorm(50), y = rnorm(50))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d_filled(alpha = 0.75)
```



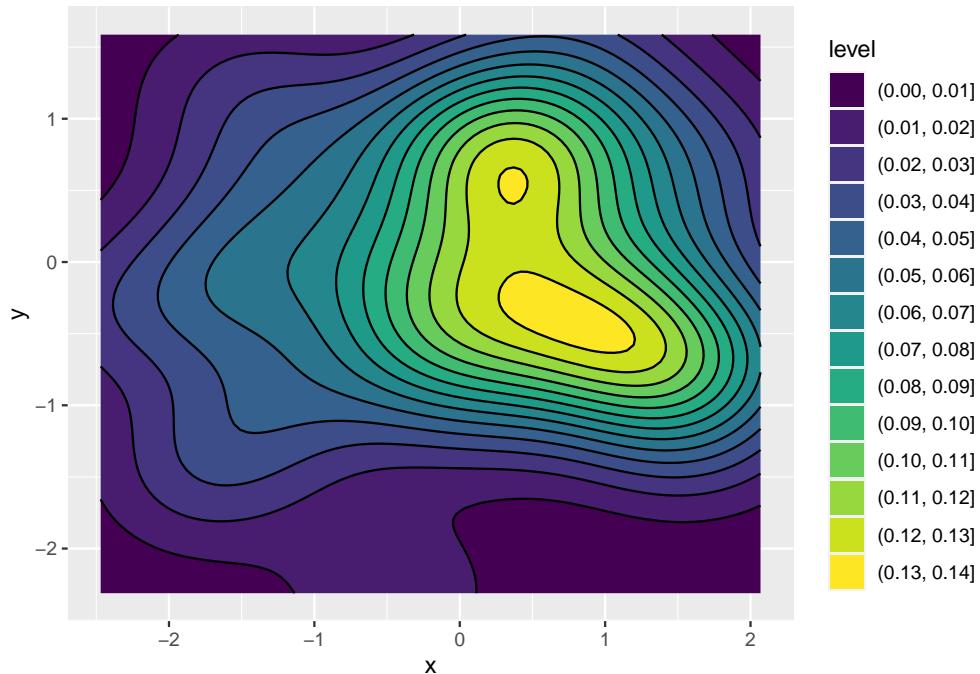
10.7.2.3 LÍNEAS DE CONTORNO SOBRE ÁREAS

También se puede añadir las líneas de contorno sobre las áreas que delimitan.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(-100)
df = data.frame(x = rnorm(40), y = rnorm(40))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d_filled() +
  geom_density_2d(colour = "black")
```



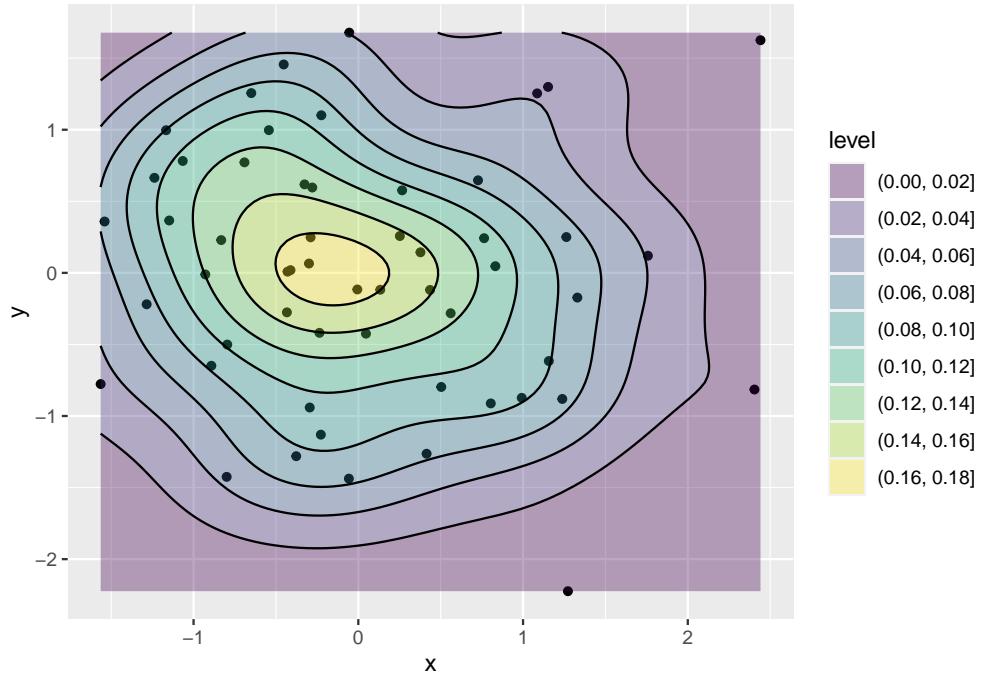
10.7.2.4 ÁREAS Y LÍNEAS DE CONTOURNO SOBRE EL DIAGRAMA DE PUNTOS

Cabe destacar que también es posible añadir las áreas y curvas de nivel a un diagrama de dispersión. Para ello basta con fijar un valor de transparencia bajo.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(0)
df = data.frame(x = rnorm(55), y = rnorm(55))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_density_2d_filled(alpha = 0.35) +
  geom_density_2d(colour = "black")
```



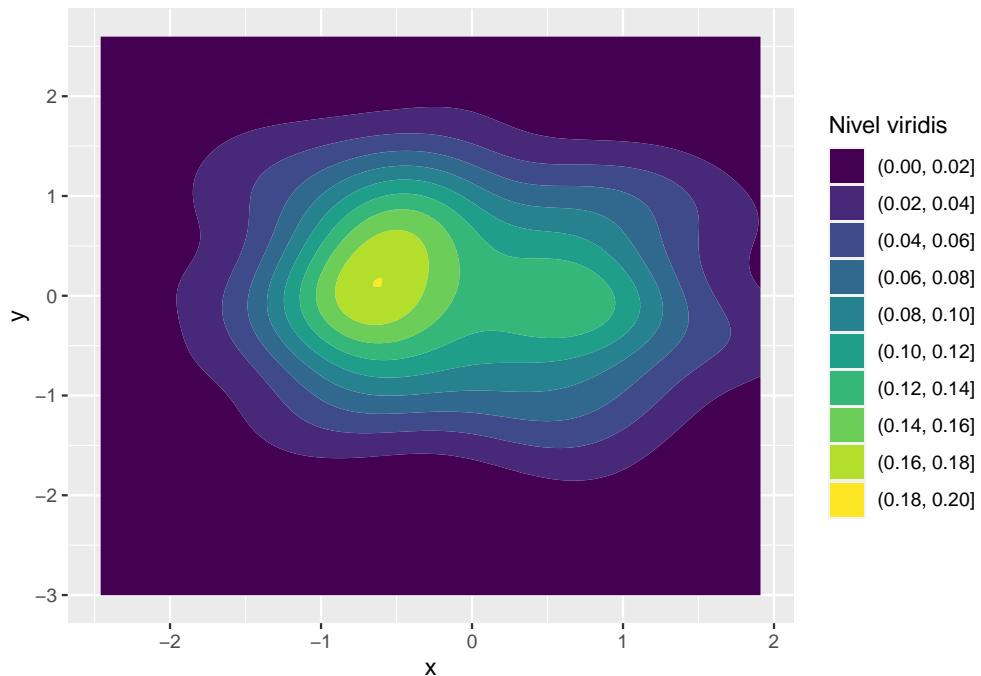
10.7.2.5 TÍTULO DE LA LEYENDA

La leyenda de los diagramas de contorno también se puede personalizar. En el siguiente ejemplo se cambia el título de la leyenda.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(1313)
df = data.frame(x = rnorm(80), y = rnorm(80))

ggplot(df, aes(x = x, y = y)) +
  geom_density_2d_filled() +
  guides(fill = guide_legend(title = "Nivel viridis"))
```



CAPÍTULO 11: DIAGRAMAS DE EVOLUCIÓN CON `ggplot2`



Los diagramas de evolución muestran cómo evoluciona una variable o conjunto de variables, generalmente a través del tiempo.

11.1 STREAMGRAPH CON `ggplot2`

11.1.1 DATOS DE MUESTRA

Considere el conjunto de datos `blockbuster` del paquete `ggstream` para esta parte del manual.

```
# install.packages("remotes")
# remotes::install_github("davisjoberg/ggstream")
```

```
library(ggstream)

blockbusters %>%
head(10) %>%
kable(booktabs = TRUE, format = "latex") %>%
kable_styling(
  latex_options = c("striped", "condensed", "HOLD_position"),
  position = "center",
  full_width = FALSE
)
```

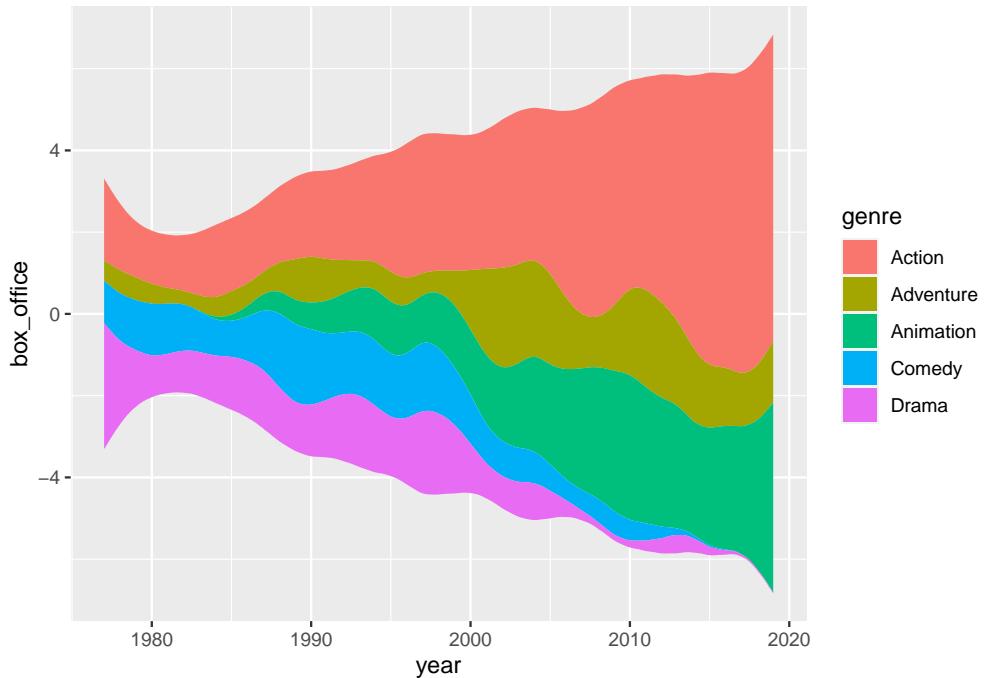
year	genre	box_office
1977	Action	2.9835206
1977	Adventure	0.2087800
1977	Comedy	0.5164815
1977	Drama	2.5397649
1978	Action	1.9151704
1978	Adventure	0.7598255
1978	Comedy	1.0386010
1978	Drama	0.2024703
1979	Action	1.1479297
1979	Adventure	0.3118831

11.1.2 STREAM PLOT EN `ggplot2` CON LA FUNCIÓN `geom_stream()`

La función `geom_stream()` del paquete `ggstream` permite crear streamplots en `ggplot2`. Pasando el dataframe y usando la función para generar un diagrama básico:

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

# Se genera el más básico
ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream()
```

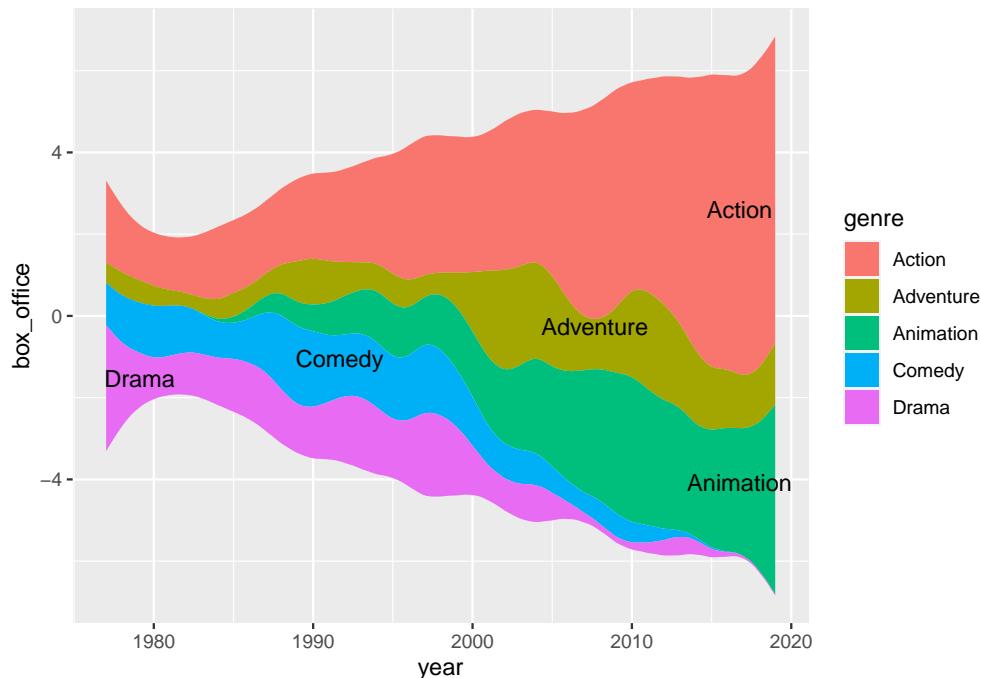


11.1.2.1 ETIQUETANDO LAS ÁREAS

Hay que tener en cuenta que la librería contiene una función adicional llamada `geom_stream_label()` que puede ser usada para agregar etiquetas a cada área del streamgraph.

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream() +
  geom_stream_label(aes(label = genre))
```



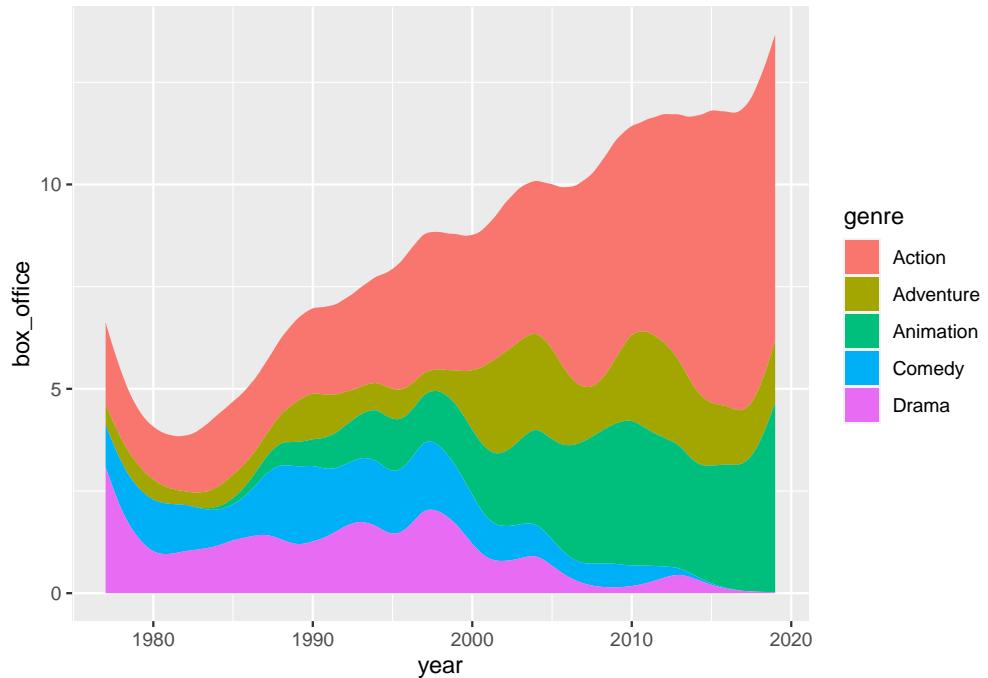
11.1.3 TIPOS DE STREAM GRAPHS

Por defecto, la función `geom_stream()` usa el tipo "mirror", que apila las áreas de manera simétrica respecto al eje X. Otras alternativas son "ridge", que apila desde el eje X y "proportional", que hace que las áreas sume 1.

11.1.3.1 RIDGE

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

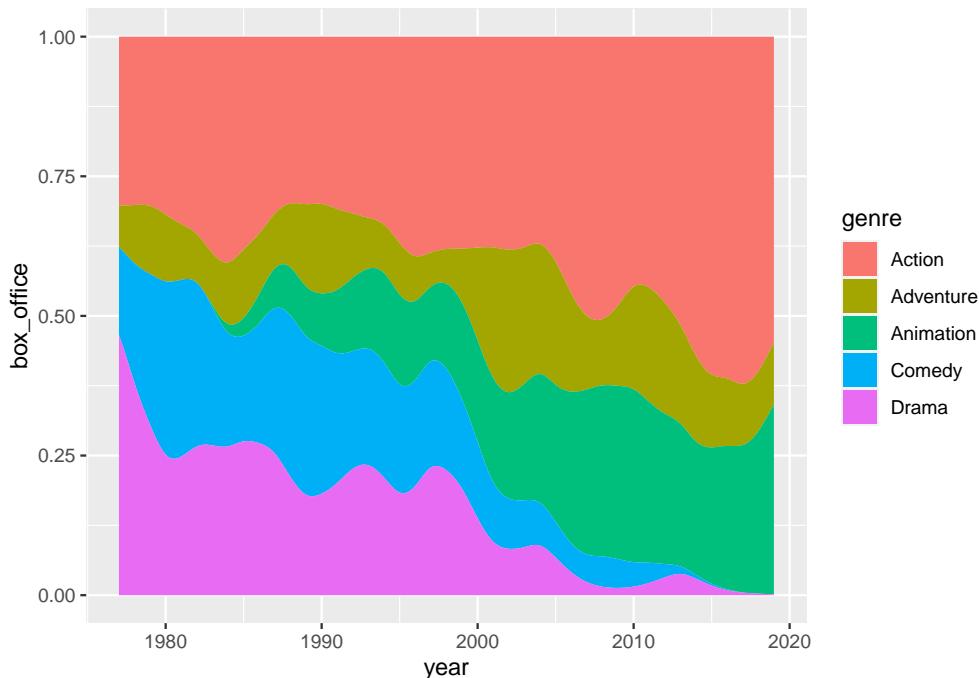
ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(type = "ridge")
```



11.1.3.2 PROPORTIONAL

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(type = "proportional")
```



11.1.4 PERSONALIZACIÓN DE LOS COLORES

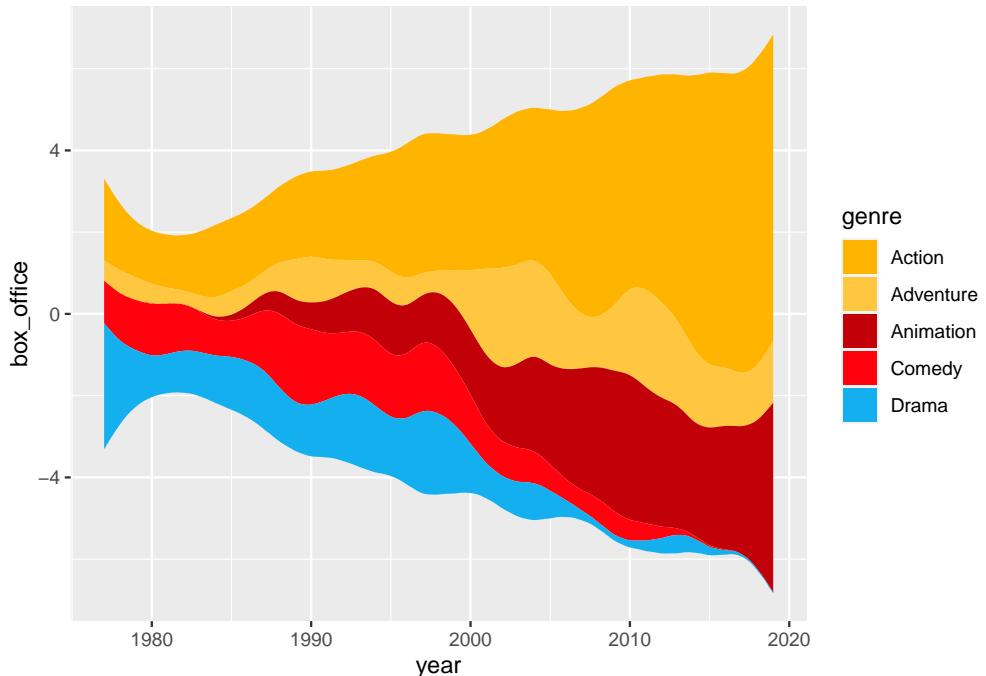
11.1.4.1 CAMBIAR LOS COLORES DE FONDO

Es posible cambiar los colores de fondo del streamgraph con la función `scale_fill_manual()` o una función equivalente.

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AFEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream() +
  scale_fill_manual(values = cols)
```



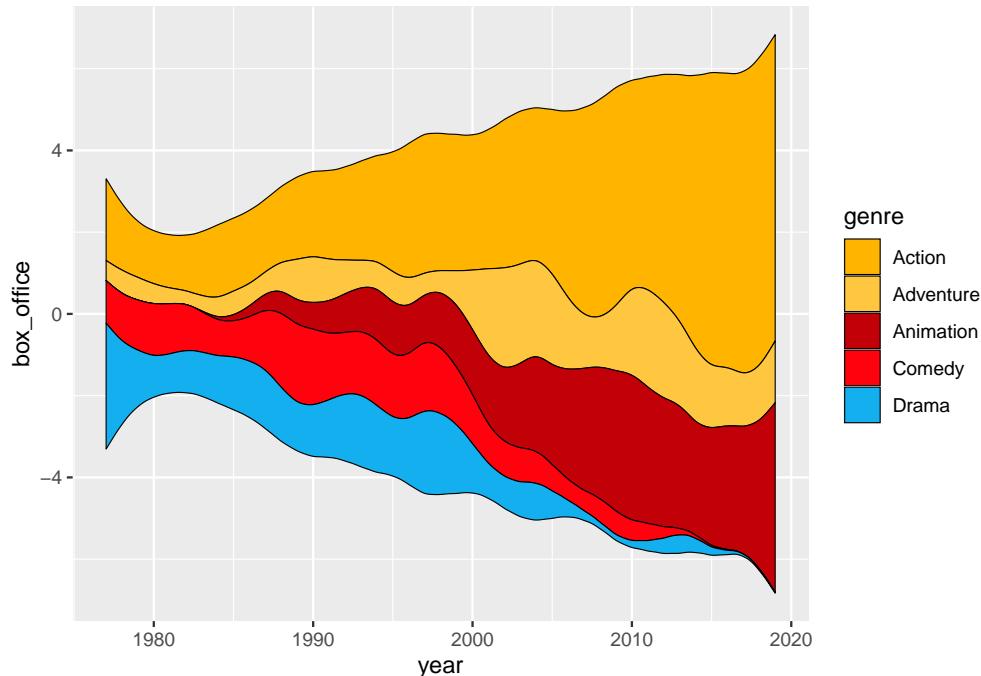
11.1.4.2 CAMBIAR EL COLOR DE LOS BORDES

Por defecto las áreas no tienen bordes pero se puede agregar pasando un color al argumento `color =` de la función `geom_stream()` y modificar su grosor con `lwd =`.

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AFEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(color = 1, lwd = 0.25) +
  scale_fill_manual(values = cols)
```



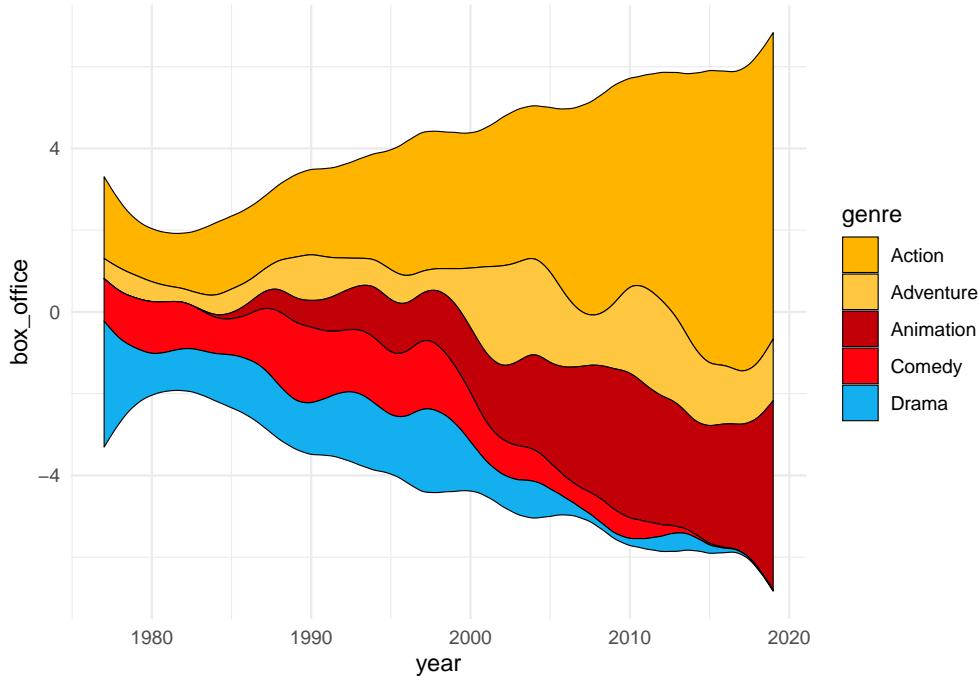
11.1.4.3 CAMBIAR EL TEMA

Hay que tener en cuenta que también se puede cambiar el tema para modificar la apariencia del diagrama.

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(color = 1, lwd = 0.25) +
  scale_fill_manual(values = cols) +
  theme_minimal()
```



11.1.5 ARGUMENTOS AVANZADOS

Para crear un streamplot es necesario realizar estimaciones no paramétricas que necesitan de una **ventana** (por defecto de 0.75) y una **rejilla de valores** en el eje X (por defecto es 1000). Se pueden cambiar estos valores por defecto con `bw` = y `n_grid` =.

Hay que tener en cuenta que también se puede usar un **rango adicional** en la estimación con `extra_span` =, que por defecto es 0.01 (1%) y especificar con `true_range` = si el rango verdadero de los datos se debería de usar en la visualización o si por el contrario se debería de usar el rango de estimación.

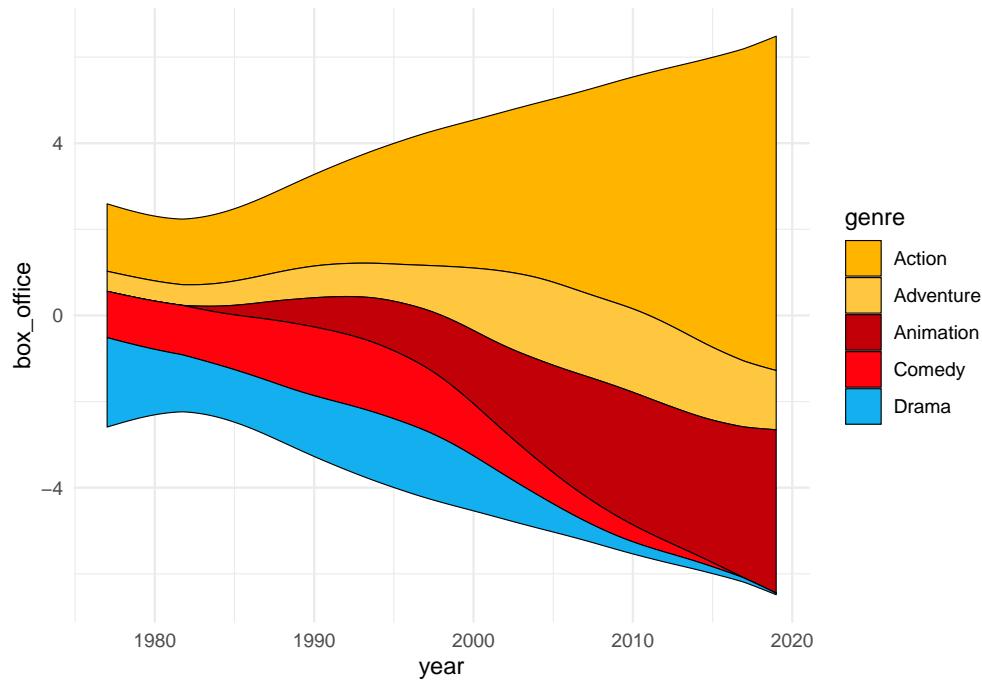
11.1.5.1 VENTANA (bw)

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(color = 1, lwd = 0.25,
             bw = 1) +
```

```
scale_fill_manual(values = cols) +
theme_minimal()
```

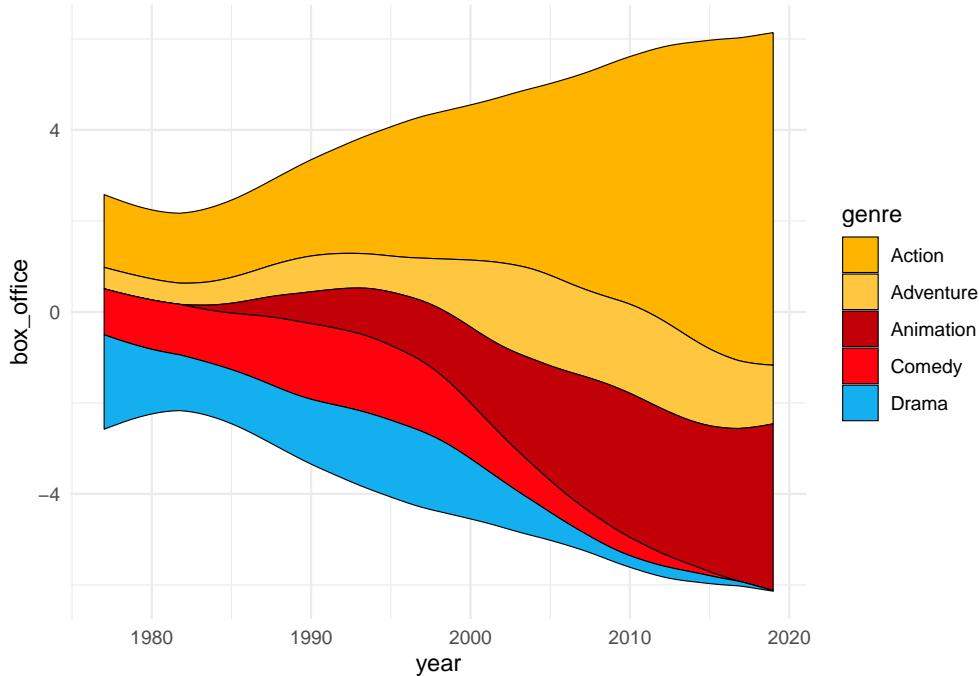


11.1.5.2 (n_grid)

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(color = 1, lwd = 0.25,
             n_grid = 100) +
  scale_fill_manual(values = cols) +
  theme_minimal()
```

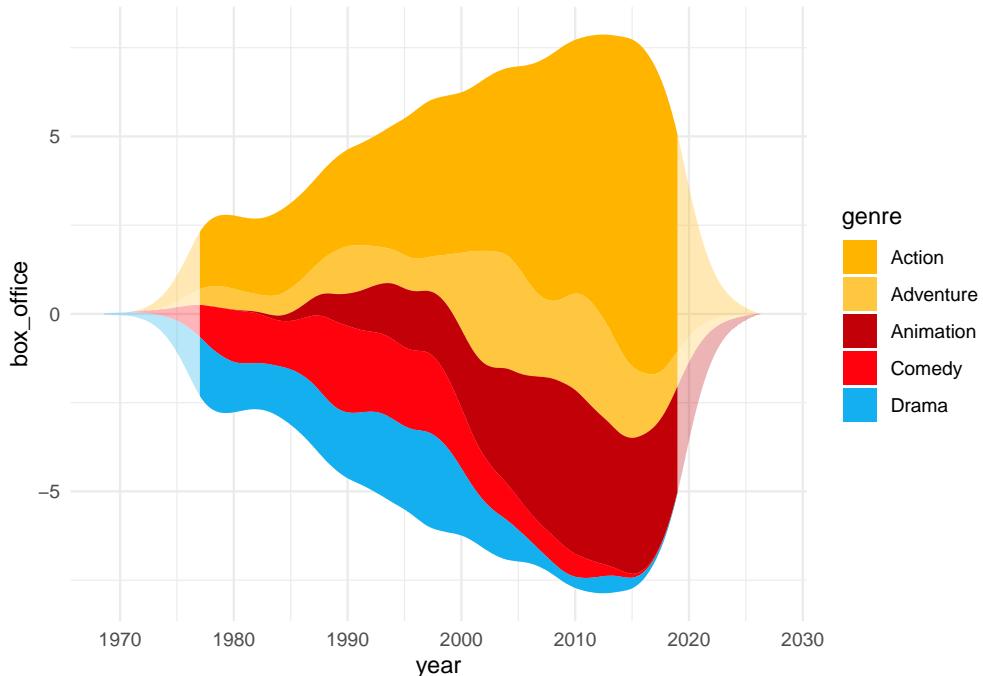


11.1.5.3 (extra_span, true_range)

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(extra_span = 0.2) +
  geom_stream(extra_span = 0.2, true_range = "none",
             alpha = 0.3) +
  scale_fill_manual(values = cols) +
  theme_minimal()
```



11.1.6 PERSONALIZACIÓN DE LA LEYENDA

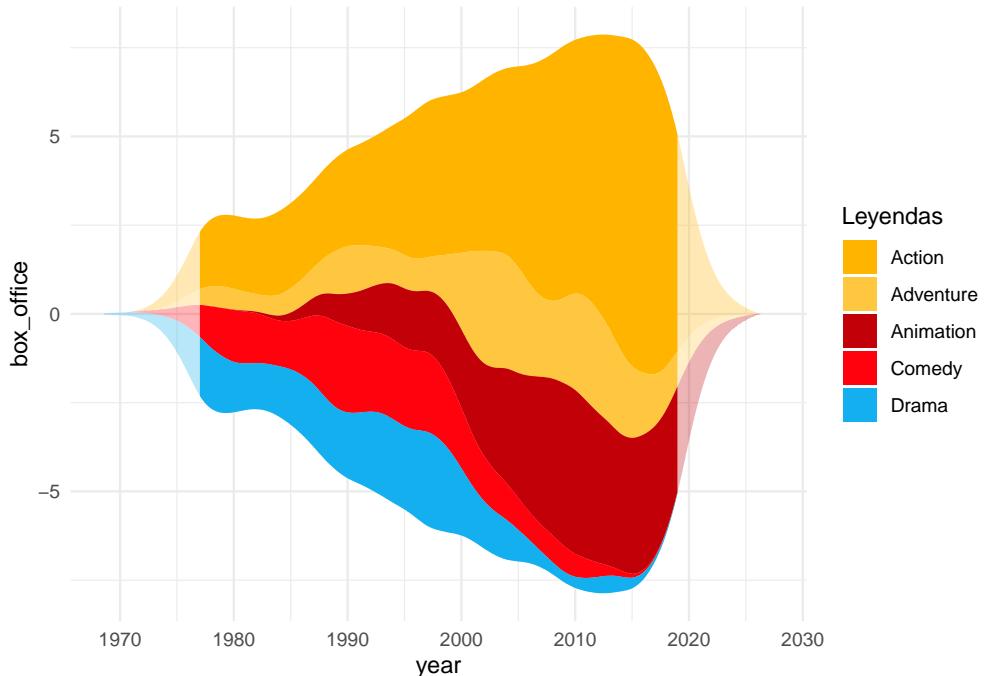
Como en otros diagramas de ggplot2 se puede cambiar el título, las etiquetas y eliminar o cambiar la posición de la leyenda, tal y como se muestra en los siguientes ejemplos:

11.1.6.1 CAMBIAR EL TÍTULO

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AFEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(extra_span = 0.2) +
  geom_stream(extra_span = 0.2, true_range = "none",
             alpha = 0.3) +
  scale_fill_manual(values = cols) +
  theme_minimal() +
  guides(fill = guide_legend(title = "Leyendas"))
```

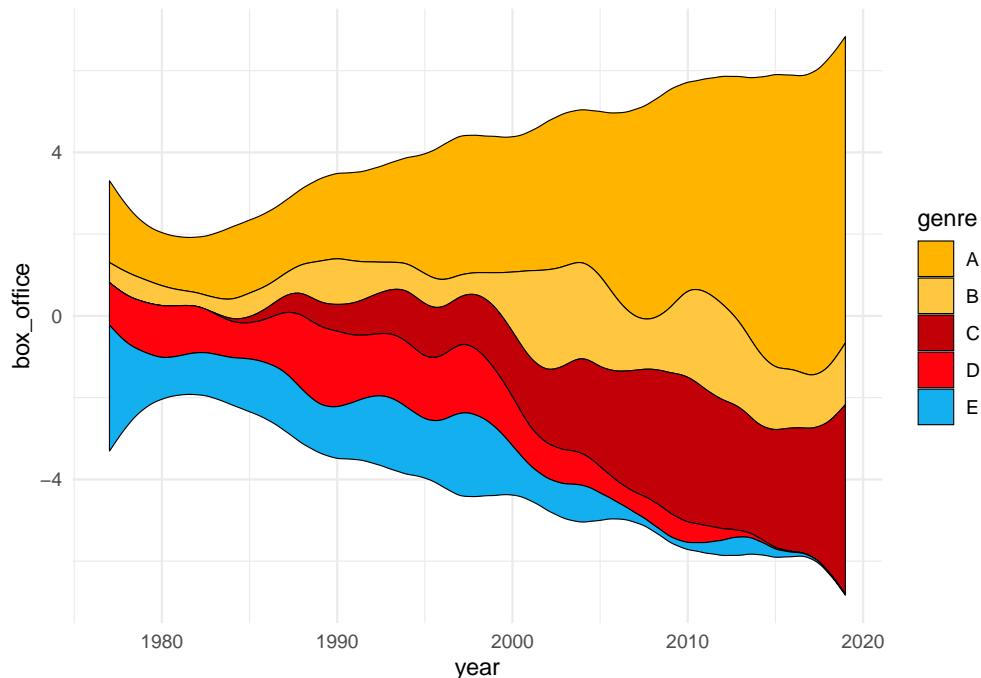


11.1.6.2 CAMBIAR LAS ETIQUETAS

```
# install.packages("remotes")
# remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
# install.packages("ggplot2")
library(ggplot2)

cols = c("#FFB400", "#FFC740", "#C20008", "#FF020D", "#13AEF")

ggplot(blockbusters, aes(x = year, y = box_office, fill = genre)) +
  geom_stream(color = 1, lwd = 0.25) +
  scale_fill_manual(values = cols, labels = LETTERS[1:5]) +
  theme_minimal()
```



11.2 DIAGRAMAS DE ÁREAS (AREA CHART) EN ggplot2 CON geom_area()

11.2.1 CONJUNTO DE DATOS DE MUESTRA

Hay que considerar el siguiente data frame que contiene el precio de cierre de tres índices bursátiles europeos.

```
df = as.data.frame(EuStockMarkets[, 1:3])  
  
df %>%  
  head(10) %>%  
  kable(booktabs = TRUE, format = "latex") %>%  
  kable_styling(  
    latex_options = c("striped", "condensed", "HOLD_position"),  
    position = "center", full_width = FALSE  
)
```

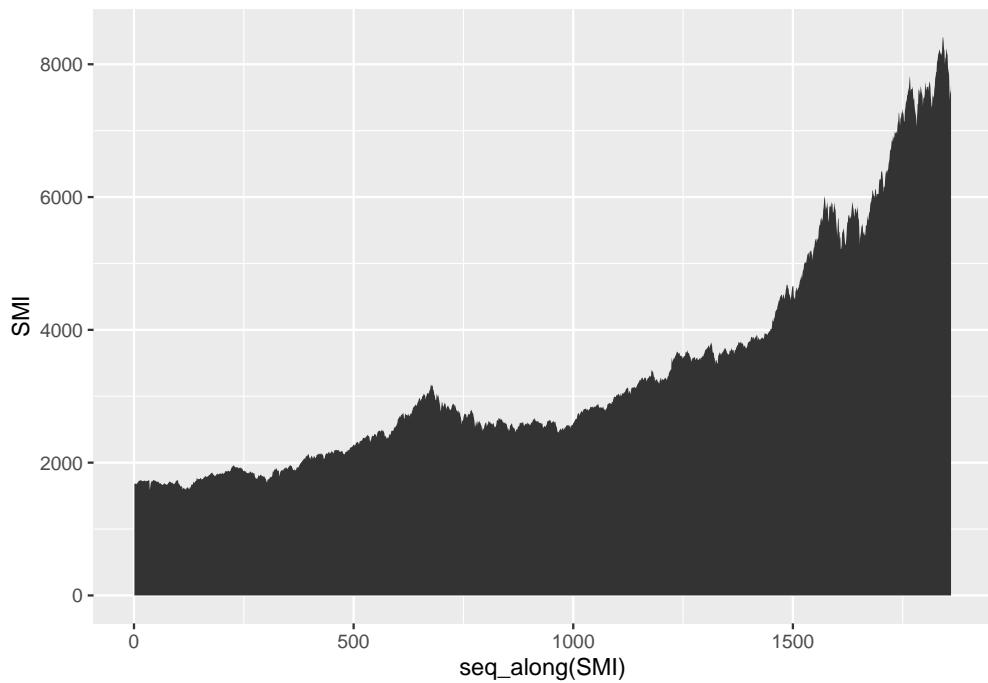
	DAX	SMI	CAC
1	1628.75	1678.1	1772.8
2	1613.63	1688.5	1750.5
3	1606.51	1678.6	1718.0
4	1621.04	1684.1	1708.1
5	1618.16	1686.6	1723.1
6	1610.61	1671.6	1714.3
7	1630.75	1682.9	1734.5
8	1640.17	1703.6	1757.4
9	1635.47	1697.5	1754.0
10	1645.89	1716.3	1754.3

11.2.2 DIAGRAMA DE ÁREA DE UNA ÚNICA LÍNEA

Para crear un diagrama de área de una columna del data frame, se puede pasar las fechas (si están disponibles), o un índice a `x` y la variable `y` y usar `geom_area()`.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
# Gráfico de área  
ggplot(df, aes(x = seq_along(SMI), y = SMI)) +
```

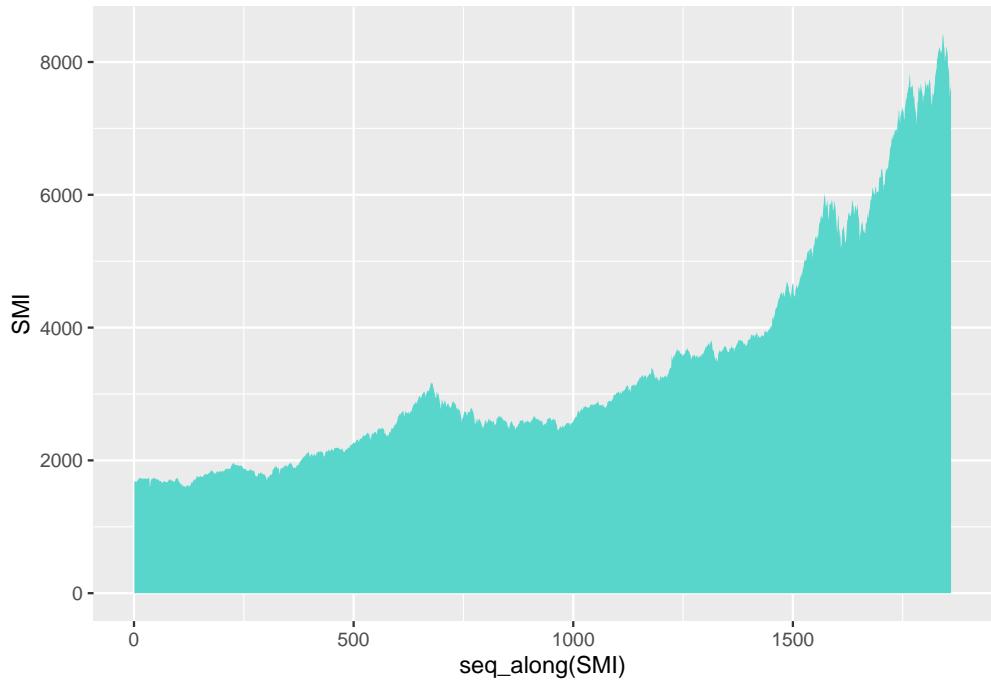
```
geom_area() +  
theme_grey()
```



11.2.2.1 COLOR DEL ÁREA

Por defecto, el área serpia de un color gris muy oscuro. Sin embargo, se puede cambiar pasando un color al argumento `fill =`.

```
# install.packages("ggplot2")  
library(ggplot2)  
  
# Gráfico de área  
ggplot(df, aes(x = seq_along(SMI), y = SMI)) +  
  geom_area(fill = "#58D6CC") +  
  theme_grey()
```

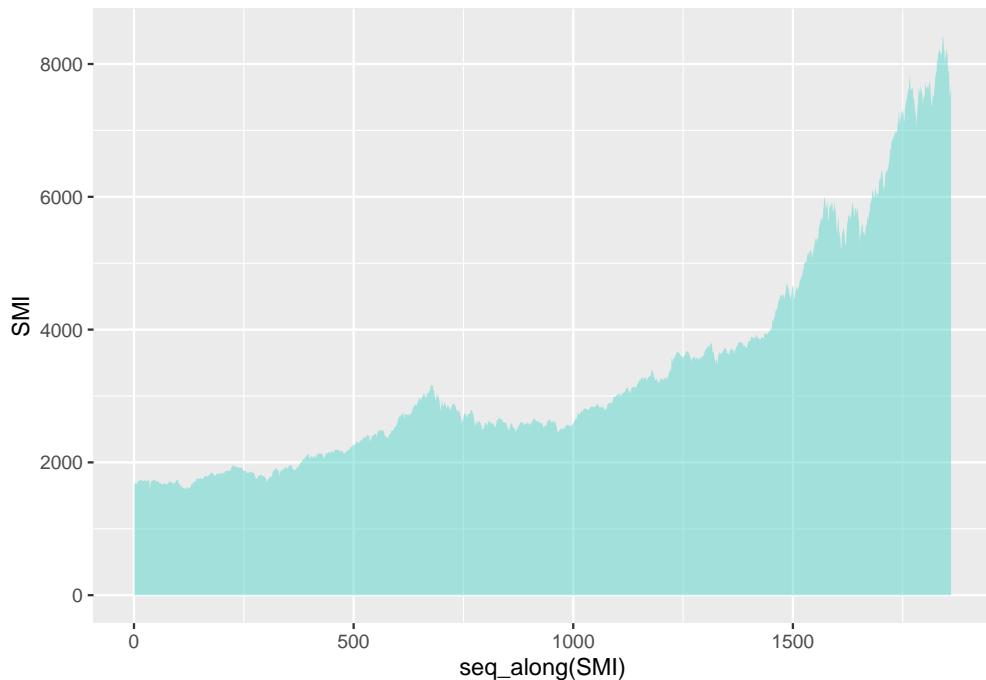


11.2.2.2 TRANSPARENCIA DEL ÁREA

También se puede cambiar la transparencia del área con el argumento `alpha =` de la función.

```
# install.packages("ggplot2")
library(ggplot2)

# Gráfico de área
ggplot(df, aes(x = seq_along(SMI), y = SMI)) +
  geom_area(fill = "#58D6CC",
            alpha = 0.5) +
  theme_grey()
```

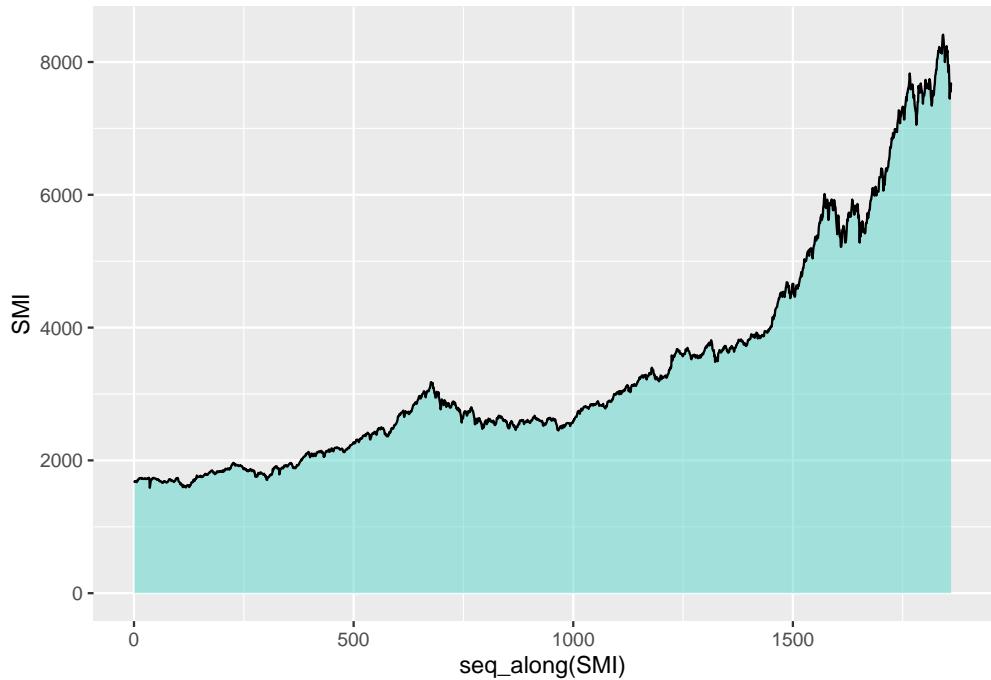


11.2.2.3 PERSONALIZAR LA LÍNEA

La línea superior del área se puede personalizar con varios argumentos, como lo son: `color` =, `lwd` =, o `linetype` =.

```
# install.packages("ggplot2")
library(ggplot2)

# Gráfico de área
ggplot(df, aes(x = seq_along(SMI), y = SMI)) +
  geom_area(fill = "#58D6CC",
            alpha = 0.5,
            color = 1,      # Color de la línea
            lwd = 0.5,      # Ancho de la línea
            linetype = 1) + # Tipo de línea
  theme_grey()
```

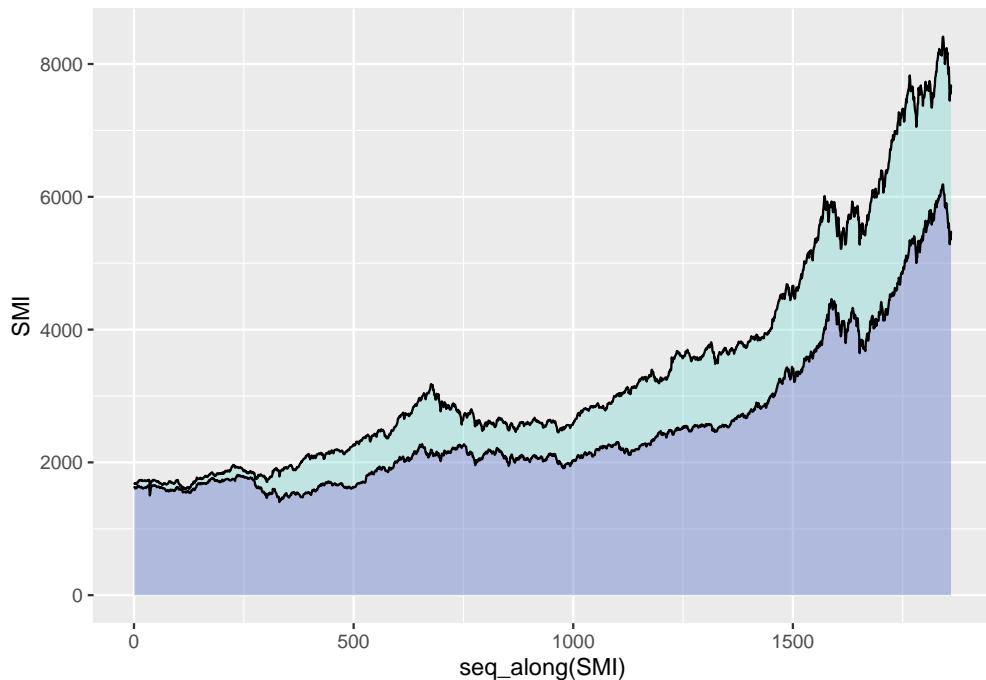


11.2.3 DIAGRAMA DE ÁREAS DE VARIAS LÍNEAS

En caso de que se quiera añadir más de una variable, se tendrá que especificar la estética dentro de cada `geom_area()` para cada variable. Hay que tener en cuenta que si el número de áreas a añadir es superior a dos o tres, se debería de considerar crear un diagrama de áreas apiladas.

```
# install.packages("ggplot2")
library(ggplot2)

# Gráfico de área
ggplot(df) +
  geom_area(aes(x = seq_along(SMI), y = SMI),
            fill = "#58D6CC", alpha = 0.3, color = 1) +
  geom_area(aes(x = seq_along(DAX), y = DAX),
            fill = "#8D58D6", alpha = 0.3, color = 1) +
  theme_grey()
```



11.3 DIAGRAMA DE MÚLTIPLES LÍNEAS EN ggplot2

11.3.1 TRANSFORMACIÓN DE LOS DATOS

Se considera el siguiente data frame donde cada columna representa la trayectoria de un movimiento browniano²⁰:

```
# Fijamos una semilla
set.seed(999999)

# Grid
t = seq(0, 1, by = 0.0001)
p = length(t) - 1

# 6 Trayectorias diferentes
n = 6
I = matrix(rnorm(n * p, 0, 1 / sqrt(p)), n, p)

# Data frame
df1 = data.frame(apply(I, 1, cumsum))
```

²⁰El movimiento browniano es el movimiento aleatorio que se observa en las partículas que se hallan en un medio fluido, como resultado de choques contra las moléculas de dicho fluido.

```
# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df1 %>%
  head(15) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(latex_options = c("striped",
                                  "condensed", "HOLD_position"),
                position = "center",
                full_width = FALSE)
```

X1	X2	X3	X4	X5	X6
0.0102539	-0.0064597	0.0027347	-0.0022774	0.0062282	-0.0100550
0.0003871	-0.0237228	-0.0052553	-0.0081524	-0.0043009	-0.0380103
-0.0116545	-0.0341826	0.0083090	-0.0051014	0.0004294	-0.0310713
-0.0241962	-0.0421597	0.0033189	-0.0041975	0.0205329	-0.0380082
-0.0439862	-0.0384864	0.0083783	0.0043132	0.0112514	-0.0401661
-0.0383754	-0.0289077	-0.0098695	0.0012381	0.0258166	-0.0422290
-0.0333804	-0.0376845	-0.0229764	0.0079878	0.0289380	-0.0346333
-0.0426626	-0.0183089	-0.0306143	0.0117244	0.0183842	-0.0394351
-0.0308666	-0.0251875	-0.0220102	0.0216805	0.0135432	-0.0685020
-0.0365802	-0.0263710	-0.0155578	0.0126756	0.0315856	-0.0699490
-0.0408996	-0.0451627	0.0021906	0.0089998	0.0182305	-0.0708469
-0.0521025	-0.0509487	0.0026809	0.0162891	0.0291998	-0.0595504
-0.0635749	-0.0511469	0.0109832	0.0233398	0.0274084	-0.0584767
-0.0590991	-0.0541465	0.0153421	0.0085307	0.0288000	-0.0527801
-0.0608510	-0.0612701	0.0164985	0.0140119	0.0181563	-0.0615023

Para usar el data frame en ggplot2 se tendrá que transformar en lo que se conoce como **long format**. Se puede transformar haciendo uso de la función `melt()` del paquete “`reshape`”.

```
# install.packages("reshape")
library(reshape)

df = data.frame(x = seq_along(df1[, 1]),
                 df1)

# Formato long
df = melt(df, id.vars = "x")
```

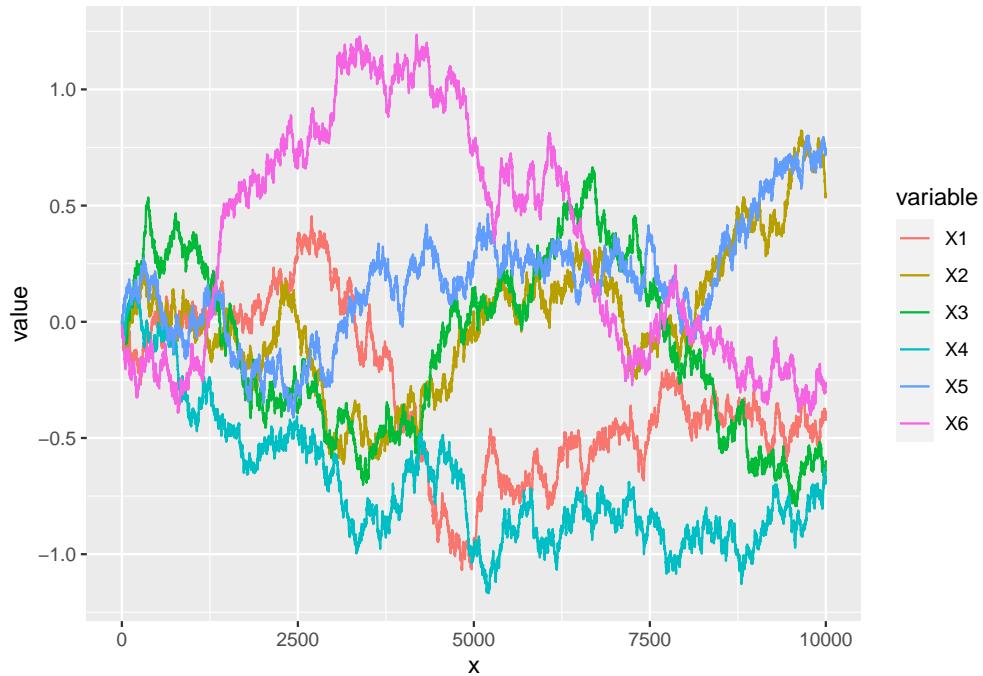
```
df %>%
  head(5) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(latex_options = c("striped",
                                  "condensed", "HOLD_position"),
                position = "center",
                full_width = FALSE)
```

x	variable	value
1	X1	0.0102539
2	X1	0.0003871
3	X1	-0.0116545
4	X1	-0.0241962
5	X1	-0.0439862

Dado un data frame en formato long como `df` es posible crear un diagrama de múltiples líneas en `ggplot2` con la función `geom_lines()` de la siguiente manera:

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("reshape")
library(reshape)

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line()
```

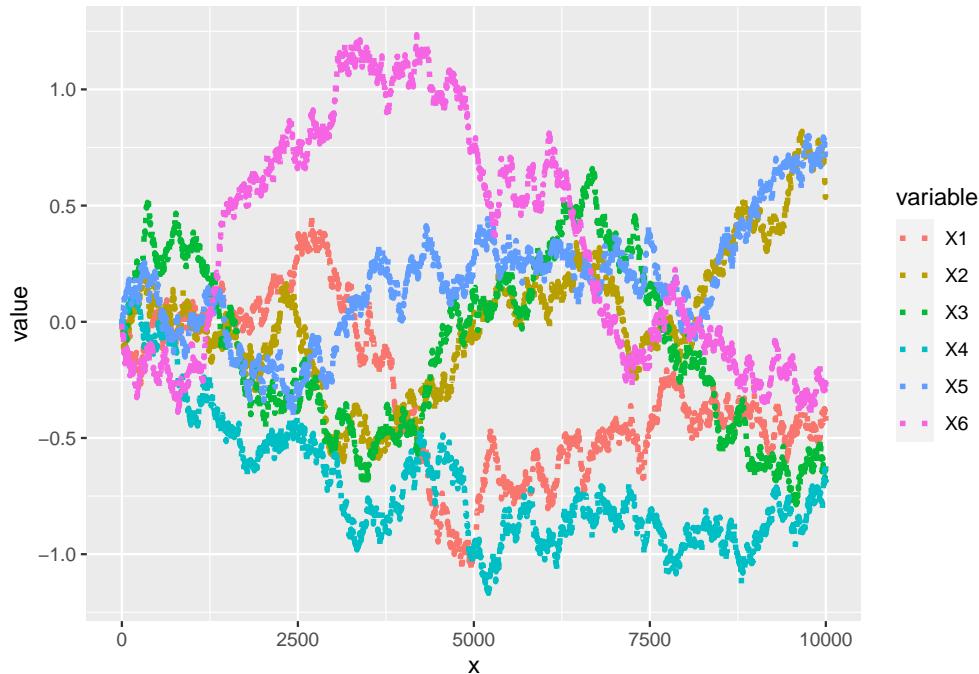


11.3.2 ANCHO Y TIPO DE LÍNEAS

El estilo de las líneas se puede cambiar haciendo uso de los argumentos de la función `geom_line()`, como el argumento `linetype` = para cambiar el tipo de línea o `lwd` = para cambiar el ancho.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line(linetype = 3,
            lwd = 1.2)
```



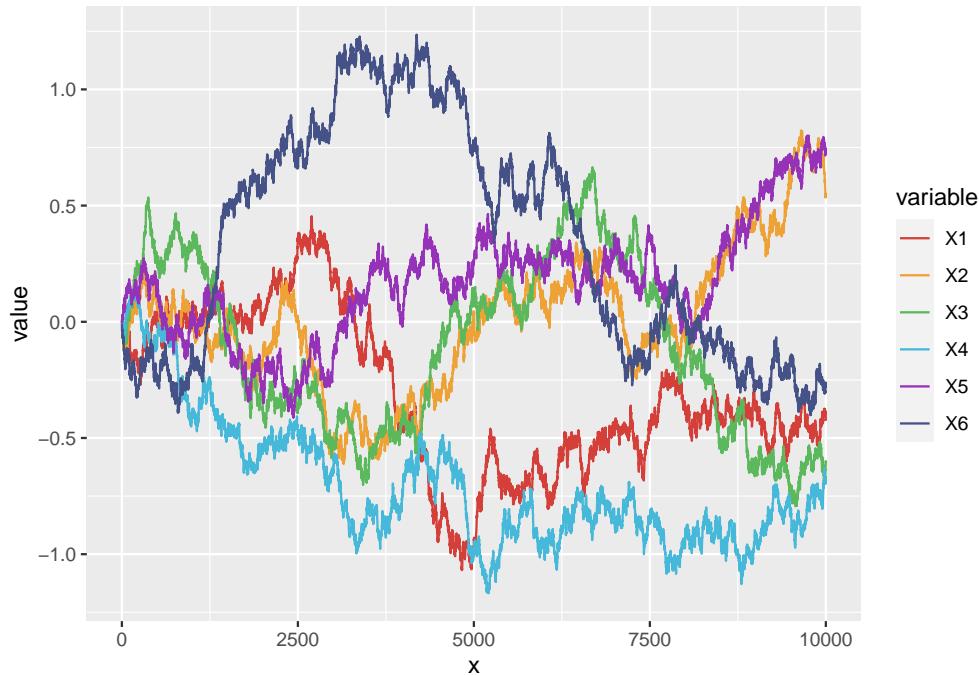
11.3.3 PERSONALIZACIÓN DEL COLOR

La paleta de colores por defecto se puede cambiar pasando un vector de colores al argumento `values` = de la función `scale_color_manual()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Selección de colores
cols = c("#D43F3A", "#EEA236", "#5CB85C", "#46B8DA", "#9632B8", "#455287")

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line() +
  scale_color_manual(values = cols)
```



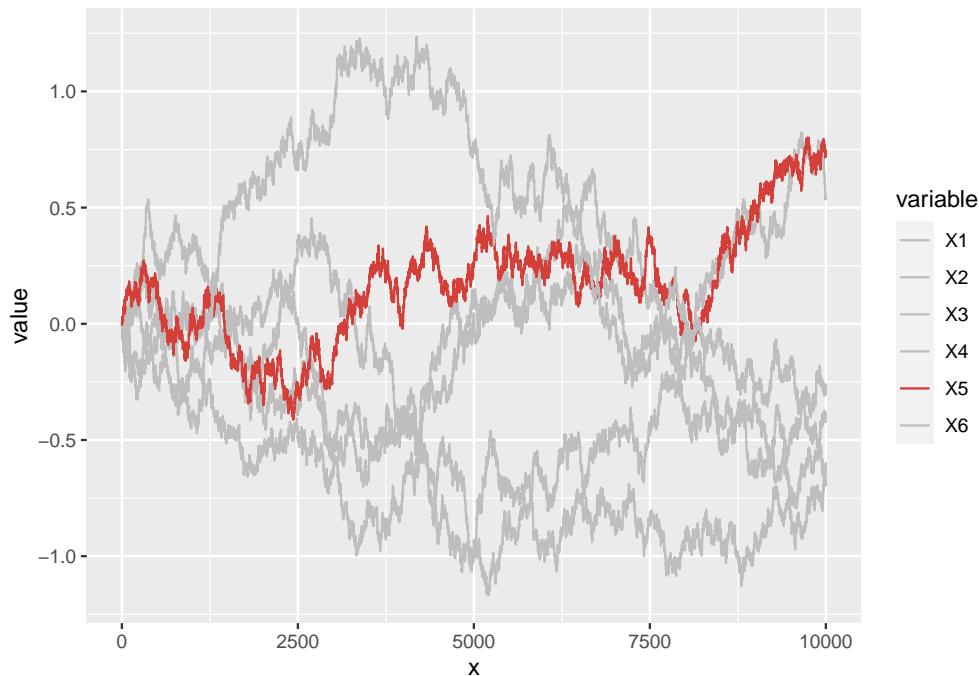
11.3.4 RESALTAR ALGUNAS LÍNEAS

Hay que tener en cuenta que usando el método anterior también se puede destacar algunas líneas del diagrama, usando el mismo color para todas menos para la que se desea resaltar.

```
# install.packages("ggplot2")
library(ggplot2)

# Selección de colores
cols <- c("gray", "gray", "gray", "gray", "#D43F3A", "gray")

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line() +
  scale_color_manual(values = cols)
```



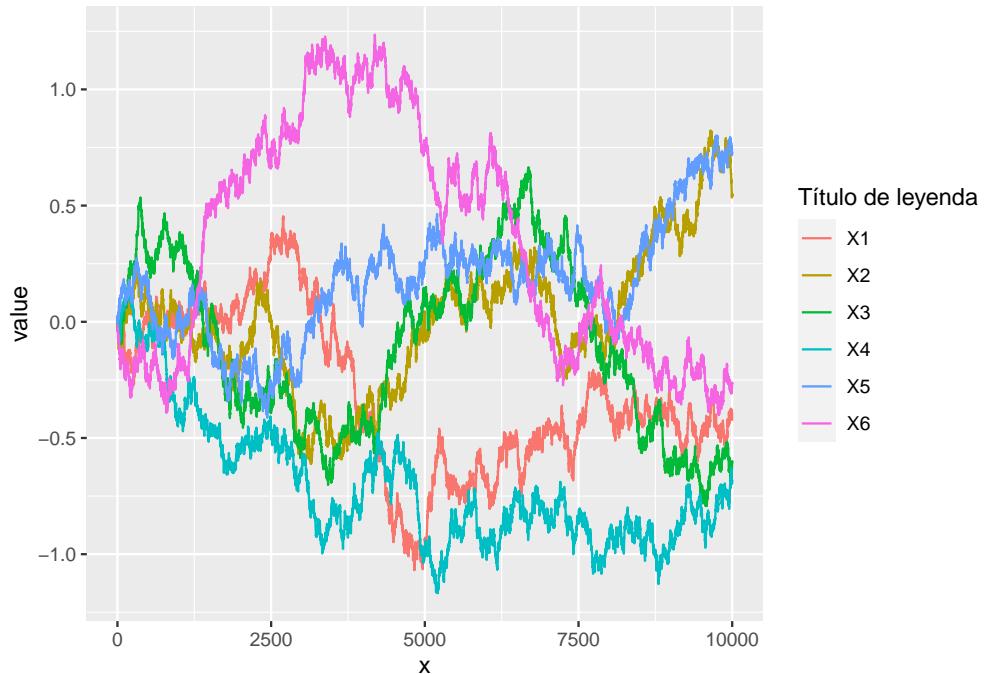
11.3.5 PERSONALIZACIÓN DE LA LEYENDA

11.3.5.1 TÍTULO

Se puede cambiar el título por defecto de la leyenda del diagrama de líneas con la función `guide_legend()` tal y como se muestra a continuación.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line() +
  guides(color = guide_legend(title = "Título de leyenda"))
```

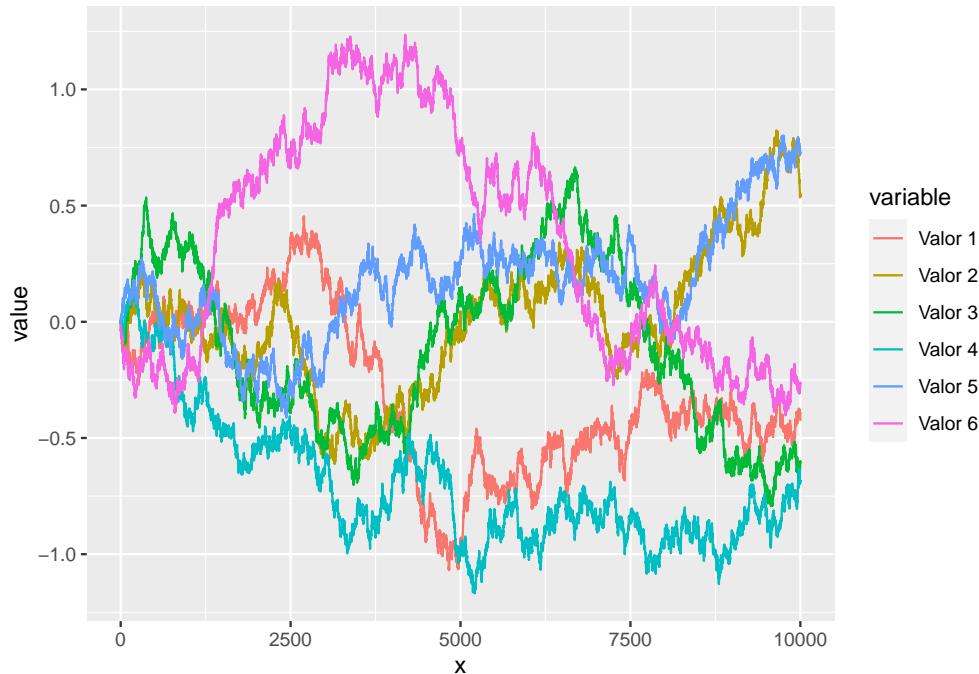


11.3.5.2 ETIQUETAS DE LA LEYENDA

Las etiquetas de la leyenda se pueden modificar haciendo uso del argumento `labels =` de la función `scale_color_discrete()`

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line() +
  scale_color_discrete(labels = paste("Valor", 1:6))
```

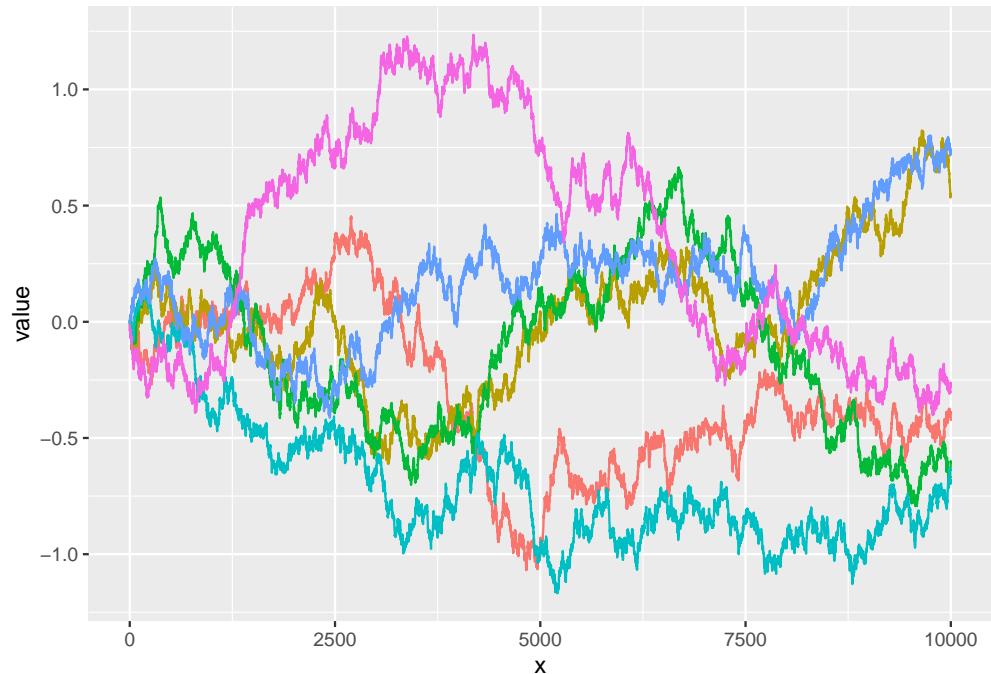


11.3.5.3 ELIMINAR LA LEYENDA DEL DIAGRAMA

Por último, si de desea deshacer de la leyenda se puede establecer el argumento `legend.position = "none"`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = value, color = variable)) +
  geom_line() +
  theme(legend.position = "none")
```



11.4 FUNCIONES CON `ggplot2`

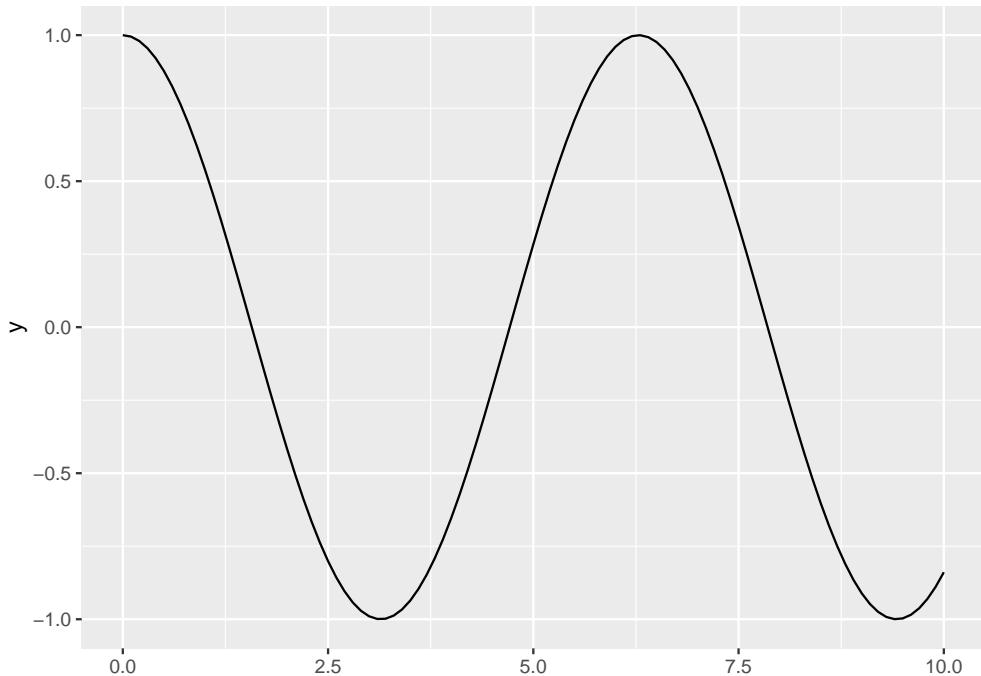
Una función es un algoritmo de correspondencia entre dos conjuntos de tal manera que a cada elemento del primero conjunto le corresponde uno y sólo un elemento al segundo conjunto.

11.4.1 DIBUJAR FUNCIONES EN `ggplot2`

La función `geom_function()` se puede usar para dibujar funciones en `ggplot2`. En este escenario no se necesita pasar un conjunto de datos en `ggplot()`, sino que se tiene que especificar los límites del Eje X con `xlim()` y la función a ser dibujada.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 10)) +
  geom_function(fun = cos)
```

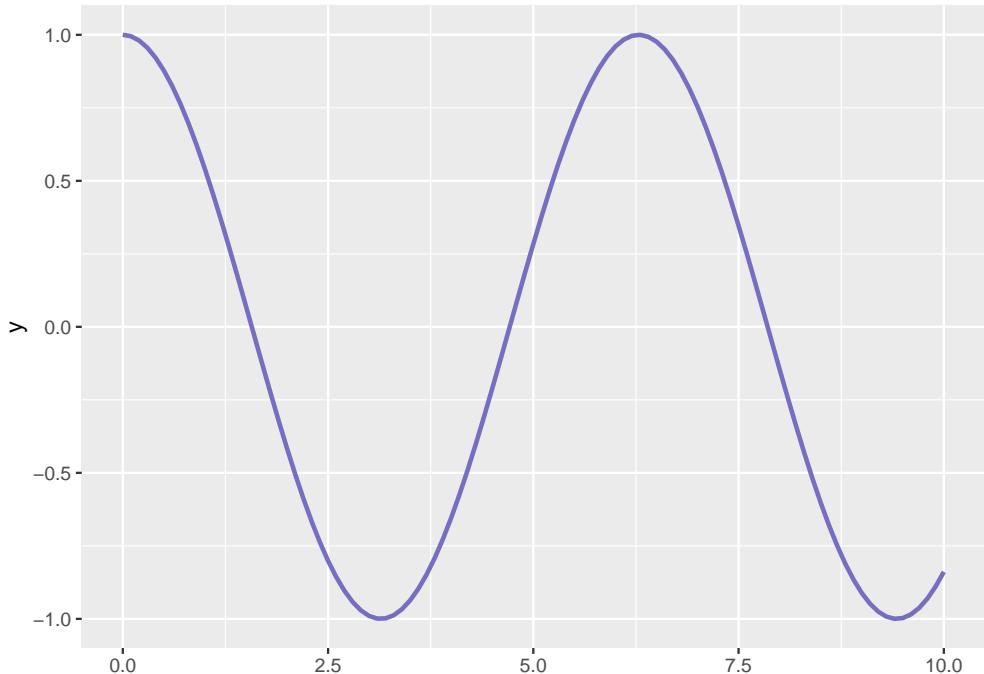


11.4.1.1 PERSONALIZACIÓN DE LA LÍNEA

La función dibujada se puede personalizar con los argumentos habituales, como `color =`, `lwd =`, y `linetype =` para el color, ancho y tipo de línea, respectivamente:

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 10)) +
  geom_function(fun = cos,
                colour = "#766ec0",
                lwd = 1,
                linetype = 1)
```

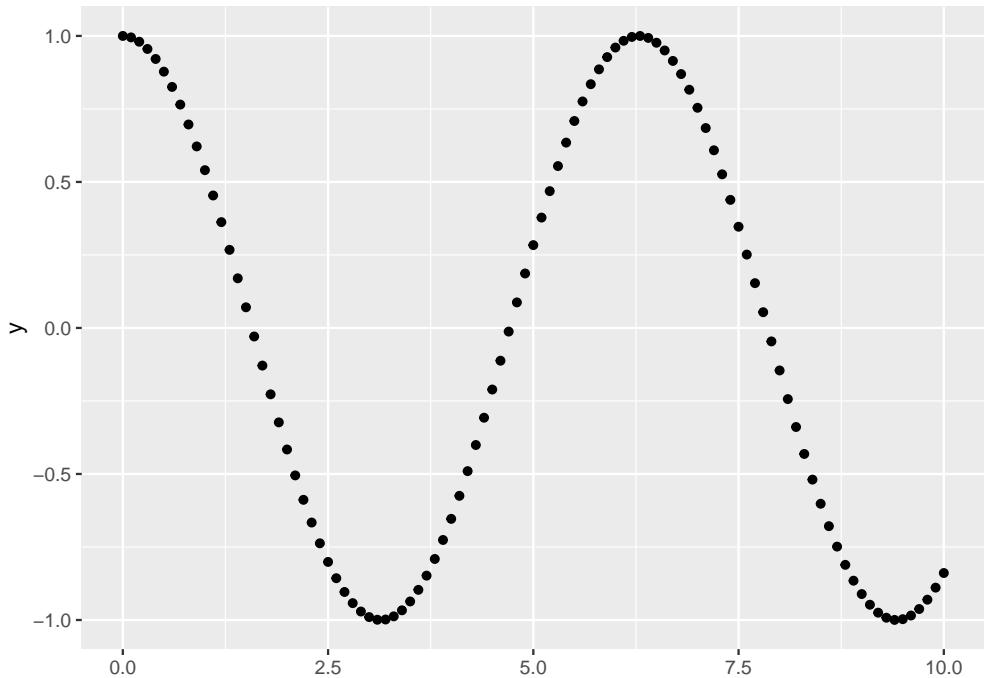


11.4.1.2 stat_function()

También se puede usar la función `stat_function()` en lugar de `geom_function()`. En este escenario se puede cambiar el geom por defecto, como se muestra en el siguiente ejemplo:

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 10)) +
  stat_function(fun = cos,
                geom = "point")
```

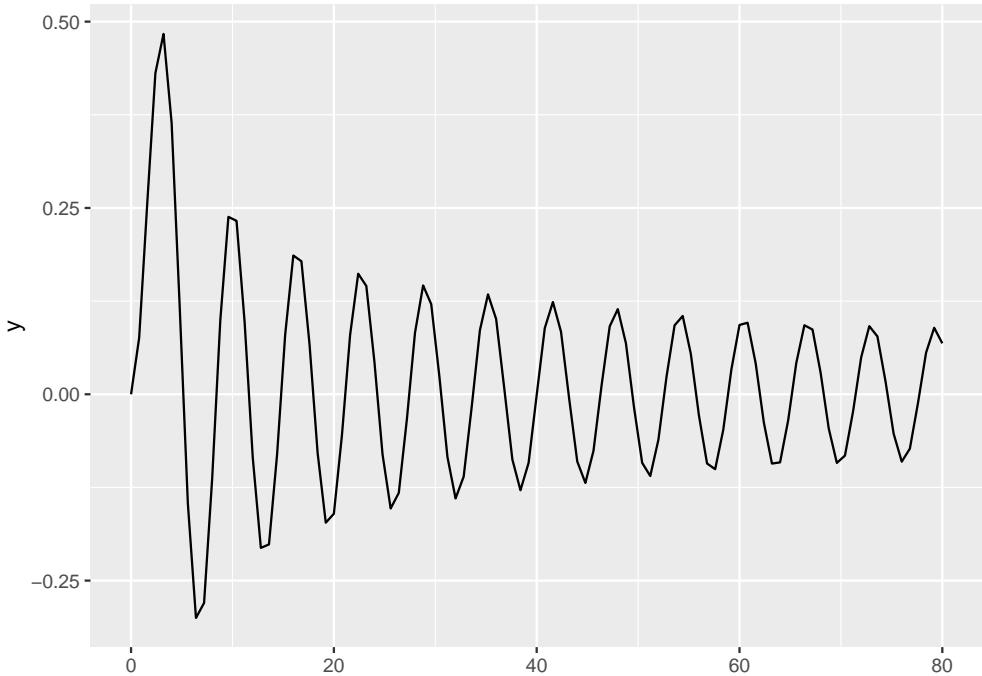


11.4.1.3 ARGUMENTOS ADICIONALES

Hay que tener en cuenta que si la función que se está dibujando tiene el argumento adicionales se les puede pasar a través de una lista de argumentos, `args`. En el siguiente ejemplo se está dibujando funciones bessel con `besselJ`, que requiere un argumento llamado `nu`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 80)) +
  geom_function(fun = besselJ,
                args = list(nu = 2))
```

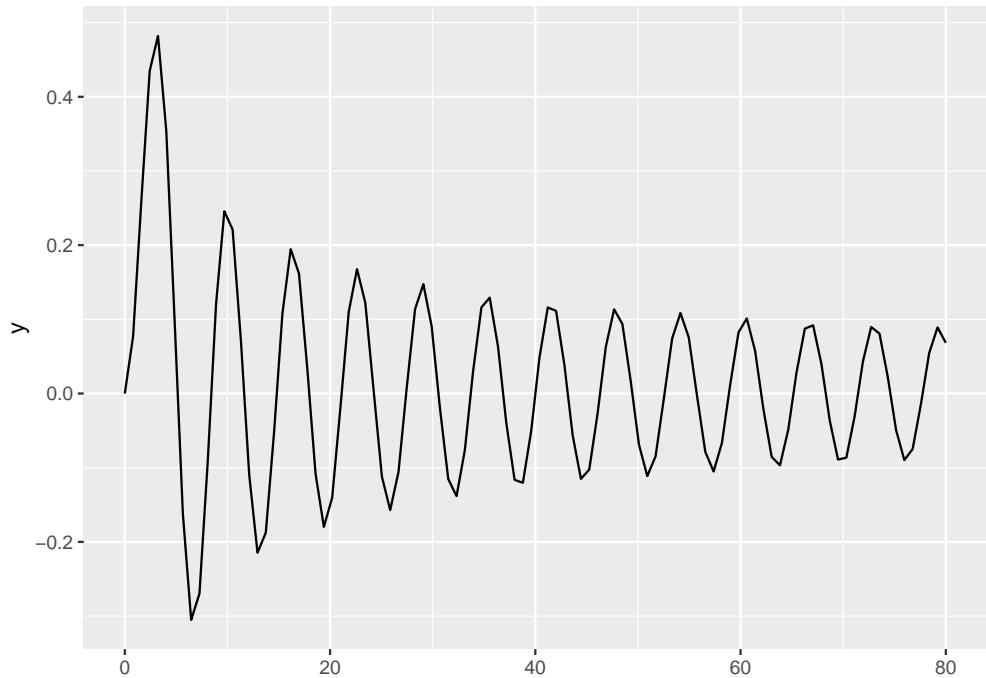


11.4.1.4 NÚMERO DE PUNTOS

Cabe destacar que el número de puntos usados para dibujar la función puede no ser suficiente, como por ejemplo el caso anterior. Para crear una función más suave, se puede incrementar el valor del argumento `n =`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 80)) +
  geom_function(fun = besselJ,
                n = 100,
                args = list(nu = 2))
```



11.4.2 SUPERPONER VARIAS FUNCIONES

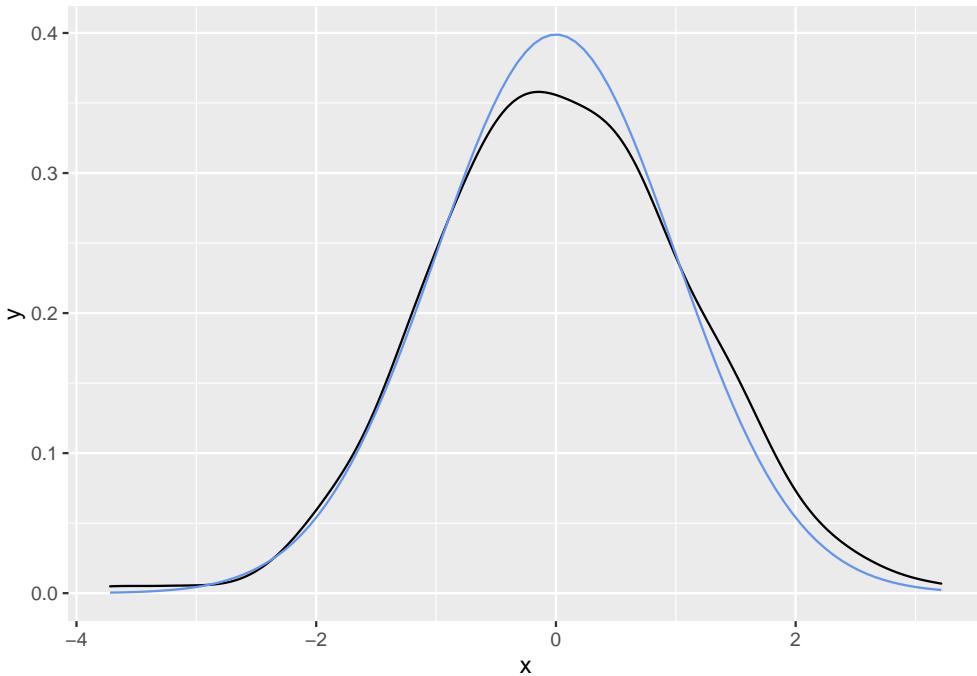
11.4.2.1 SUPERPONER SOBRE UN DIAGRAMA EXISTENTE

Si se ha creado un diagrama se puede añadir una función sobre él con la función `geom_function()`. En el siguiente ejemplo se superponen la distribución normal teórica sobre una distribucion normal generada con 300 muestras.

```
# install.packages("ggplot2")
library(ggplot2)

set.seed(777)
df = data.frame(x = rnorm(300))

ggplot(df, aes(x = x)) +
  geom_density() +
  geom_function(fun = dnorm, colour = "cornflowerblue")
```

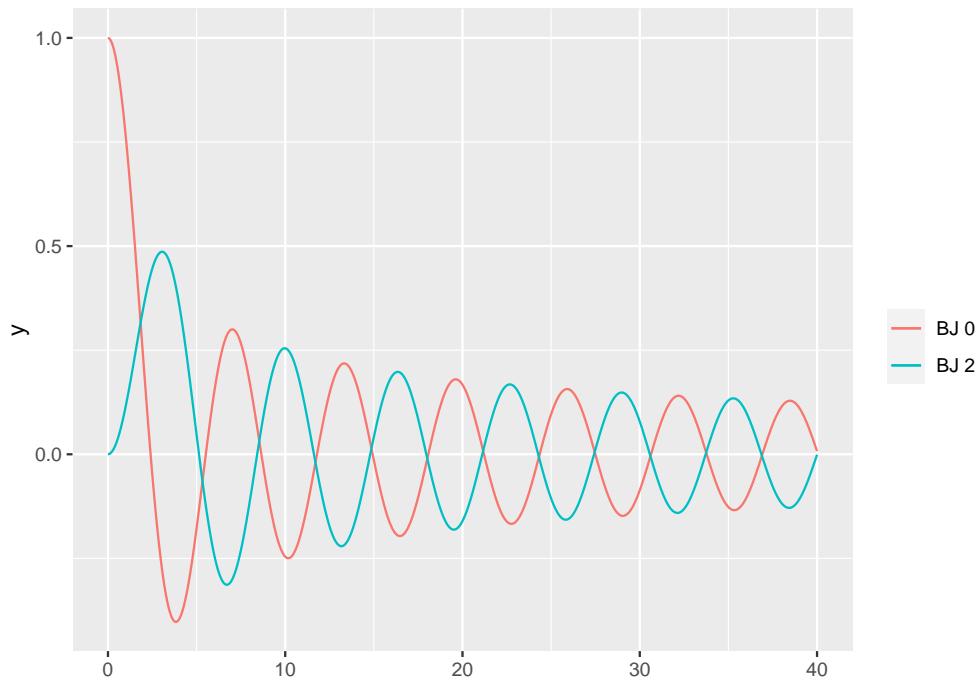


11.4.2.2 AÑADIENDO VARIAS FUNCIONES

Por último, vale la pena mencionar que se puede agregar varias funciones al mismo diagrama, añadiendo más capas.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  xlim(c(0, 40)) +
  geom_function(fun = besselJ, n = 600,
                aes(color = "BJ 0"),
                args = list(nu = 0)) +
  geom_function(fun = besselJ, n = 600,
                aes(color = "BJ 2"),
                args = list(nu = 2)) +
  guides(colour = guide_legend(title = ""))
```



11.5 DIAGRAMA DE LÍNEAS EN `ggplot2`

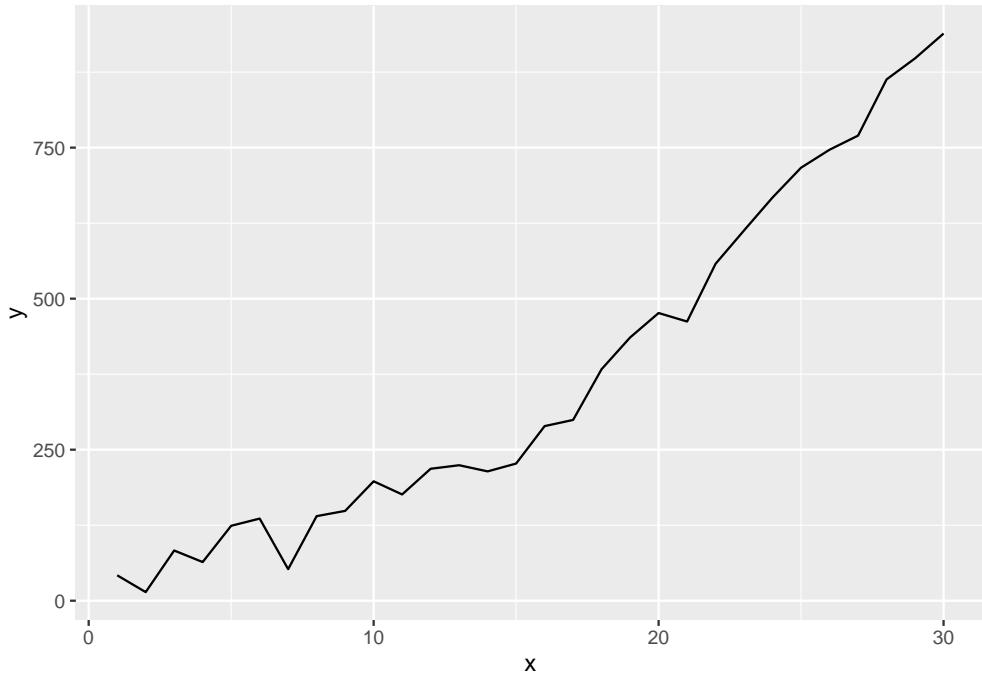
11.5.1 DIAGRAMA DE LÍNEAS CON LA FUNCIÓN `geom_line()`

Dado un data frame con una variable numérica `x` y otra variable numérica `y` que representa el valor para cada observación es posible crear un diagrama de líneas con `geom_line()` de la siguiente manera:

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(88)
x = 1:30
y = x ^ 2 + runif(30, 0, 100)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_line()
```



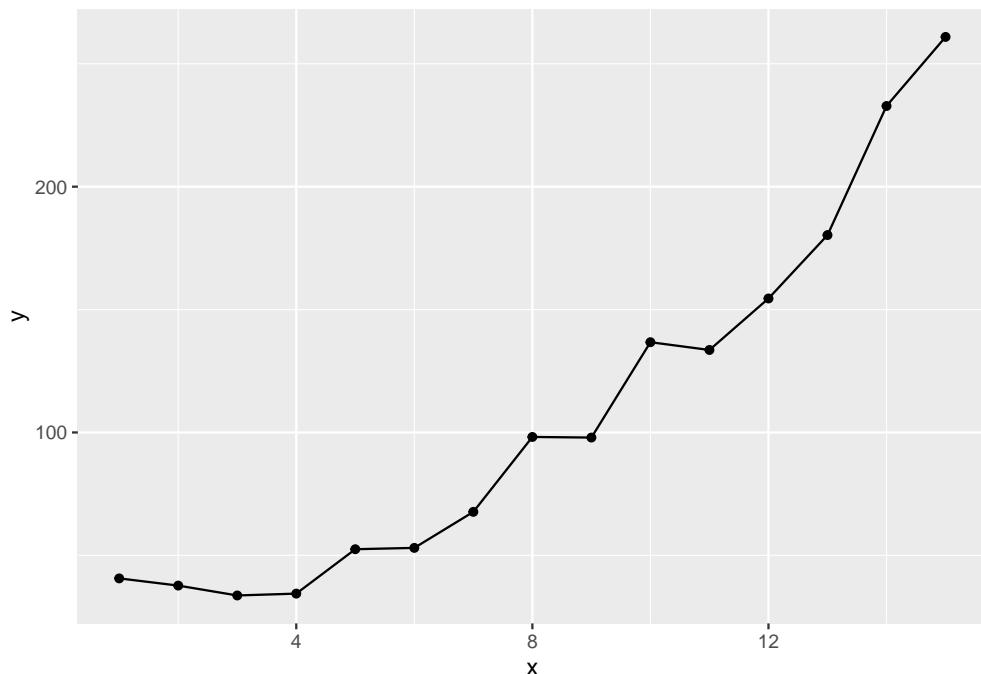
11.5.1.1 AGREGAR PUNTOS

Si se desea agregar la función `geom_point()` al diagrama, se mostrarán los puntos para cada par de observaciones.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(66)
x = 1:15
y = x ^ 2 + runif(15, 3, 40)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_line() +
  geom_point()
```



11.5.1.2 LÍNEA CON UNA FLECHA

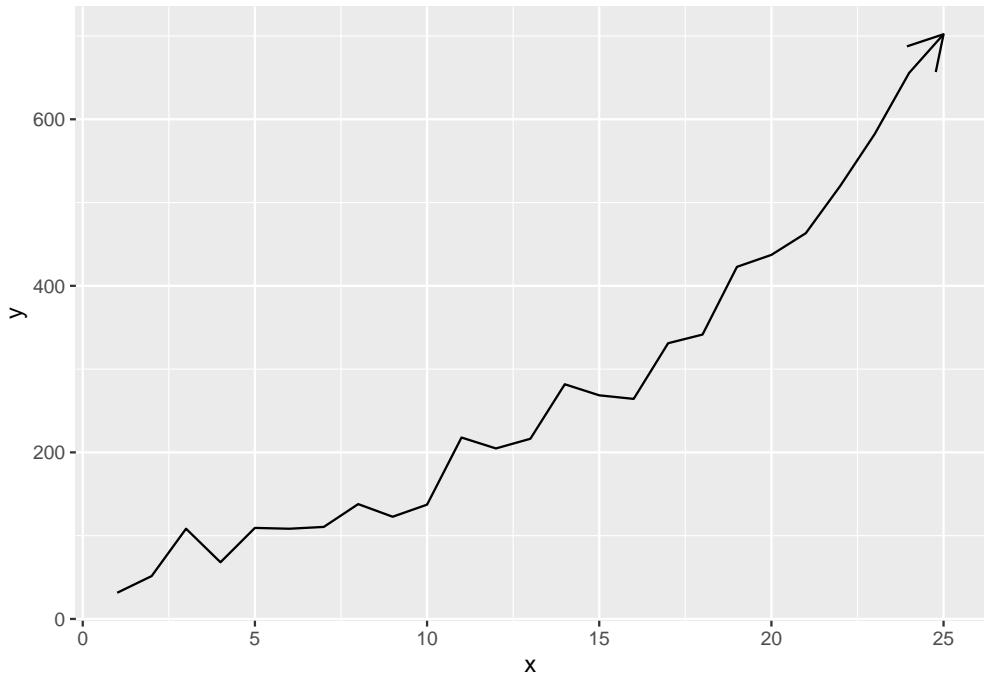
Si los datos representan una tendencia o un camino, se puede agregar una flecha al final de la línea pasando la función `arrow()` al argumento `arrow =`. Sólo se escribe `?arrow` para obtener detalles adicionales.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(22)
```

```
x = 1:25
y = x ^ 2 + runif(25, 0, 100)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_line(arrows = arrow())
```



11.5.2 DIAGRAMA DE ESCALERAS CON LA FUNCIÓN `geom_step()`

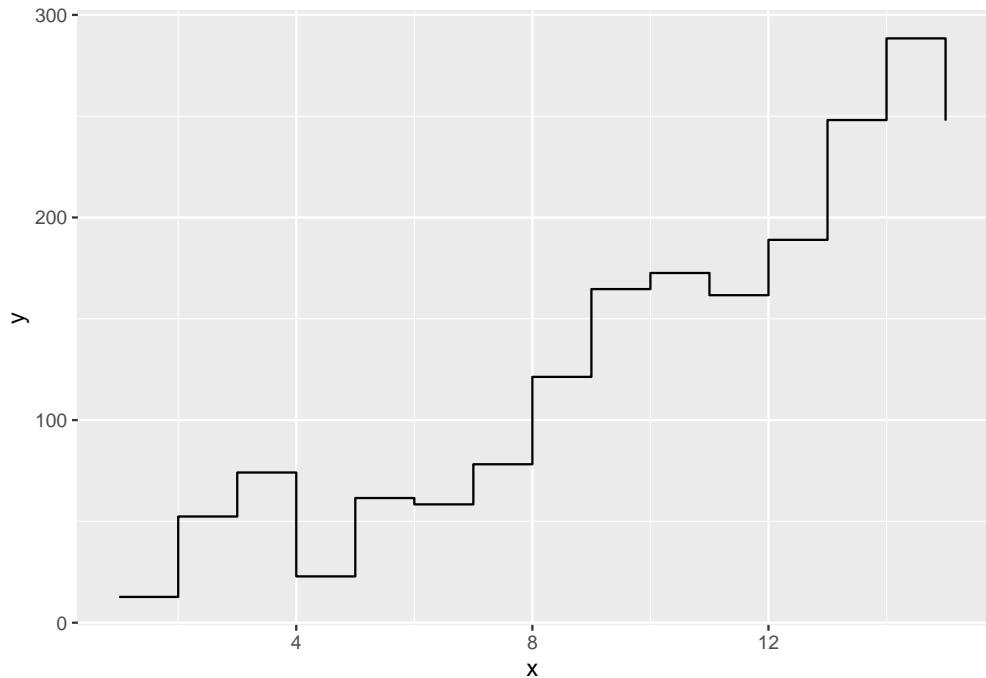
11.5.2.1 PRIMERA LÍNEA HORIZONTAL

Un geom alternativo es `geom_step()`, que creará un diagrama de escalera donde la primera línea será por defecto horizontal.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(19)
x = 1:15
y = x ^ 2 + runif(15, 0, 100)
df = data.frame(x = x, y = y)
```

```
ggplot(df, aes(x = x, y = y)) +
  geom_step()
```



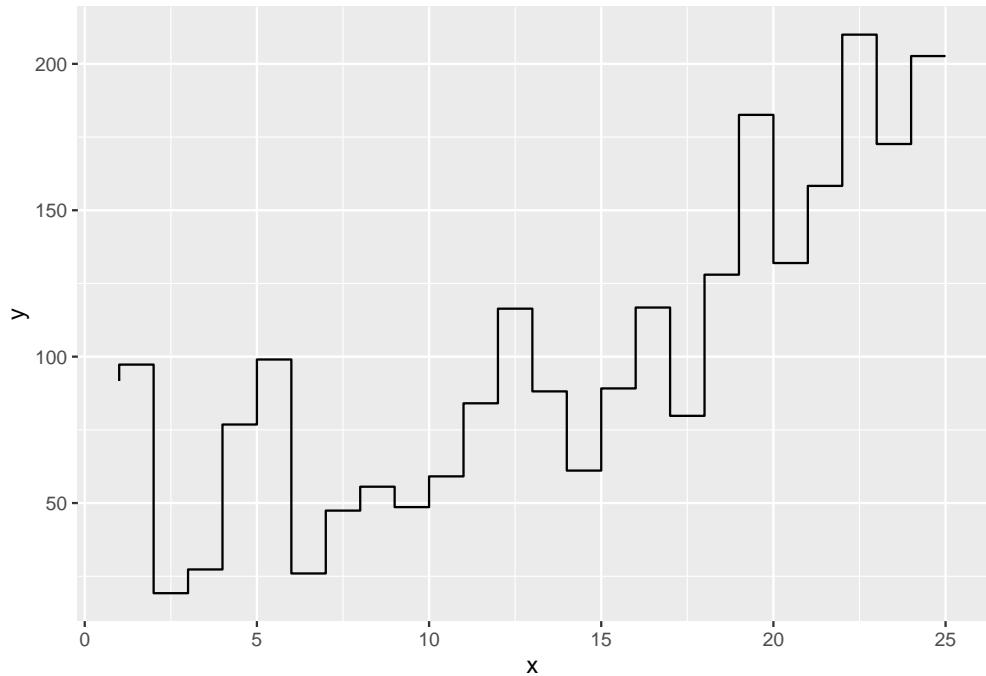
11.5.2.2 PRIMERA LÍNEA VERTICAL

Para la primera línea que sea vertical, se pasa la definición "vh" al argumento `direction =` de la función.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(2323)
x = 1:25
y = x ^ 1.5 + runif(25, 0, 100)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_step(direction = "vh")
```



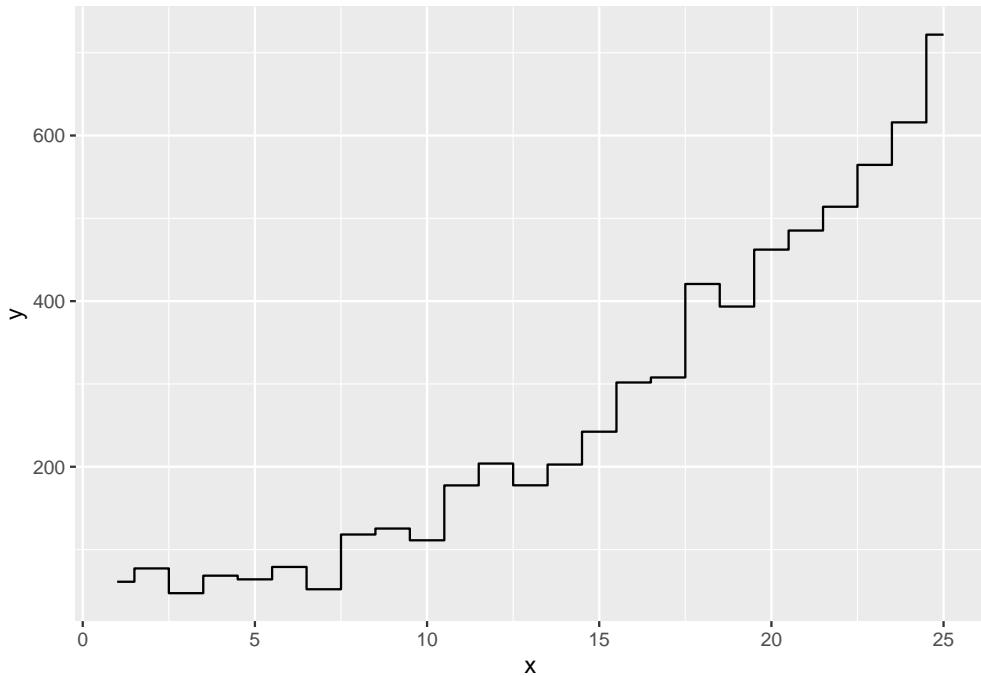
11.5.2.3 PASO INTERMEDIO ENTRE LOS VALORES

Otra opción es usar "mid", para un paso a medio camino entre los valores adyacentes del eje X.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos
set.seed(111)
x = 1:25
y = x ^ 2 + runif(25, 2, 100)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_step(direction = "mid")
```



11.5.3 COLOR DE LÍNEA Y ESTILO

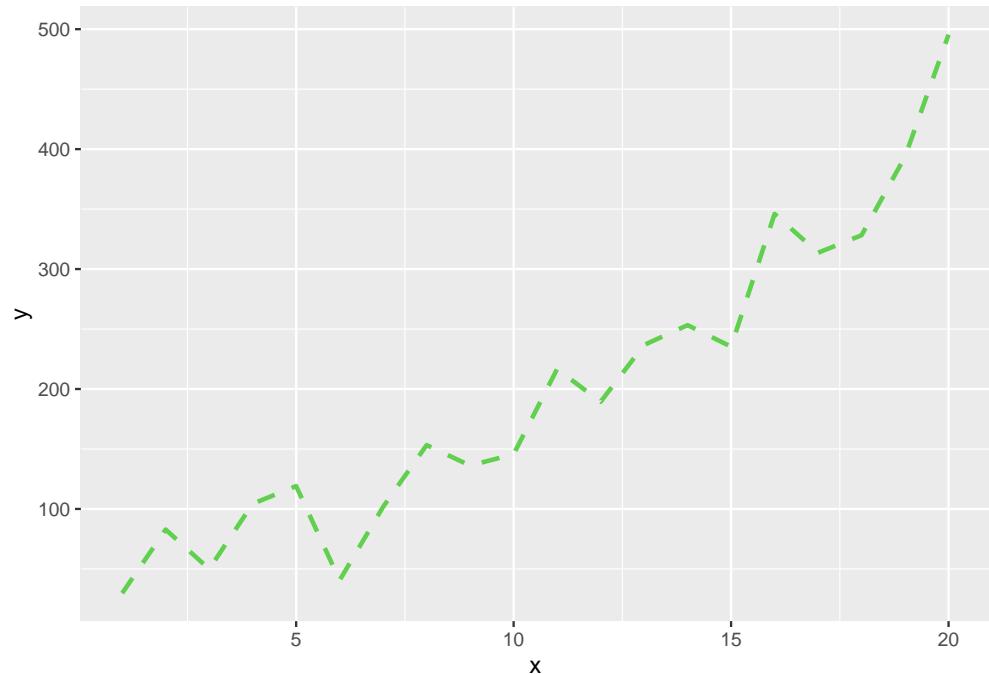
Los argumentos `color =`, `lwd =` y `linetype =` se pueden usar para modificar el color, el ancho y el tipo de líneas²¹, respectivamente. A continuación se pondrá un ejemplo en código para observar que la función puede también incluir el cambio en el diagrama.

```
# install.packages("ggplot2")
library(ggplot2)

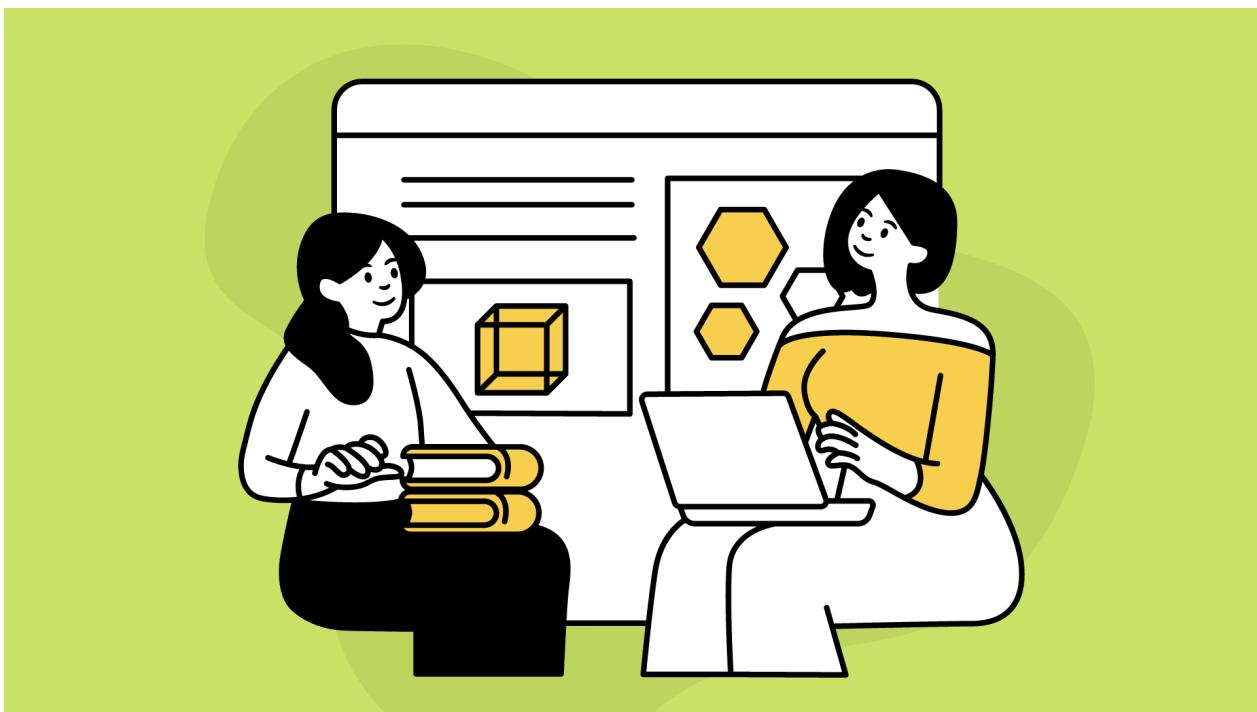
# Datos
set.seed(123)
x = 1:20
y = x ^ 2 + runif(20, 0, 100)
df = data.frame(x = x, y = y)

ggplot(df, aes(x = x, y = y)) +
  geom_line(color = 3,      # Color de la línea
            lwd = 1,      # Ancho de la línea
            linetype = 2) # Tipo de línea
```

²¹Cabe mencionar que el tipo de línea (`linetype =`), se puede establecer con números (0 a 6), con textos o con patrones.



CAPÍTULO 12 PARTE DE UN TODO



Este tipo de diagramas muestran los datos en porciones o sectores. Estas visualizaciones son especialmente útiles para representar conteos o grupos.

14.1 DIAGRAMA DE BARRAS APILADAS EN `ggplot2`

14.1.1 DATOS DE MUESTRA

Los siguientes datos representan las respuestas a la pregunta: “¿Cuántas horas al día ves la televisión?”. La variable X representa la edad de la persona, la variable Y representa su respuesta, la variable grupo representa su ciudad. Estos datos ficticios serán usados en los siguientes ejemplos de este subcapítulo.

```

# Datos
set.seed(999)

edad = factor(sample(c("Niño", "Joven", "Adulto", "Anciano"),
                     size = 150, replace = TRUE),
              levels = c("Niño", "Joven", "Adulto", "Anciano"))
horas = sample(1:4, size = 150, replace = TRUE)
ciudad = sample(c("A", "B", "C", "D"),
                size = 150, replace = TRUE)

df = data.frame(x = edad, y = horas, grupo = ciudad)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_option = c("striped", "condensed", "HOLD_position"),
    position = "center", full_width = FALSE
  )

```

x	y	grupo
Adulto	4	B
Anciano	3	C
Niño	2	D
Adulto	2	A
Niño	4	A
Joven	2	C
Niño	3	C
Joven	1	B
Joven	1	A
Joven	2	B

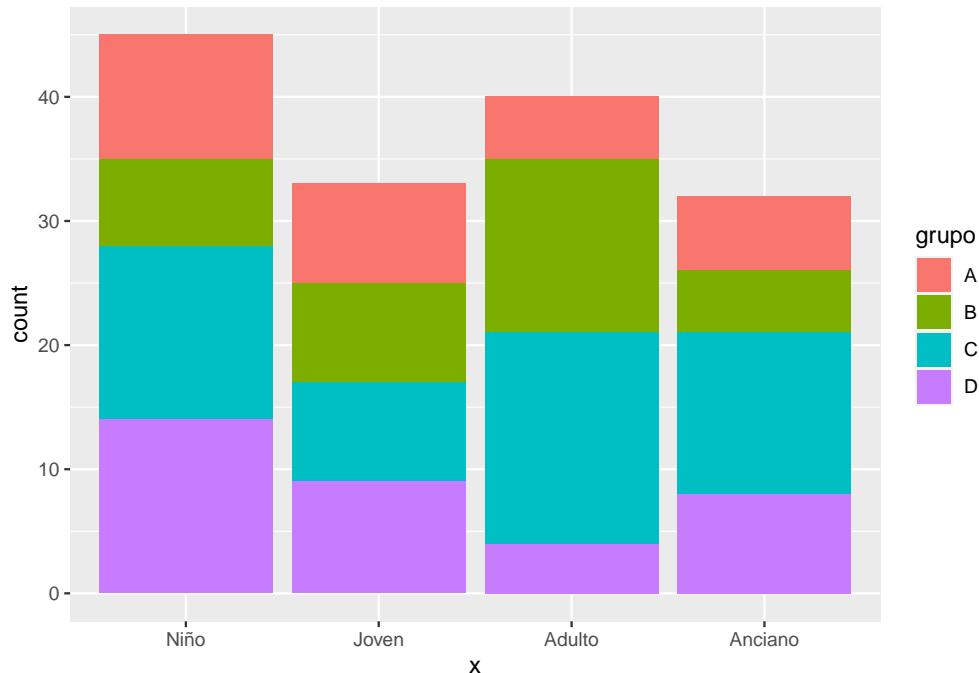
14.1.2 DIAGRAMA DE BARRAS APIADAS CON LA FUNCIÓN `geom_bar()`

14.1.2.1 EL ARGUMENTO POR DEFECTO `stat = "count"`

Si se usa la función `geom_bar()` con los argumentos por defecto se tendrá que pasar sólo `x` = o `y` = a la función `aes()`, además del argumento `fill` =. El diagrama de barras mostrará la suma apilada para cada grupo de la variable.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_bar()
```

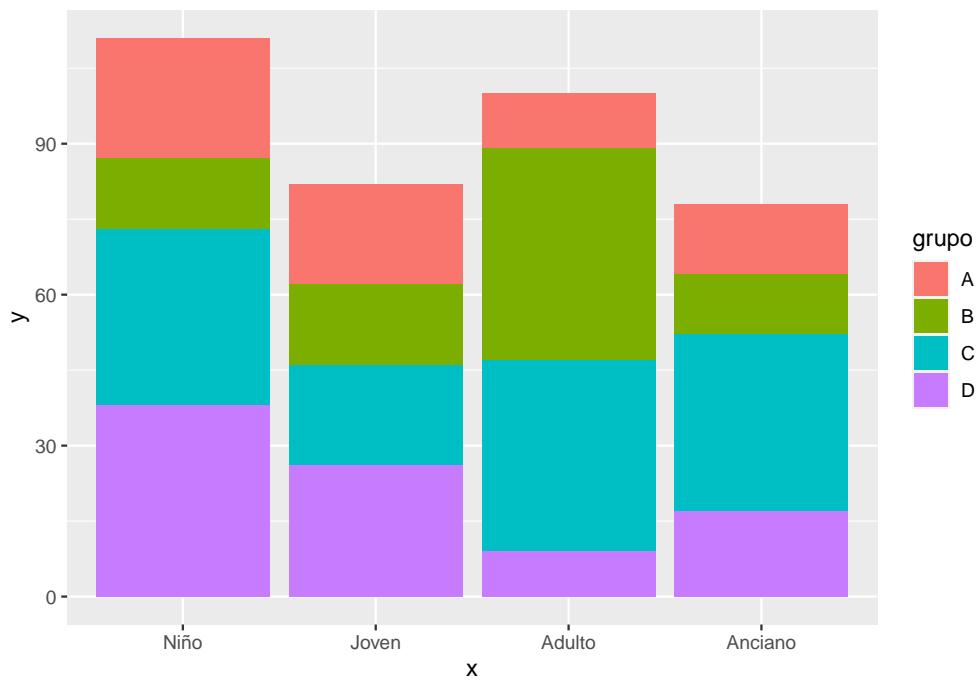


14.1.2.2 EL ARGUMENTO `stat = "identity"`

Si se establece `stat = "identity"` se puede crear un daigrama de barras apiladas para múltiples variables. En este argumento y escenario se puede pasar otra variable a la función `aes()`, representando el valor o el conteo de la variable.

```
# install.packages("ggplot2")
library(ggplot2)
```

```
ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity")
```

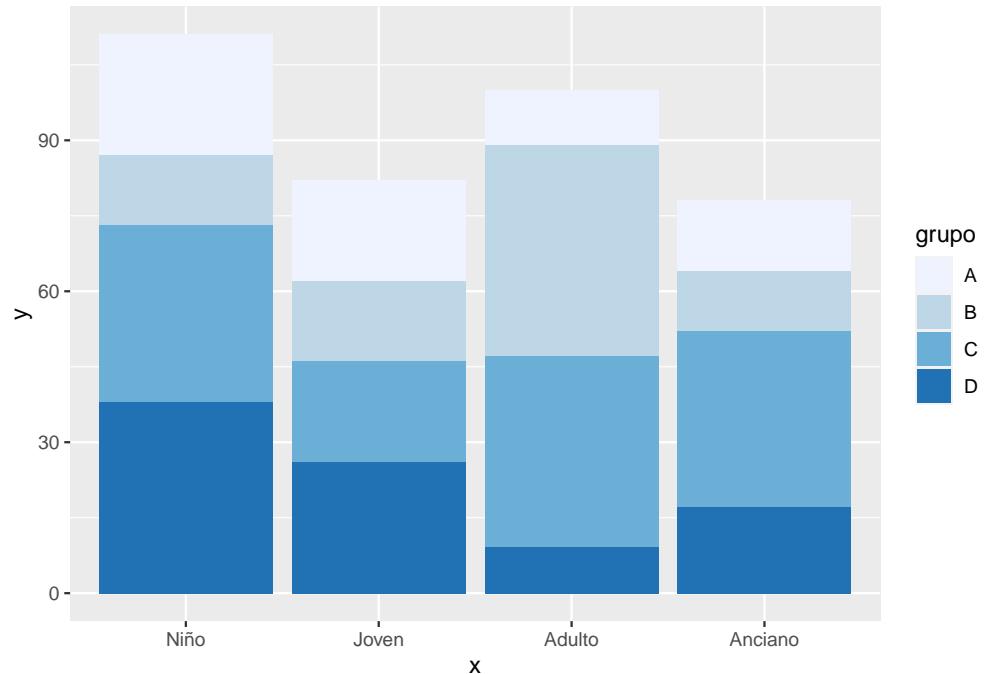


14.1.3 COLOR DE FONDO Y DEL BORDE DE LAS BARRAS

14.1.3.1 PALETA PREDEFINIDA Se puede cambiar los colores de las barras apiladas con una paleta predefinida como las que proporciona la función `scale_fill_brewer()`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity") +
  scale_fill_brewer()
```

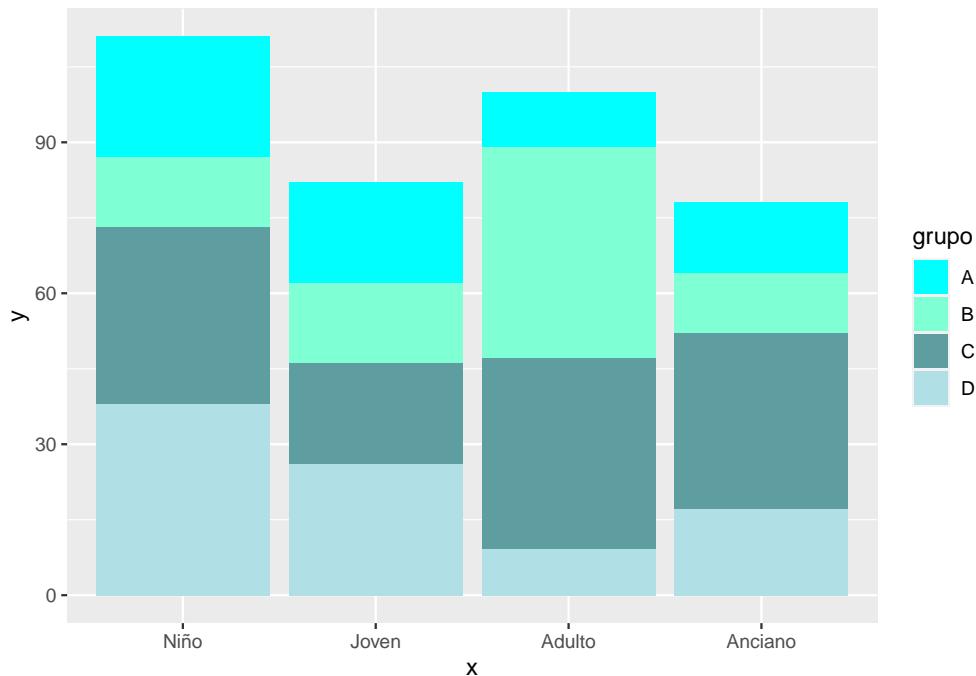


14.1.3.2 COLORES PERSONALIZADOS

Si se prefiere elegir colores se puede usar la función `scale_fill_manual()` y pasar un vector de colores al argumento `values`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("#00FFFF", "#7FFFD4", "#5F9EA0", "#BOE0E6"))
```

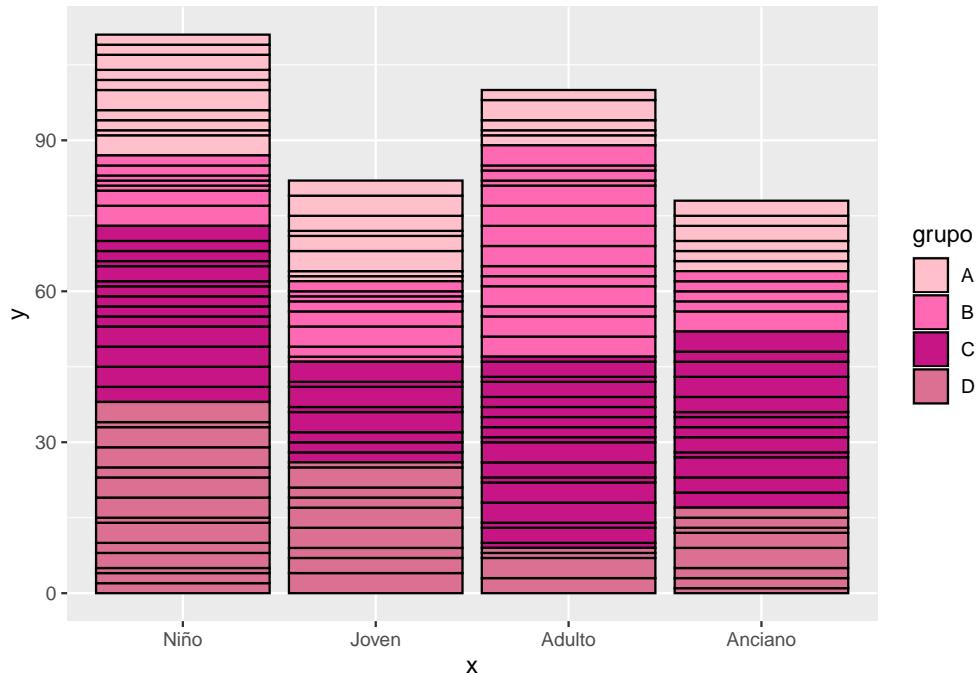


14.1.3.3 COLOR DEL BORDE (`stat = "identity"`)

En caso de que se esté creando un diagrama de barras con el argumento `stat = "identity"`, se puede establecer un color de borde pasando un color al argumento `color` = de la función `geom_bar()`, pero el borde resaltará todas las barras representando los valores de la variable `y`.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity", color = "black") +
  scale_fill_manual(values = c("#FFCOCB", "#FF69B4", "#C71585", "#DB7093"))
```



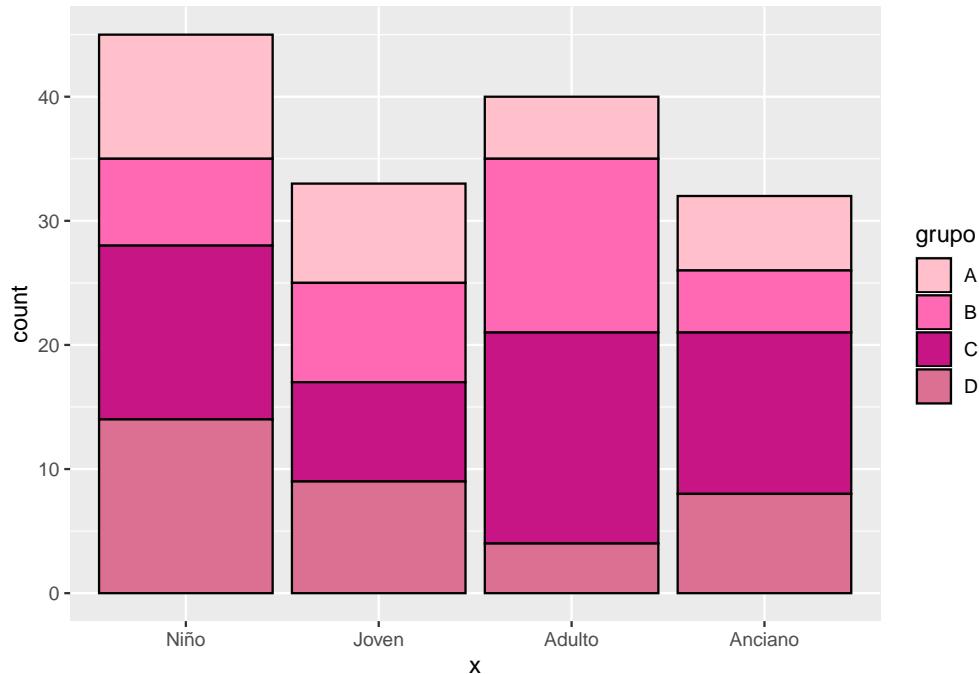
Nótese que esta forma no es la más presentable para mostrar un resumen de un diagrama de barras apilado. Ya que muestra la división por cada parte del data frame.

14.1.3.4 COLOR DEL BORDE (`stat = "count"`)

Si se desea crear un diagrama de barras apiladas más adecuado y estético a la vista en los bordes, se mostrará alrededor de cada barra con el argumento `stat = "count"`, ya que no habrá otra variable, mas que un conteo por cada sector de grupo.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = grupo)) +
  geom_bar(color = "black") +
  scale_fill_manual(values = c("#FFCOCB", "#FF69B4", "#C71585", "#DB7093"))
```



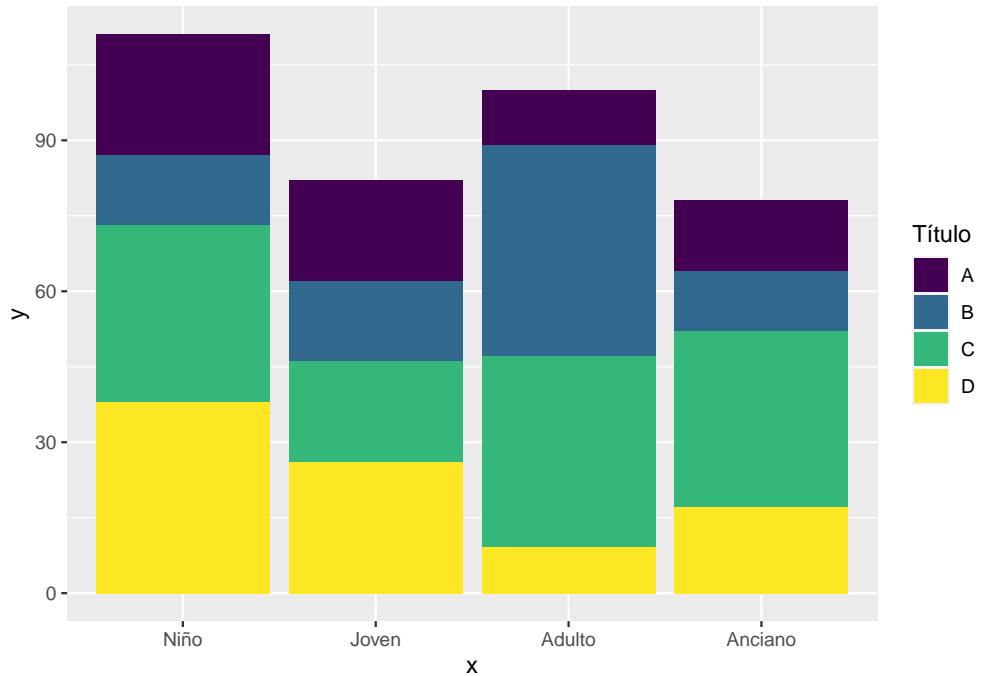
14.1.4 PERSONALIZACIÓN DE LA LEYENDA

14.1.4.1 TÍTULO DE LA LEYENDA

El título de la leyenda del diagrama de barras se corresponde por defecto al nombre de la variable pasada al argumento `fill =` de la función `aes()` dentro de la función `ggplot()`, pero se puede sobreescribir con el siguiente código como ejemplo:

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("viridis")
library(viridis)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity") +
  # La siguiente función ayuda a personalizar título de la leyenda
  guides(fill = guide_legend(title = "Título")) +
  scale_fill_viridis(discrete = TRUE)
```

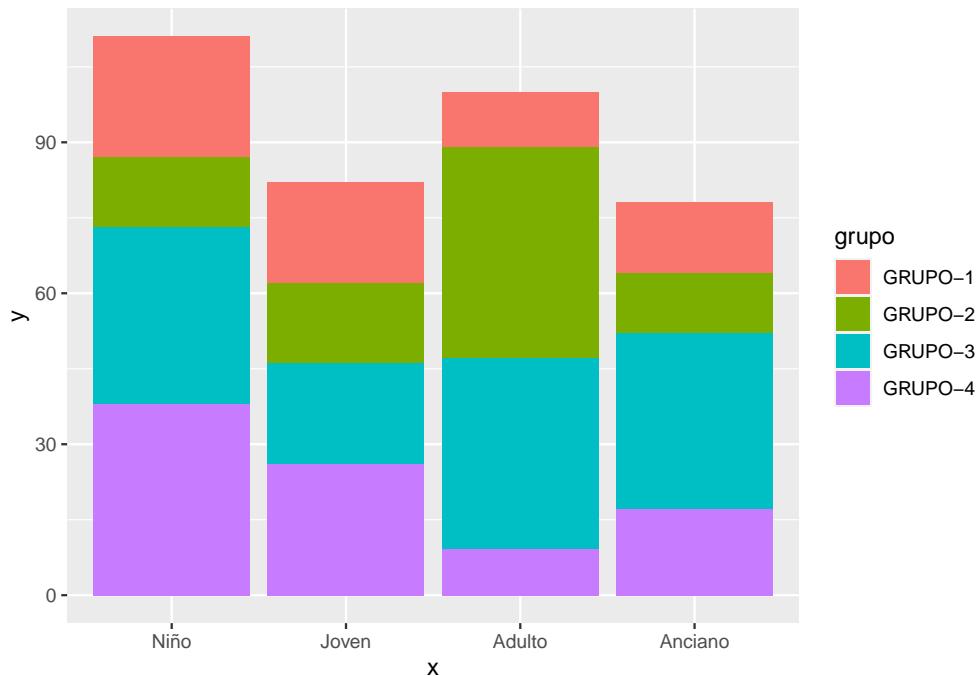


14.1.4.2 ETIQUETAS DE LA LEYENDA

Las etiquetas de la leyenda son los nombres de la variable categórica pasada al argumento `fill =`. Si se necesita cambiar estos valores se puede usar el argumento `labels =` de la función `scale_fill_discrete()` o de la función `scale_fill_manual()` si se cambian los colores de fondo:

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity") +
  scale_fill_discrete(labels = c("GRUPO-1", "GRUPO-2", "GRUPO-3",
                                "GRUPO-4"))
```

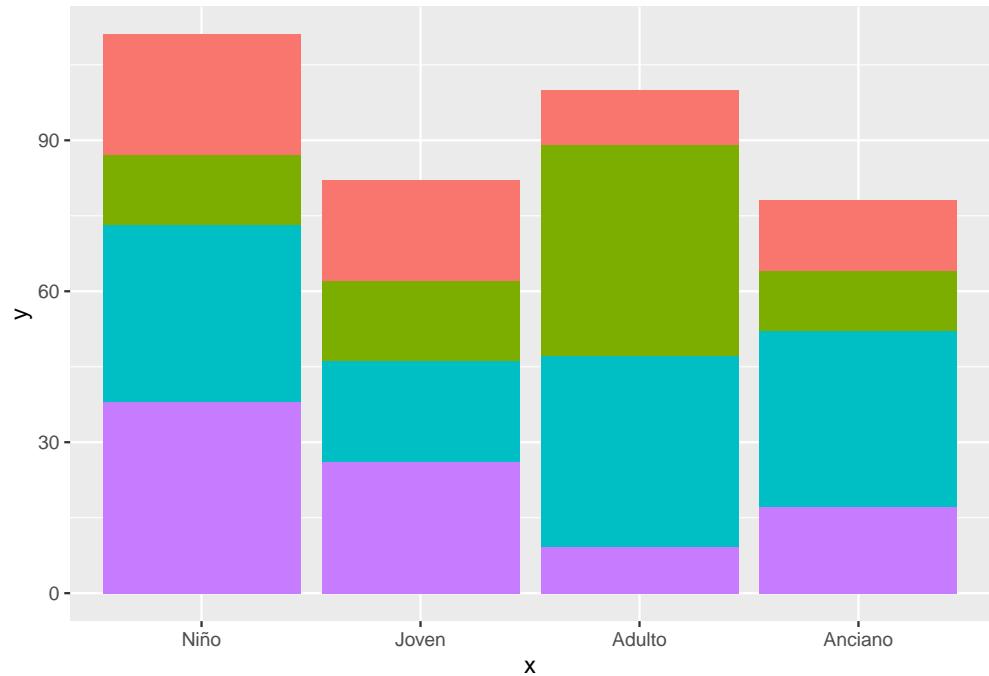


14.1.4.3 ELIMINAR LA LEYENDA

Por último, si se desea borrar la leyenda, sólamente se utiliza el argumento `legend.position =` de la función `theme()` como "none". Hay que tener en cuenta que también se puede cambiar la posición de la leyenda con este componente.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, y = y, fill = grupo)) +
  geom_bar(stat = "identity") +
  theme(legend.position = "none")
```



14.2 DIAGRAMA DE SECTORES CON PORCENTAJES EN ggplot2

14.2.1 TRANSFORMACIÓN DE LOS DATOS

El siguiente conjunto de datos contiene las respuestas (Si, No, o N/A) de una encuesta. Estos datos serán transformados y usados en los ejemplos de esta parte del capítulo.

```
set.seed(2022024)

# Variables
res = sample(c("Si", "No", "N/A"),
             size = 100, replace = TRUE,
             prob = c(0.4, 0.35, 0.25))
gen = sample(c("Hombre", "Mujer"),
             size = 100, replace = TRUE)

# Cambiamos los niveles de la variable para
# que "Si" aparezca primero en la leyenda
res = factor(res, levels = c("Si", "No", "N/A"))

# Data frame
datos = data.frame(respuesta = res,
                     genero = gen)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center", full_width = FALSE)
```

x	y	grupo
Adulto	4	B
Anciano	3	C
Niño	2	D
Adulto	2	A
Niño	4	A
Joven	2	C
Niño	3	C
Joven	1	B
Joven	1	A
Joven	2	B

Haciendo uso de la librería `dplyr` se puede obtener el porcentaje de cada tipo de respuesta para cada género. El siguiente ejemplo se está calculando el porcentaje por tipo de respuesta y agregando una nueva columna con porcentajes, haciendo uso de la función `pcntent()` de la librería `scales`.

```
# install.packages("dplyr")
# install.packages("scales")
library(dplyr)

# Transformación de los datos
df = datos %>%
  group_by(respuesta) %>% # Variable a ser transformada
  count() %>%
  ungroup() %>%
  mutate(pcnt = `n` / sum(`n`)) %>%
  arrange(pcnt) %>%
  mutate(etiquetas = scales::percent(pcnt))

# install.packages("kableExtra")
library(kableExtra)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center", full_width = FALSE)
```

respuesta	n	pcnt	etiquetas
N/A	25	0.25	25.0 %
No	36	0.36	36.0 %
Si	39	0.39	39.0 %

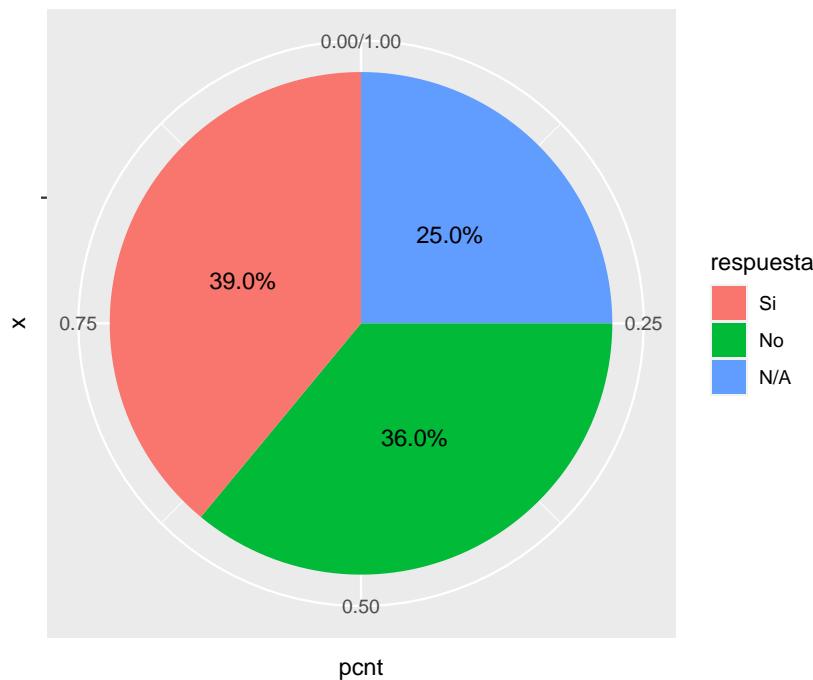
De la anterior tabla, la primer columna representa las posibles respuestas, la segunda columna representa la frecuencia absoluta, la tercera columna representa la frecuencia relativa y la cuarta columna representa el porcentaje con símbolo.

14.2.2 AGREGANDO LOS PORCENTAJES COMO ETIQUETAS

La columna `labels` permite añadir las etiquetas con porcentajes. En este ejemplo se están agregando con la función `geom_text()`:

```
# install.packages("ggplot2")
library(ggplot2)

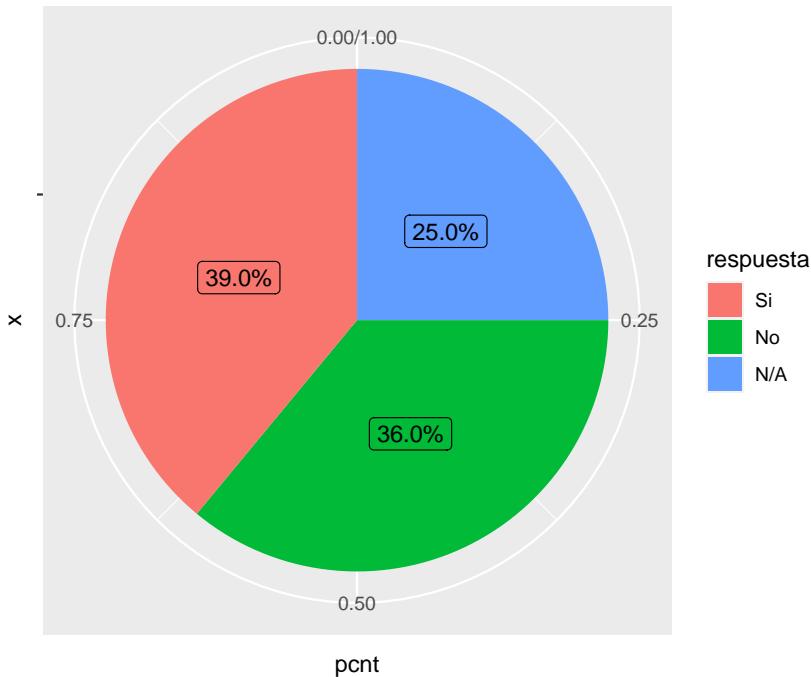
ggplot(df, aes(x = "", y = pcnt, fill = respuesta)) +
  geom_col() +
  geom_text(aes(label = etiquetas),
            position = position_stack(vjust = 0.5)) +
  coord_polar(theta = "y")
```



Una alternativa a la función `geom_text()` es la función `geom_label()`. Hay que tener en cuenta que se ha tenido que añadir el argumento `show.legend = FALSE` para evitar que se muestre una letra sobre los recuadros de la leyenda.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = "", y = pcnt, fill = respuesta)) +
  geom_col() +
  geom_label(aes(label = etiquetas),
             position = position_stack(vjust = 0.5),
             show.legend = FALSE) +
  coord_polar(theta = "y")
```



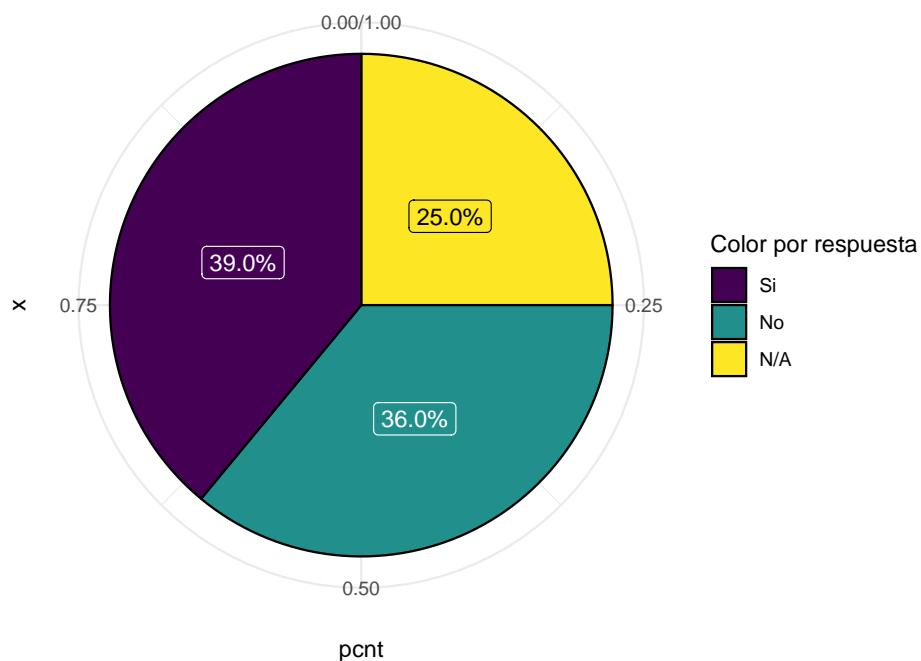
14.2.3 PERSONALIZACIÓN AVANZADA

Los diagramas de sectores se pueden personalizar de distintas maneras. Se puede personalizar la leyenda, los colores o los temas. En el siguiente ejemplo se ha eliminado el tema por defecto con la función `theme_minimal()`.²²

²²Se recomienda leer el artículo sobre [diagramas de sectores en ggplot2](#) para obtener detalles adicionales sobre personalizar los diagramas de sectores en ggplot2 y su leyenda.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = "", y = pcnt, fill = respuesta)) +
  geom_col(color = "black") +
  geom_label(aes(label = etiquetas), color = c(1, "white", "white"),
             position = position_stack(vjust = 0.5),
             show.legend = FALSE) +
  guides(fill = guide_legend(title = "Color por respuesta")) +
  scale_fill_viridis_d() +
  coord_polar(theta = "y") +
  theme_minimal()
```



14.3 DIAGRAMA VORONOI EN `ggplot2` CON EL PAQUETE `ggeomvoronoi`

Un diagrama de Voronoi es un conjunto de puntos en un plano de la división de dichas regiones, de tal forma, que a cada punto le asigna una región del plano formada por los puntos que son más cercanos a él que a ninguno de los otros objetos. En otras palabras, ayuda a dividir el plano en tantas regiones como puntos se tengan, de tal forma que a cada punto se le asigna la región formada por todo lo que está más cerca de él que ningún otro.

14.3.1 DATOS DE MUESTRA

El siguiente conjunto de datos contiene tres columnas donde la primera son los valores de la variable del eje X, la segunda los del eje Y, y la tercera la distancia euclídea entre los puntos y el centro del diagrama, que corresponde con el punto (300, 300).

```
# Datos
set.seed(0)
x = sample(1:400, size = 130)
y = sample(1:400, size = 130)
dist = sqrt((x - 300) ^ 2 + (y - 300) ^ 2)

df = data.frame(x, y, dist = dist)

# install.packages("kableExtra")
library(kableExtra)
# install.packages("dplyr")
library(dplyr)

df %>%
  head(10) %>%
  kable(booktabs = TRUE, format = "latex") %>%
  kable_styling(
    latex_options = c("striped", "condensed", "HOLD_position"),
    position = "center",
    full_width = FALSE
)
```

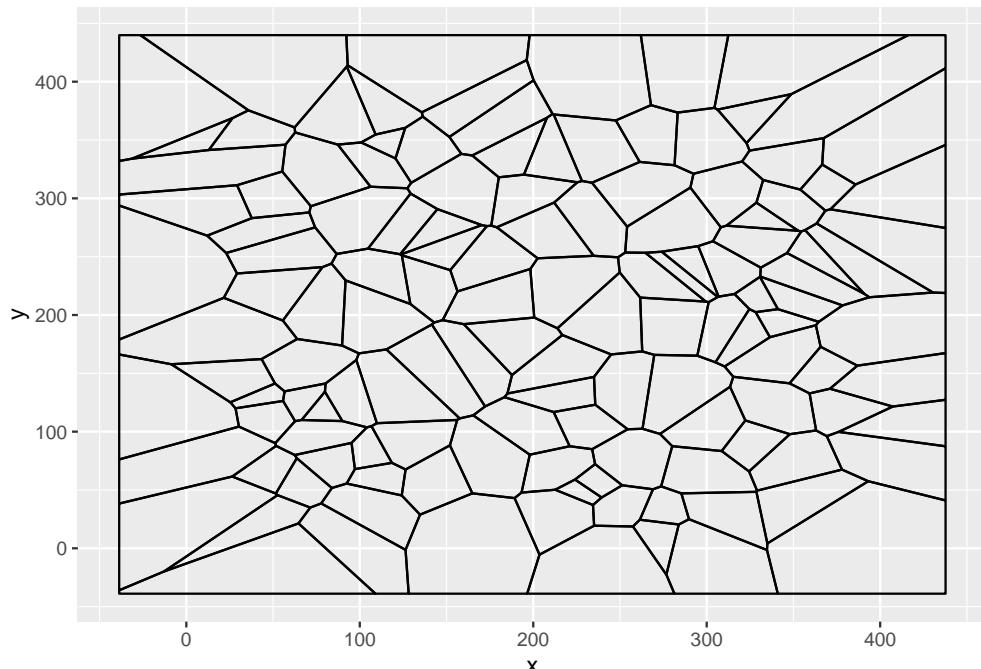
x	y	dist
398	98	224.51726
324	194	108.68303
167	19	310.88583
129	273	173.11846
299	31	269.00186
270	355	62.64982
187	174	169.24834
307	237	63.38770
85	75	311.20733
277	16	284.92982

14.3.2 DIAGRAMA DE VORONOI CON `stat_voronoi()`

La función `stat_voronoi()` se puede utilizar para crear un diagrama de Voronoi, pasando las variables `x` e `y` a la función `aes()` y usando `stat_voronoi(geom = "path")`.

```
#install.packages("ggbvoronoi")
# install.packages("ggplot2")
library(ggbvoronoi)
library(ggplot2)

ggplot(df, aes(x, y)) +
  stat_voronoi(geom = "path")
```

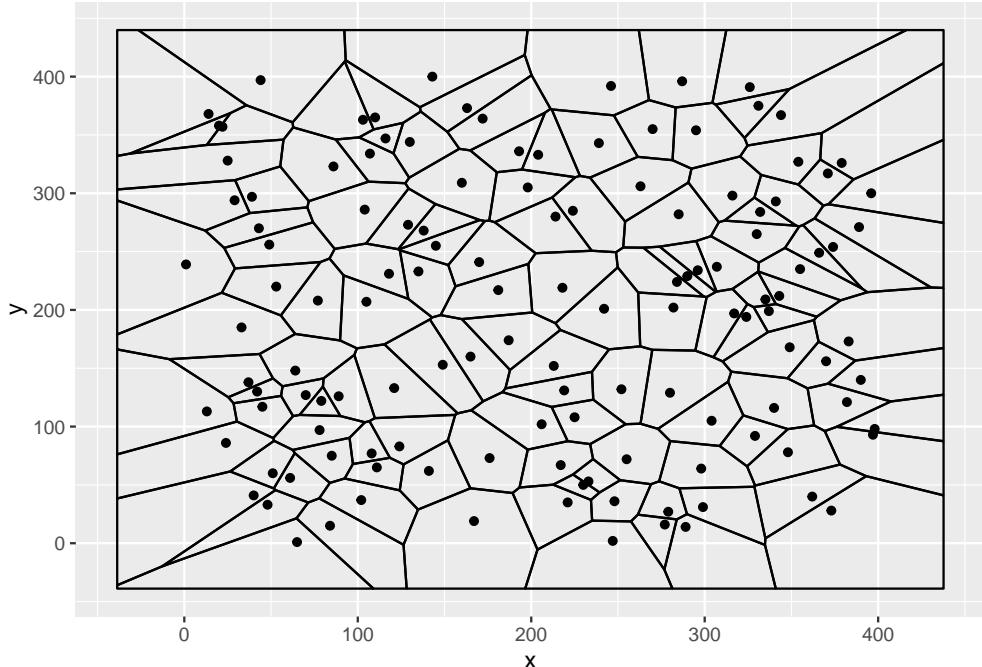


14.3.3 OBSERVACIONES

También se pueden agregar las observaciones haciendo uso de la función `geom_point()`

```
# install.packages("ggevoronoi")
# install.packages("ggplot2")
library(ggevoronoi)
library(ggplot2)

ggplot(df, aes(x, y)) +
  stat_voronoi(geom = "path") +
  geom_point()
```



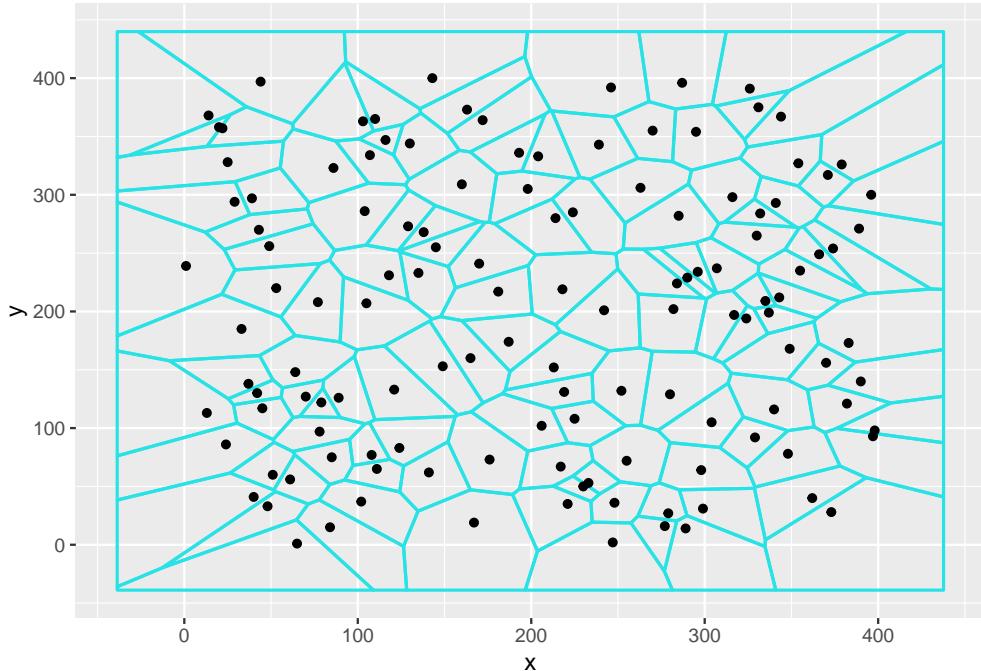
14.3.4 PERSONALIZACIÓN DE LAS LÍNEAS

Hay que tener en cuenta que se pueden cambiar el estilo de las líneas con los argumentos `color =`, `lwd =` y `linetype =`.

```
# install.packages("ggevoronoi")
# install.packages("ggplot2")
library(ggevoronoi)
library(ggplot2)

ggplot(df, aes(x, y)) +
```

```
stat_voronoi(geom = "path",
             color = 5,      # Color de las líneas
             lwd = 0.7,       # Grosor de las líneas
             linetype = 1) + # Tipo de líneas
geom_point()
```

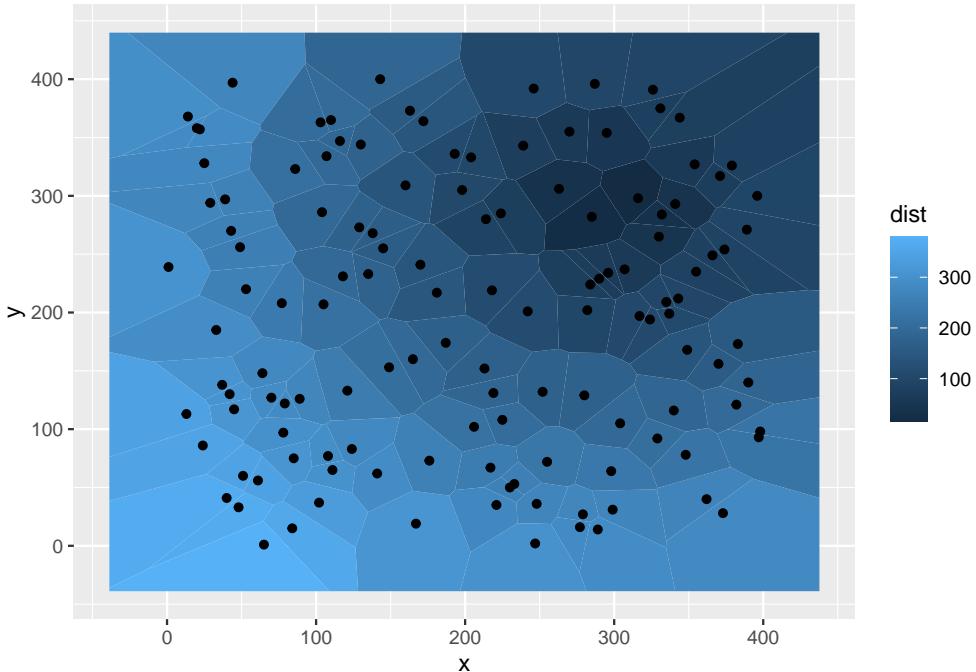


14.3.5 MAPA DE CALOR CON VORONOI

Se puede pasar una variable al argumento `fill =` de la función `aes()` ára crear un mapa de calor de Voronoi. Para este propósito se tendrá que usar la función `geom_voronoi()`

```
# install.packages("ggevoronoi")
# install.packages("ggplot2")
library(ggevoronoi)
library(ggplot2)

ggplot(df, aes(x, y, fill = dist)) +
  geom_voronoi() +
  geom_point()
```



14.4 DIAGRAMA DE WAFFLE (SQUARE PIE) EN ggplot2

14.4.1 DIAGRAMA DE WHAFFLE BÁSICO

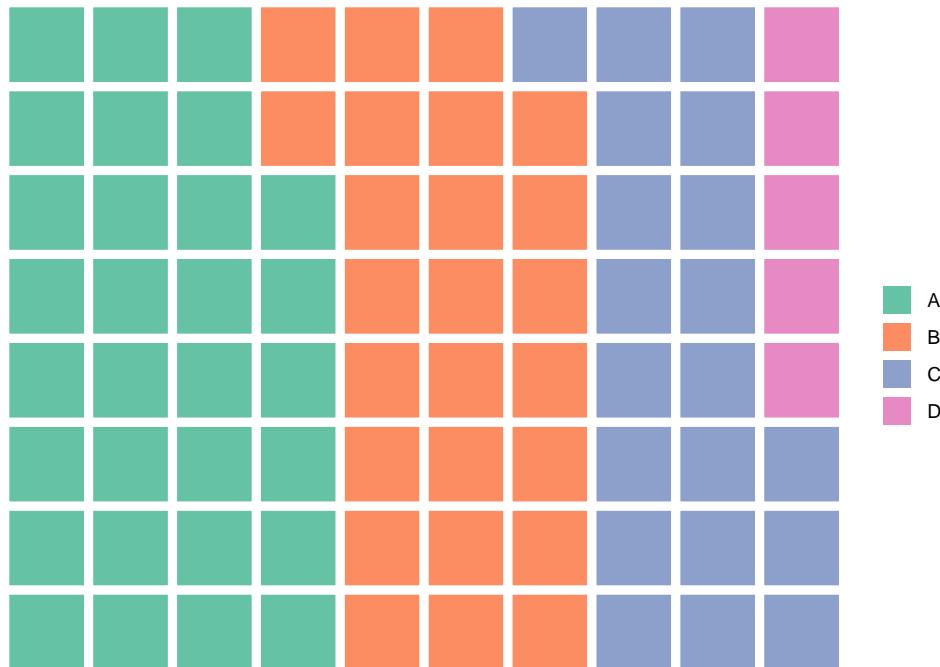
La librería `waffle` contiene una función del mismo nombre que puede ser usada para crear diagramas de waffle (square pie o gridplots) en `ggplot2`.

Para crear un diagrama de waffle básico se debe **pasar el vector que contiene la cuenta para cada grupo a la función**. El número de filas del diagrama puede ser seleccionado con `rows` (por defecto es 10).

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

# Vector
x = c(30, 25, 20, 5)

# Gráfico de waffle
waffle(x, rows = 8)
```



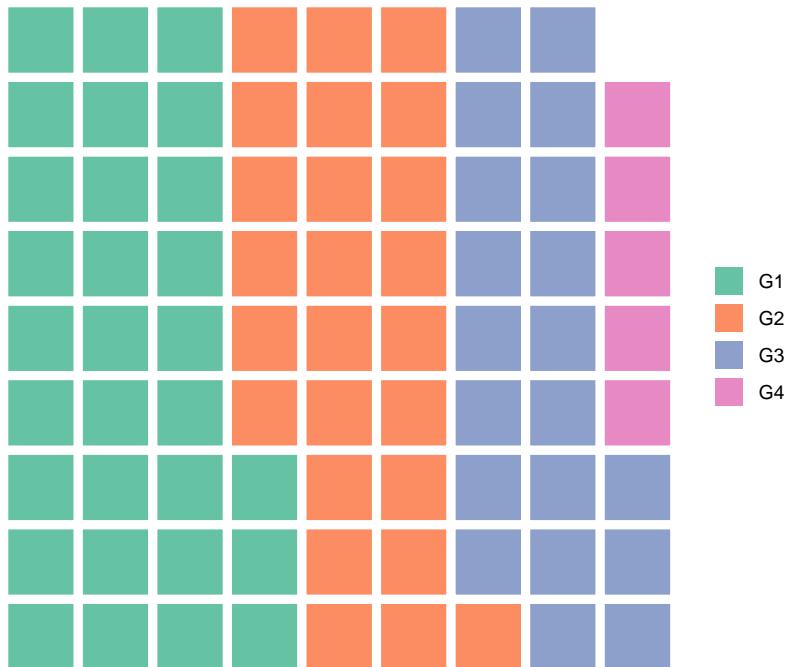
14.4.2 USA UN VECTOR CON NOMBRES PARA CAMBIAR LA LEYENDA

Si se desea nombrar las variables del vector, la leyenda mostrará sus nombres.

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

# Vector
x = c(G1 = 30, G2 = 25, G3 = 20, G4 = 5)

# Gráfico de waffle
waffle(x, rows = 9)
```



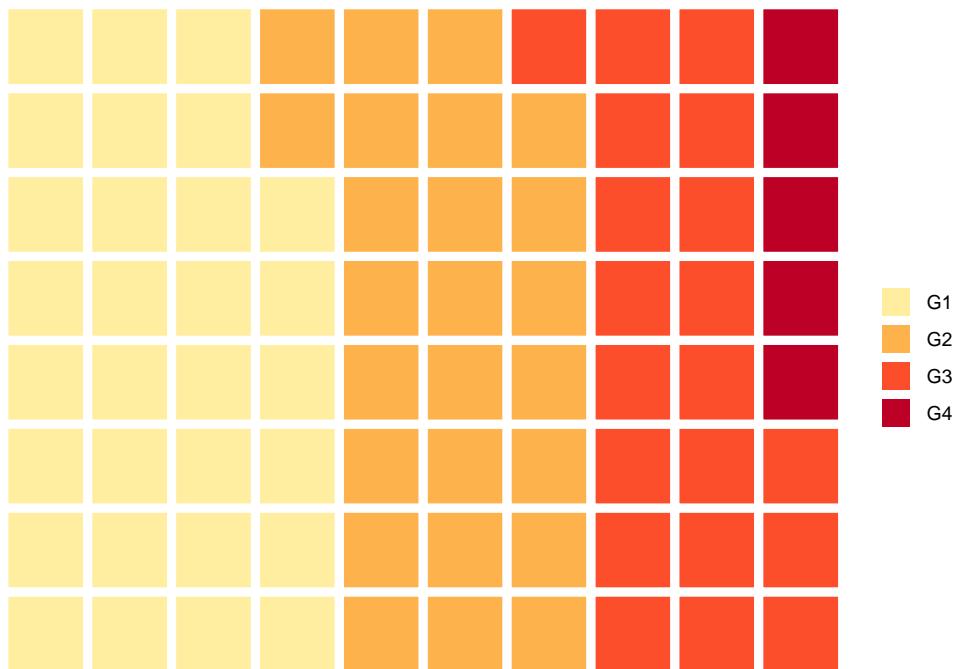
14.4.3 PERSONALIZACIÓN DEL COLOR

Se puede pasar un vector de colores al argumento `colors`. Se puede pasar tantos colores como número de componentes tenga el vector de entrada.

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

# Vector
x = c(G1 = 30, G2 = 25, G3 = 20, G4 = 5)

# Gráfico de waffle
waffle(x, rows = 8,
       colors = c("#FFEDAO", "#FEB24C", "#FC4E2A", "#BD0026"))
```



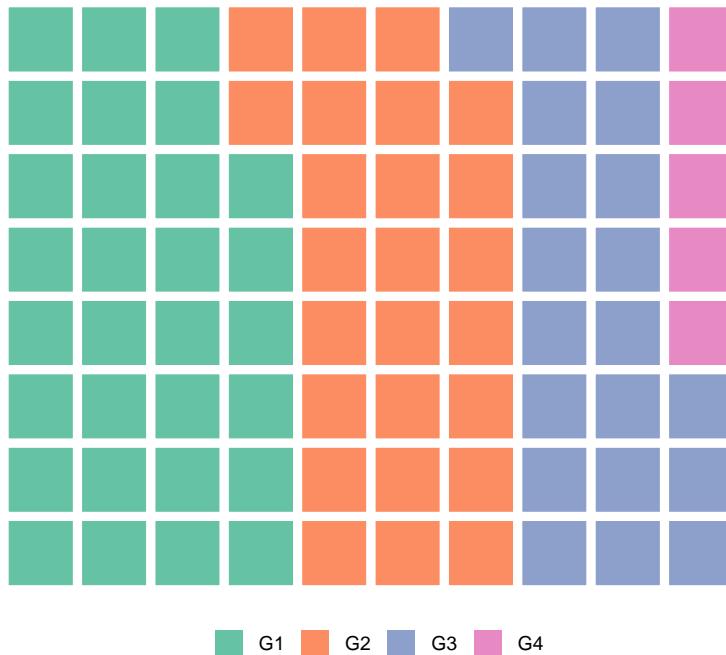
14.4.4 POSICIÓN DE LA LEYENDA

La posición de la leyenda se puede cambiar con el argumento `legend_pos`. Los posibles valores son "right" (por defecto), "bottom", "left", "top", y "none" para eliminar la leyenda.

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

# Vector
x = c(G1 = 30, G2 = 25, G3 = 20, G4 = 5)

# Gráfico de waffle
waffle(x, rows = 8,
       legend_pos = "bottom")
```



En caso de que se desee incrementar el margen del diagrama respecto a la leyenda, se pasa un valor numérico al argumento `pad` (por defecto es 0).

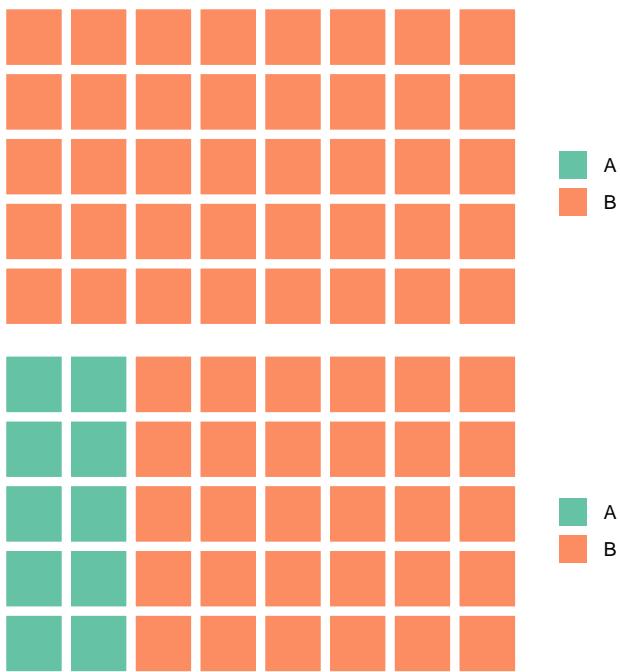
14.4.5 COMBINANDO DIAGRAMAS

El paquete también proporciona una función llamada `iron()` que puede ser usada para **combinar varios diagramas de waffle**, como en el siguiente ejemplo:

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

w1 = waffle(c(A = 0, B = 40), rows = 5)
w2 = waffle(c(A = 10, B = 30), rows = 5)

# Combinar los gráficos
iron(w1, w2)
```



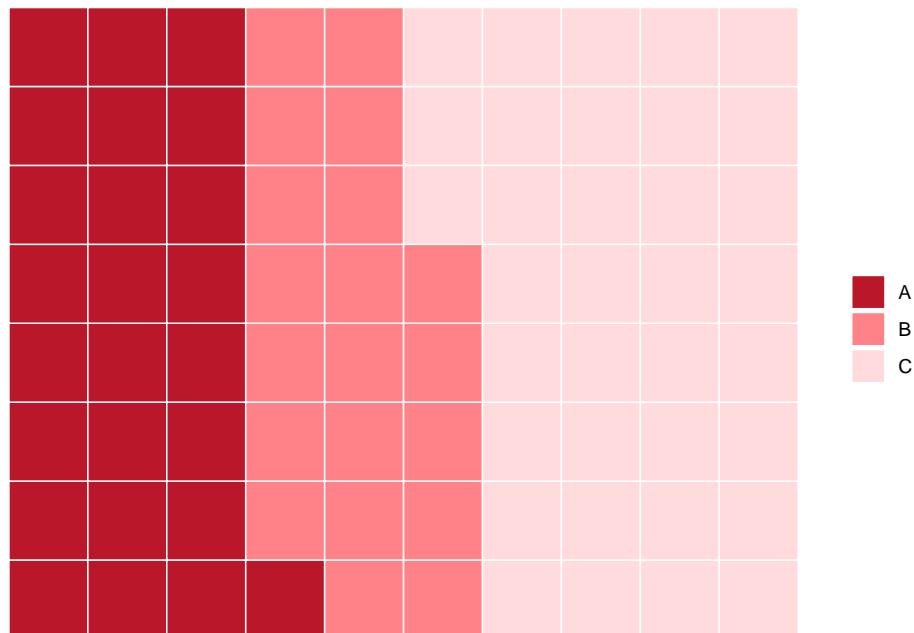
14.4.6 USANDO `geom_waffle()`

Si se prefiere utilizar `geom` en lugar de un *wrapper*, el paquete proporciona la función `geom_waffle()`, que puede ser utilizada y como se muestra en el siguiente ejemplo. Hay que tener en cuenta que `scale_fill_manual()` se puede usar para sobrescribir los colores o para personalizar la leyenda.

```
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)

# Datos
df = data.frame(grupo = LETTERS[1:3],
                 valor = c(25, 20, 35))

# Gráfico waffle
ggplot(df, aes(fill = grupo, values = valor)) +
  geom_waffle(n_rows = 8, size = 0.33, colour = "white") +
  scale_fill_manual(name = NULL,
                    values = c("#BA182A", "#FF8288", "#FFDBDD"),
                    labels = c("A", "B", "C")) +
  coord_equal() +
  theme_void()
```



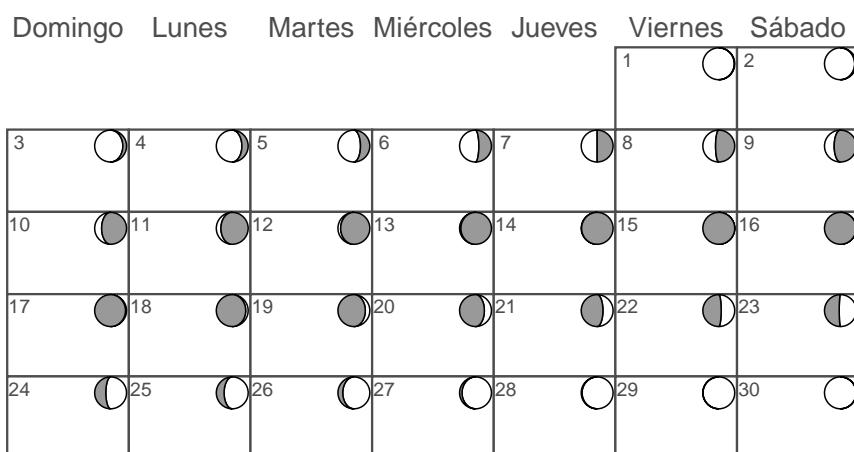
15. CALENDARIO LUNAR CON ggplot2

El paquete `calendR` permite crear calendarios con fases lunares cuando se especifica un mes y se establece el argumento `lunar = TRUE`.

```
# install.packages("calendR")
library(calendR)

calendR(year = 2023,
        month = 9,
        lunar = TRUE)
```

SEPTIEMBRE 2023



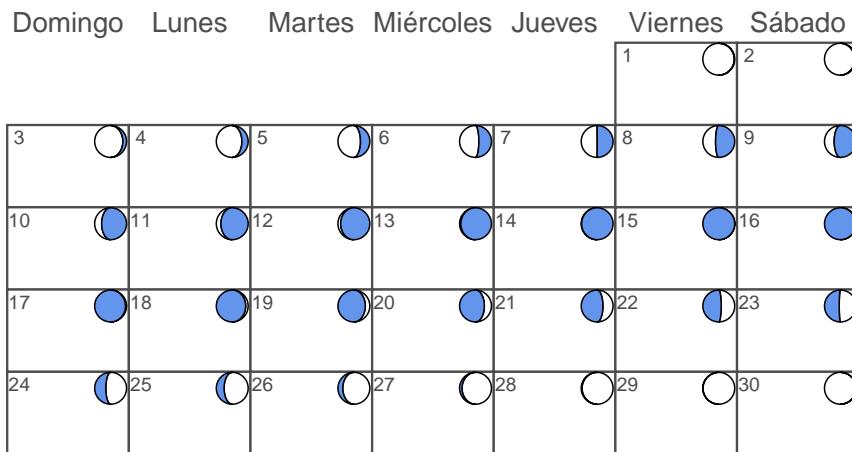
15.1 COLOR DE LAS LUNAS

El color de la parte no visible de las lunas es gris por defecto, pero puedes cambiarlo con el argumento `lunar.col`:

```
# install.packages("calendR")
library(calendR)

calendR(year = 2023,
        month = 9,
        lunar = TRUE,
        lunar.col = "cornflowerblue")
```

SEPTIEMBRE 2023



15.2 TAMAÑO DE LAS LUNAS

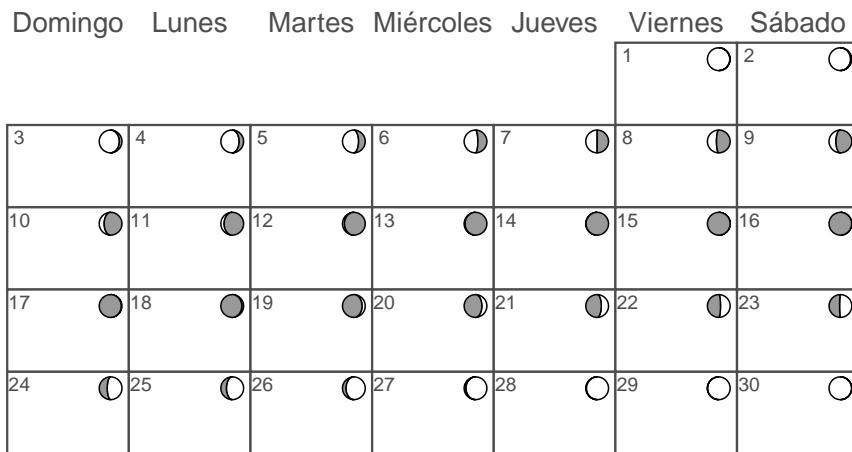
El tamaño de las lunas también se puede modificar. Para ello, se puede usar el argumento `lunar.size`, que por defecto es 7.

```
# install.packages("calendR")
library(calendR)

calendR(year= 2023,
        month = 9,
```

```
lunar = TRUE,  
lunar.size = 5)
```

SEPTIEMBRE 2023



16. BIBLIOGRAFÍA

1. “*ggplot2: Elegant Graphics for Data Analysis*” por Hadley Wickham (Editorial Springer, 2016).
2. “*R Graphics Cookbook*” por Winston Chang (Editorial O'Reilly, 2013).
3. “*Data Visualization with ggplot2*” por Hadley Wickham (Editorial Springer, 2016).
4. “*The Grammar of Graphics*” por Leland Wilkinson (Editorial Springer, 2005).
5. “*Fundamentals of Data Visualization*” por Claus O. Wilke (Editorial O'Reilly, 2019).

■ Páginas web:

1. Documentación oficial de ggplot2 en R: <https://ggplot2.tidyverse.org/>
2. Galería de gráficos ggplot2: <https://ggplot2.tidyverse.org/gallery/>
3. R Graph Gallery: <https://www.r-graph-gallery.com/ggplot2-package.html>
4. Cookbook for R: <http://www.cookbook-r.com/Graphs/>
5. DataCamp: <https://www.datacamp.com/community/tutorials/ggplot2-tutorial-data-visualization-in-r>
6. R-bloggers: <https://www.r-bloggers.com/?s=ggplot2>
7. The R Project for Statistical Computing: <https://www.r-project.org/>
8. Stack Overflow: <https://stackoverflow.com/questions/tagged/ggplot2>
9. GitHub: <https://github.com/tidyverse/ggplot2>

