Introducción a R y RStudio

Visualización de datos

Índice

Ideas clave

6.1. Introducción y objetivos

Graficar nuestros datos es una de las mejores maneras para explorarlos y para observar las relaciones que existen entre las variables.

R cuenta en su paquete base con múltiples funciones para la producción de gráficas, pudiendo generar representaciones en forma de nubes de puntos, líneas, barras, gráficos circulares, etc. También tenemos funciones para elaborar histogramas y curvas de densidad. Esos gráficos, además de ser útiles como vía de exploración de los datos, pueden ser almacenados para su posterior reutilización en cualquier tipo de documento.

Las funciones gráficas de R suelen ser genéricas y tienen un gran número de opciones. Por lo tanto, aquí nos limitaremos a comentar sus usos más comunes.

Al finalizar este tema, esperamos alcanzar los siguientes objetivos:

- Adquirir habilidades en la realización gráficas sencillas de acuerdo con el tipo de dato a graficar.
- Crear gráficos de barras, histogramas, diagramas de dispersión, diagramas de líneas y diagramas de caja con funcionalidades de R base y paquetes relacionados, como ggplot2.
- Entender la gramática básica de los gráficos, incluyendo estética y capas geométricas, agregando estadísticas, transformando las escalas y los colores, o dividiendo por grupos.

6.2. La función `plot`

La función `plot` es una de las funciones más usadas para generar gráficos en R. Se trata de una función genérica, por lo que puede ser invocada para representar gráficamente distintos tipos de datos.

Dada una muestra de puntos $(x_1, y_1), \ldots, (x_n, y_n)$, podemos usar la función `plot` para obtener su gráfico. La construcción básica para hacerlo es simplemente

```
plot(x,y)
```

donde `x` e `y` son los vectores que contienen los datos $x_1, \ldots, x_n$ y $y_1, \ldots, y_n$ respectivamente.

Esto parece muy sencillo ¿verdad? Veámoslo en un ejemplo.

Ejemplo: mi primer gráfico

Para ejemplificar el uso de la función `plot`, vamos a crear dos vectores, x e y, para luego obtener la gráfica de estos.

```
x <- c(1, 3, -2, 0, 5, 4, 2.5, 1)
y <- c(0, 5, 7, 4, 1, -1, -3, 9)
plot(x, y)
```

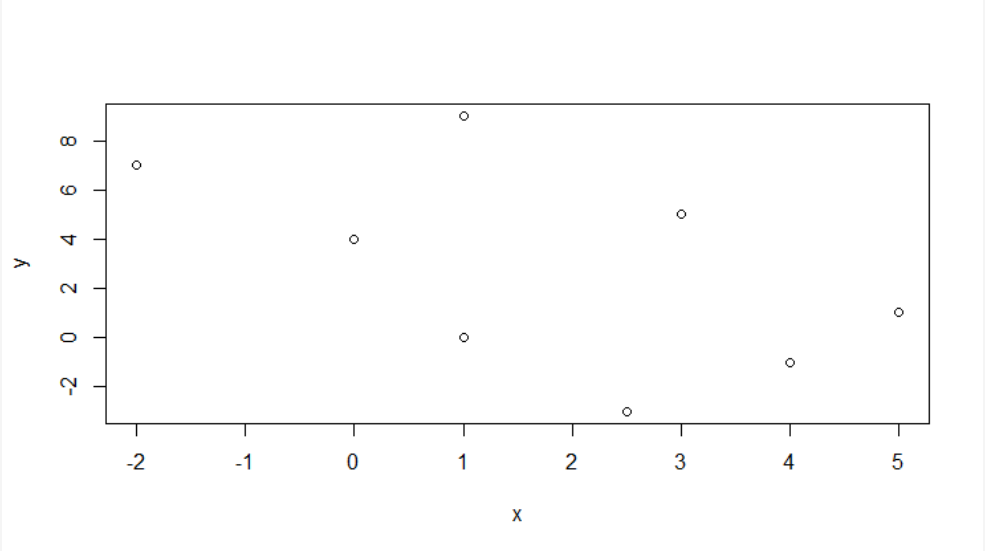


Figura 1: La función `plot` para grafico de puntos.

Y tenemos nuestro primer gráfico en R.

Como podemos observar, `plot()` dibuja cada par de puntos considerando como primera coordenada al primer vector que pasamos como argumento (`x` en el ejemplo) y como segunda coordenada a los elementos del segundo vector que le pasamos (`y` en el ejemplo).

Si llamamos a función aplicándola a un único vector, digamos $x = (x_1, \dots, x_n)$, entonces R construirá el gráfico correspondiente a los pares de puntos $(1, x_1), \dots, (n, x_n)$. Es decir, `plot(x)` es lo mismo que `plot(1:n, x)`.

La función `plot()` también sirve para dibujar el gráfico de una función definida mediante `function`. Veamos como ejemplo, cómo podemos obtener el gráfico de la función $f(x) = x^2 + 1$, que sabemos corresponde a una parábola.

Ejemplo: gráfico de una función.

```
f <- function(x){x^2 +1}
plot(f)
```

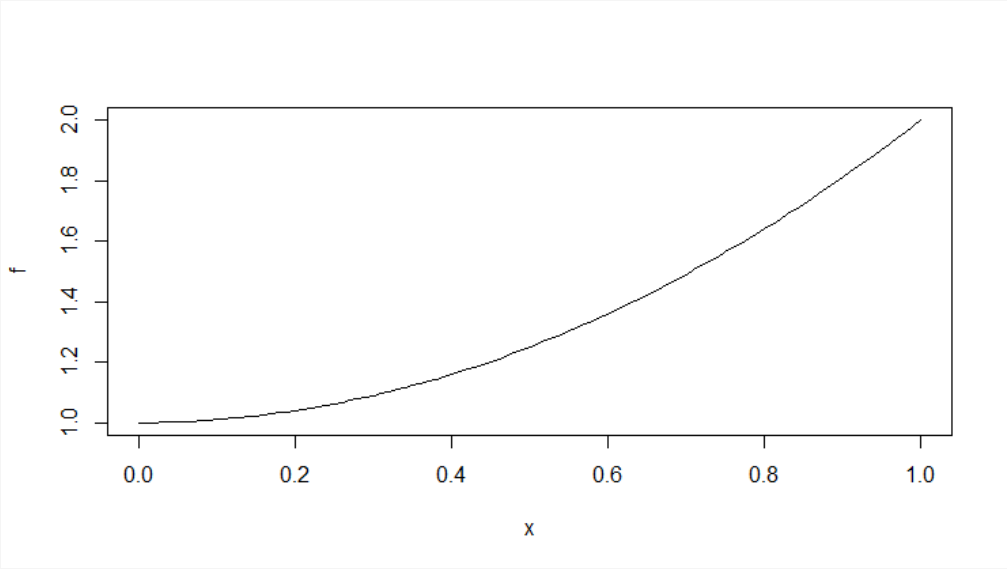


Figura 2: La función `plot` para el gráfico de una función.

Por defecto el rango del eje de abscisas es el intervalo [0, 1] y el rango del eje de ordenadas va del valor mínimo de f sobre el rango de las abscisas al máximo. Podemos especificar desde donde y hasta donde queremos que grafique la función, por ejemplo

```
plot(f, -1, 4)
```

realizará el gráfico de `f` entre los valores `x = -1` y `x = 4`.

Bien, probablemente a esta altura estarán pensado ¿no se puede hacer algo mejor? ¿Estéticamente más agradable? La respuesta, por supuesto, es sí. Para ello veamos los argumentos que podemos agregarle a la función `plot` para personalizar los gráficos.

6.2.1. Argumentos de la función `plot`

Esta función tiene múltiples argumentos para configurar el gráfico final, que permiten agregar un título, cambiar las etiquetas de los ejes, personalizar colores, cambiar tipos de línea, etc.

Veamos los más usados, mostrando su uso con sendos ejemplos.

Argumento `type`

Este argumento nos permite personalizar el tipo de gráfico que se dibuja. La selección del **tipo dependerá de los datos** que estés representando. Veamos algunos ejemplos con los tipos de gráficos más comunes que se pueden crear con la función `plot()` en R.

Ejemplo: especificando el argumento `type`.

```
# argumento type
x <- 0:10
y = dbinom(x, size = 10, p = 0.3)

par(mfrow = c(1, 3))

plot(x, y, type = "p", main = "type = 'p'")
plot(x, y, type = "l", main = "type = 'l'")
plot(x, y, type = "h", main = "type = 'h'")

plot(x, y, type = "o", main = "type = 'o'")
plot(x, y, type = "b", main = "type = 'b'")
plot(x, y, type = "c", main = "type = 'c'")

plot(x, y, type = "s", main = "type = 's'")
plot(x, y, type = "S", main = "type = 'S'")
plot(x, y, type = "n", main = "type = 'n'")

par(mfrow = c(1, 1))
```

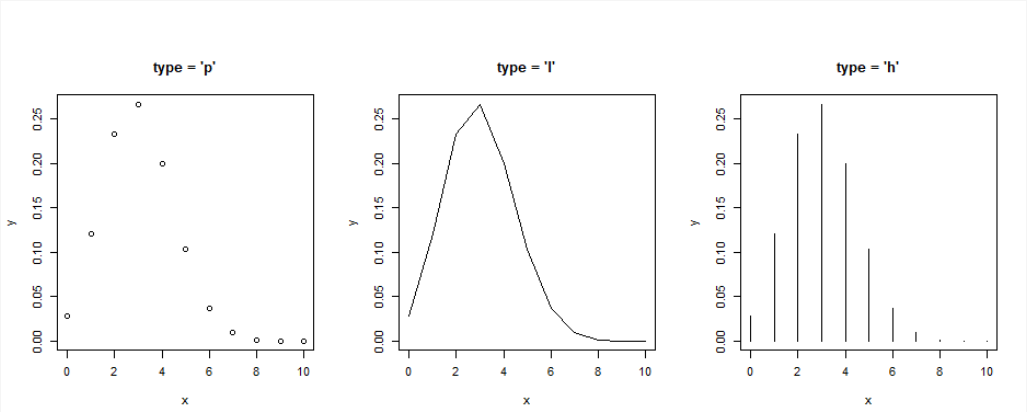


Figura 3: Distintos tipos posibles especificando argumento `type`.

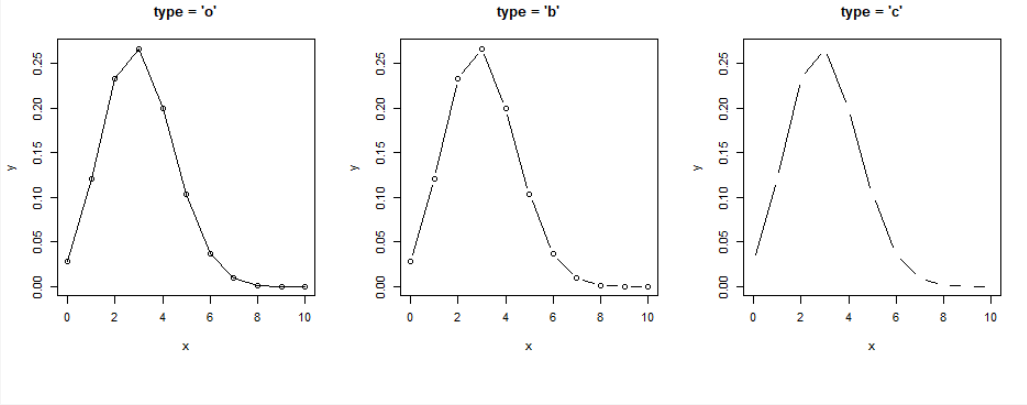


Figura 4: Distintos tipos posibles especificando argumento `type`.

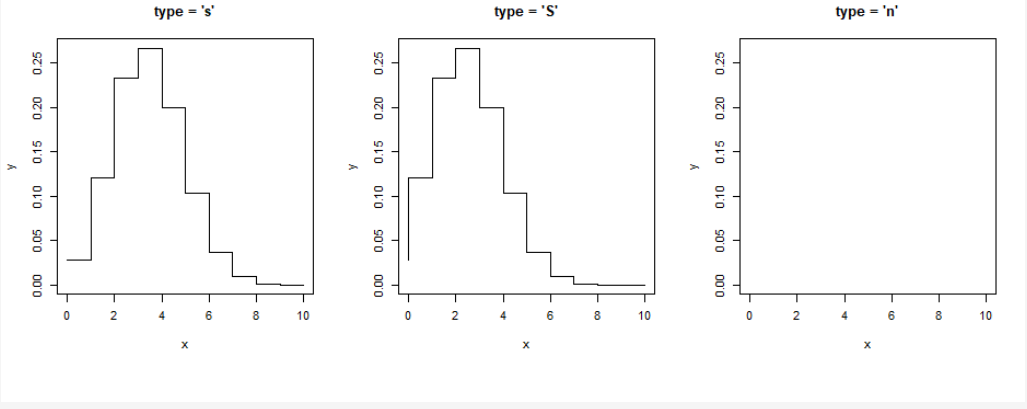


Figura 5: Distintos tipos posibles especificando argumento `type`.

En la Tabla 1 resumimos los posibles valores de `type`.

Valor en type	Descripción
p	Gráfico de puntos (es la opción por defecto)
l	Gráfico de líneas
h	Gráfico estilo histograma
o	Ambos (líneas y puntos encima)
b	Ambos (puntos y líneas separados)
c	sola la parte de líneas de "b"
s	Gráfico de escalones
S	Otro tipo de gráfico de escalones
n	Gráfico vacío

Tabla 1: valores posibles para el argumento type.

Modificar los puntos del gráfico

El argumento `pch` permite seleccionar el símbolo utilizado para representar los puntos en el gráfico. Los símbolos principales se pueden seleccionar pasando los números 0 a 25 como parámetros (ver Figura 6). También podemos cambiar el tamaño de los símbolos con el argumento `cex`, el color especificando `col`. Los puntos que se obtienen con `pch` de 21 a 25 admiten un color para el borde (que se especifica con `col`) y uno de diferente para el relleno, que se especifica con `bg` (ver Figura 7). El ancho de los bordes de los símbolos (excepto los símbolos `15` a `18`) pueden modificarse con el argumento `lwd`.

```
r <- rep(seq(4, 28, by = 6), each = 5)
r <- c(r, 34)
t <- rep(seq(25, 5, -5), 5)
t <- c(t, 25)

plot(r, t, pch = 0:25, cex = 3, yaxt = "n", xaxt = "n",
     ann = FALSE, xlim = c(1, 36), ylim = c(3, 27), lwd = 1:3,
     bg = "grey")
text(r - 2.7, t, 0:25, cex = 0.9)

r <- r[-26]
t <- t[-26]
plot(r, t, pch = 21:25, cex = 3, yaxt = "n", xaxt = "n", lwd = 3,
```

```
     ann = FALSE, xlim = c(1, 30), ylim = c(3, 27), bg = 1:25,
     col = rainbow(25))
```

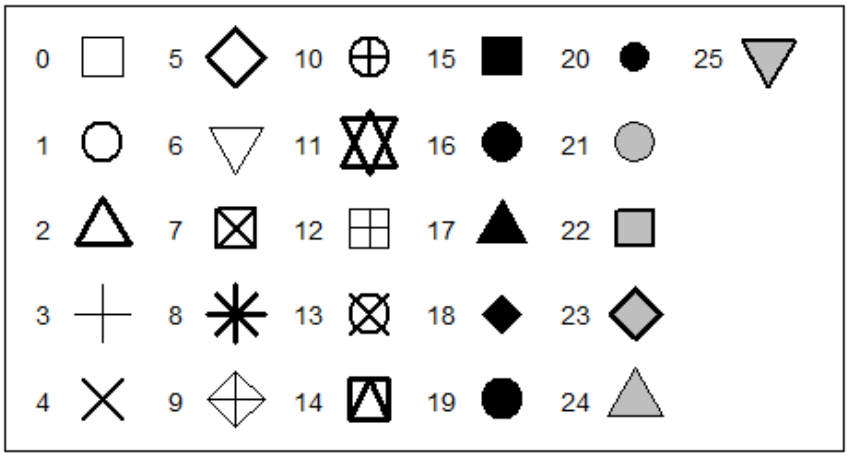


Figura 6: Símbolos principales para puntos del gráfico.

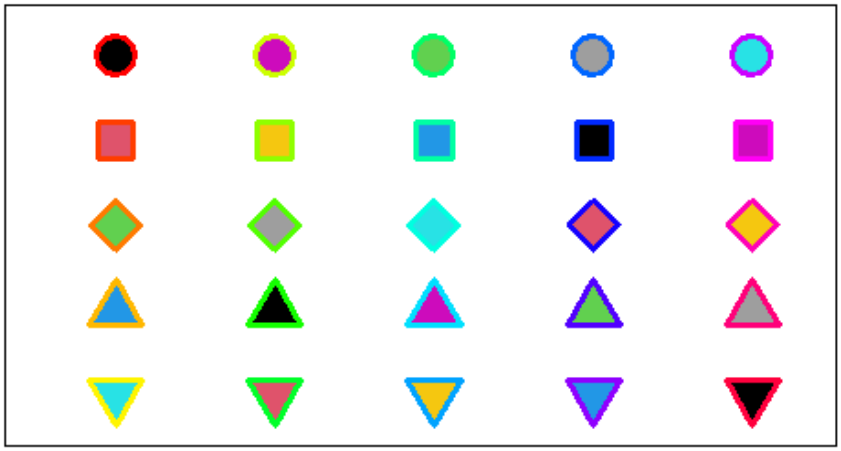


Figura 7: Símbolos del 21 al 25, personalizando color de borde y de fondo.

Probemos a personalizar el grafico de puntos que hicimos al comienzo de la sección.

Ejemplo: mi primer gráfico personalizado

Vamos a retomar el ejemplo inicial para mostrar el uso de los argumentos que hemos visto.

```
x <- c(1, 3, -2, 0, 5, 4, 2.5, 1)
y <- c(0, 5, 7, 4, 1, -1, -3, 9)
plot(x, y,
  pch = 21,   # tipo de punto
  col = "#0098CD",  # color de borde
  bg = "#B8E5F5",   # color de fondo
  cex = 1.5,    # tamaño del punto
  lwd = 2   # ancho del borde
)
```

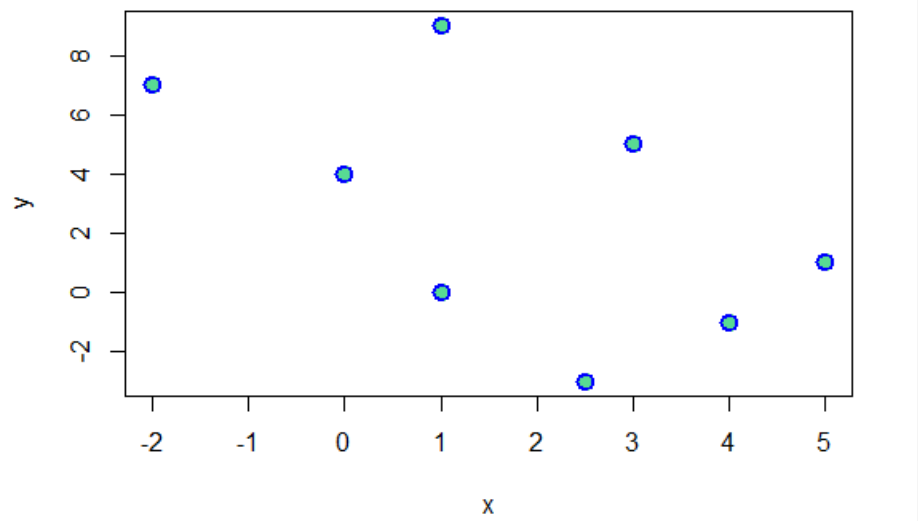


Figura 8: Gráfico de puntos con algunos elementos de personalización

Como podemos observar del código, podemos indicar el color por nombre o por su código hexadecimal (el código con el #). En RStudio podemos utilizar el *addin* `Colour Picker`, disponible con la instalación del paquete `{colourpicker}` desarrollado por Dean Attali. `Colour Picker` no sólo te permite elegir muy fácilmente colores de una paleta con su mouse o el *touch pad*, sino que además exporta directamente sus nombres o sus códigos hexadecimales en tu código R.

Para poder utilizar el addin selector de color, primero tendrás que instalar el paquete `{colourpicker}`. Reccuerda que esto puedas hacerlo ejecutando `install.packages("colourpicker")`, desde la pestaña *Packages* `clicando en Install` o bien, desde el menú de RStudio `Tools -> Install Packages…`

Para acceder al selector de color, clicamos en *Addins* (en la barra de herramientas general de RStudio) y del desplegable que se abre elegimos `Colour Picker` (ver Figura 9).

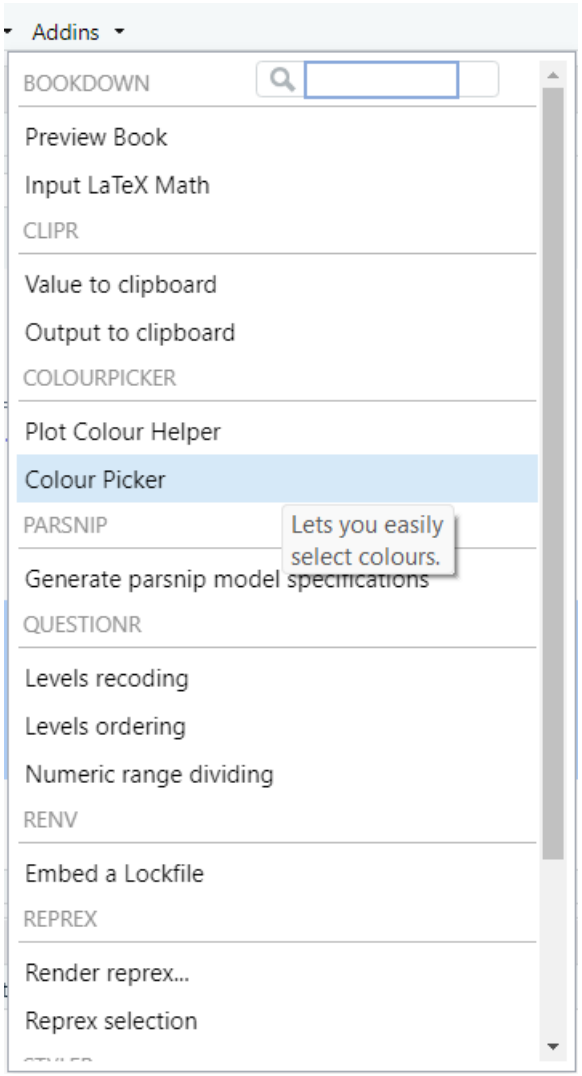


Figura 9: *Addin* (complemento) para seleccionar colores en RStudio.

Al hacer clic en Colour Picker, se abrirá un cuadro de diálogo como el que vemos en la Figura 10. Haciendo clic en el área "Select any colour" (Seleccionar cualquier color) se desplegará una paleta de color. Elegimos el color que nos gusta (haciendo clic sobre la paleta). Si lo deseamos, podemos elegir más colores clicando en "2" y repitiendo el procedimiento anterior. Una vez seleccionados todos los colores que queremos, haciendo clic en Done (hecho), se importaran a la consola (o al script, dependiendo de donde se encuentre el cursor) los códigos hexadecimales de los colores elegidos.

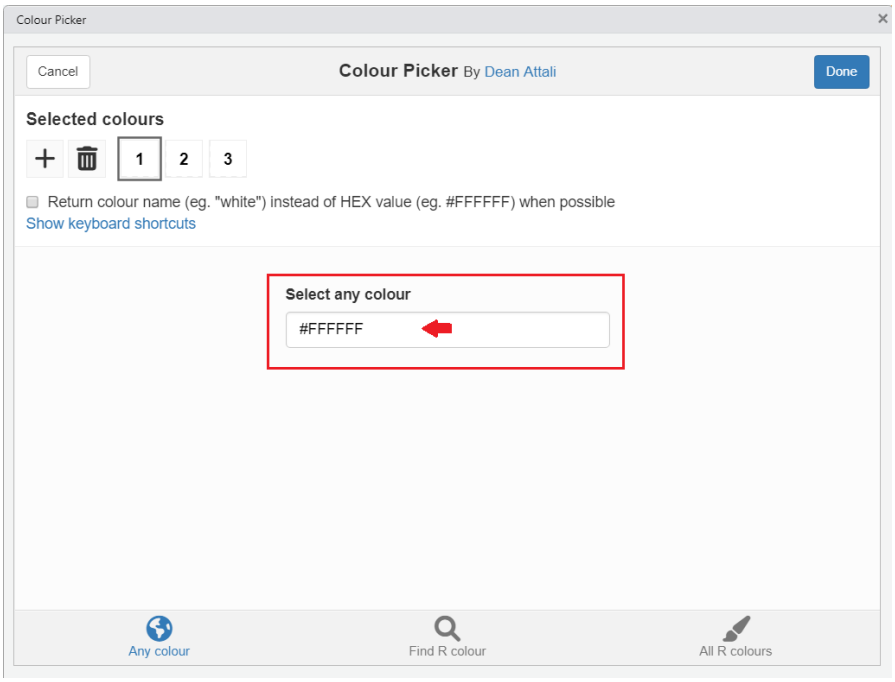


Figura 10: Área de selección de color.

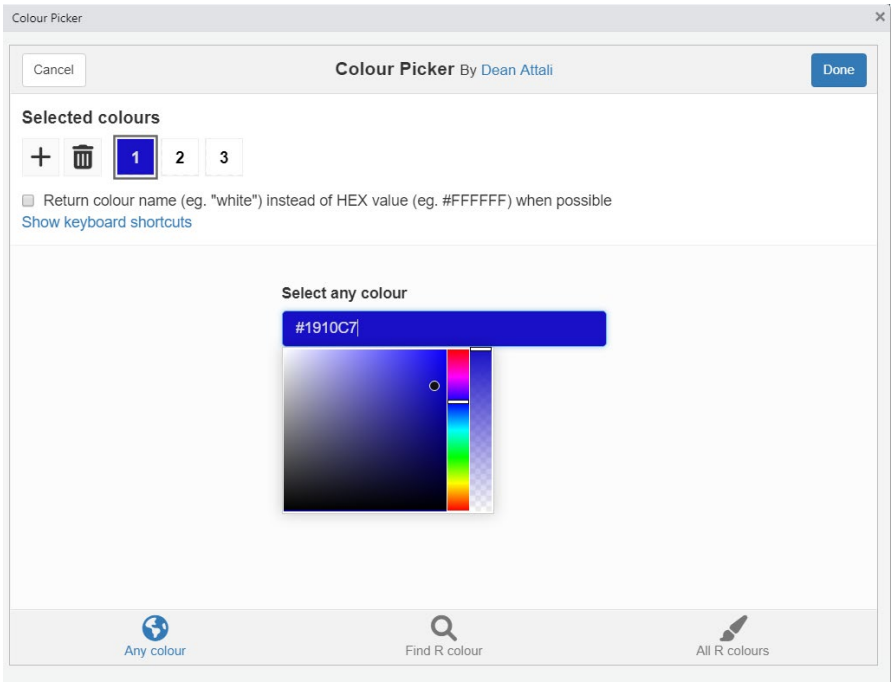


Figura 11: Selección del color desde la paleta de colores.

Tipos de líneas

Cuando especificamos como argumento `type = "l"`, podemos también escoger el tipo de trazo que se empleará en el gráfico. Existen siete tipos diferentes de líneas,

que pueden ser especificadas haciendo uso del argumento `lty` igual a un número (0, 1, 2, 3, 4, 5, o 6) o una palabra (`"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, `"twodash"`). Por ejemplo, `lty = "solid"` es lo mismo que `lty = 1`.

Además, podemos utilizar el argumento `lwd` para modificar el grosor de la línea. Puede tomar cualquier valor positivo y el valor por defecto es igual a 1.

Ejemplo: Sobre como modificar el aspecto de las líneas en un gráfico.

```
# tipos de líneas
par(mfrow = c(3,3))
# Línea solida (por defecto)
plot(1:10, 1:10, type="l", main = 'lty = 1')
# Línea discontinua
plot(1:10, 1:10, type="l", lty=2, main = 'lty = 2')
# Línea de puntos
plot(1:10, 1:10, type="l", lty=3, main = 'lty = 3')
# Línea de puntos y guiones cortos
plot(1:10, 1:10, type="l", lty=4, main = 'lty = 4')
# Línea de guiones largos
plot(1:10, 1:10, type="l", lty=5, main = 'lty = 5')
# Línea de guiones cortos y largos
plot(1:10, 1:10, type="l", lty=6, main = 'lty = 6')
# modificando el grosor
plot(1:10, 1:10, type="l", lty=1, lwd = 2, main = 'lty = 1 y lwd = 2')
# modificando el grosor
plot(1:10, 1:10, type="l", lty=1, lwd = 2.5, main = 'lty = 1 y lwd =
2.5')
# modificando el grosor
plot(1:10, 1:10, type="l", lty=1, lwd = 0.5, main = 'lty = 1 y lwd =
0.5')
```

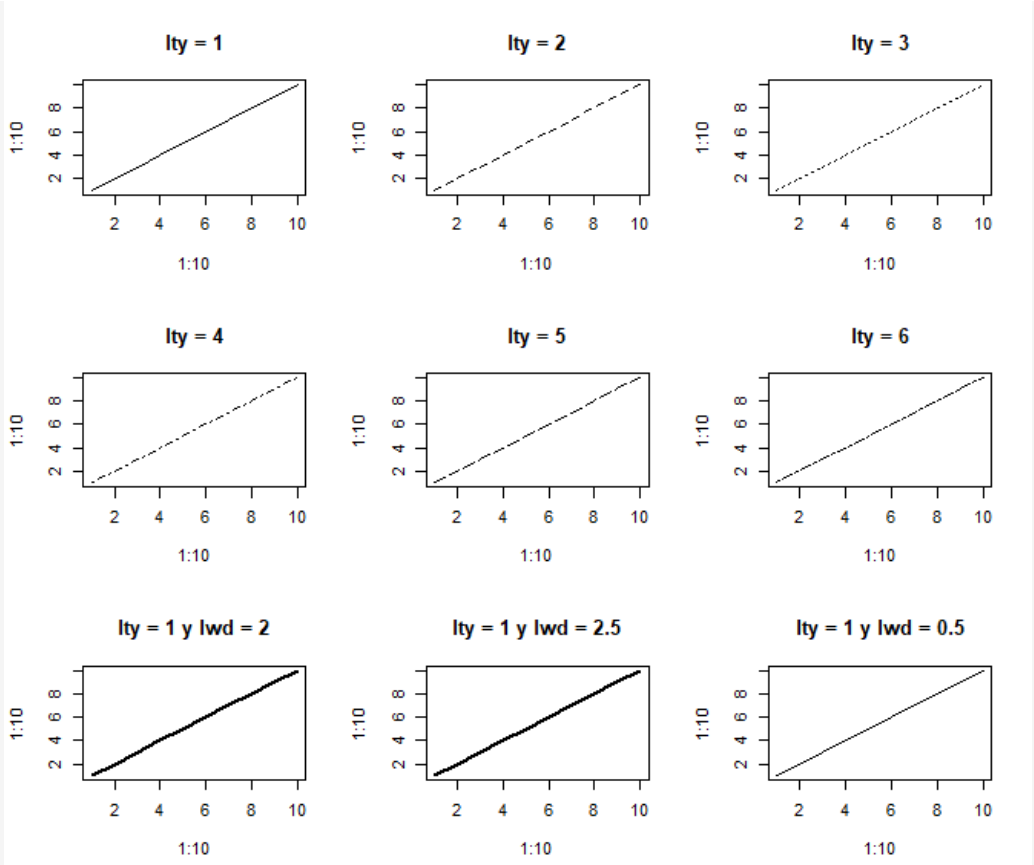


Figura 12: Tipos de líneas de un gráfico.

Título y nombre de los ejes

Veamos cómo hacemos para modificar agregar título a un gráfico y para poner nombres o etiquetas a los ejes:

- Para poner un título al gráfico, tenemos que especificarlo con el parámetro `main`.
- Para modificar las etiquetas de los ejes de coordenadas, tenemos que usar los parámetros `xlab` e `ylab`.

Los valores de estos parámetros los tenemos que entrar entre comillas o, si son fórmulas matemáticas, aplicarles la función `expression()`, para que aparezcan en un formato matemático más adecuado.

> Ejemplo: Título y nombre de los ejes
> Exploremos el uso de estos parámetros, realizando la gráfica de una función, en este caso, la función seno.

```
x <- seq(-4, 4, length.out = 101)
y <- sin(x)
plot(x, y, type = "l",
     ylab = expression(italic(sen) ~ '( ' * phi~')'),
     xlab = expression(paste("Angulo de fase ", phi)),
     main = 'El título',
     col.main = "#0098CD"
)
```

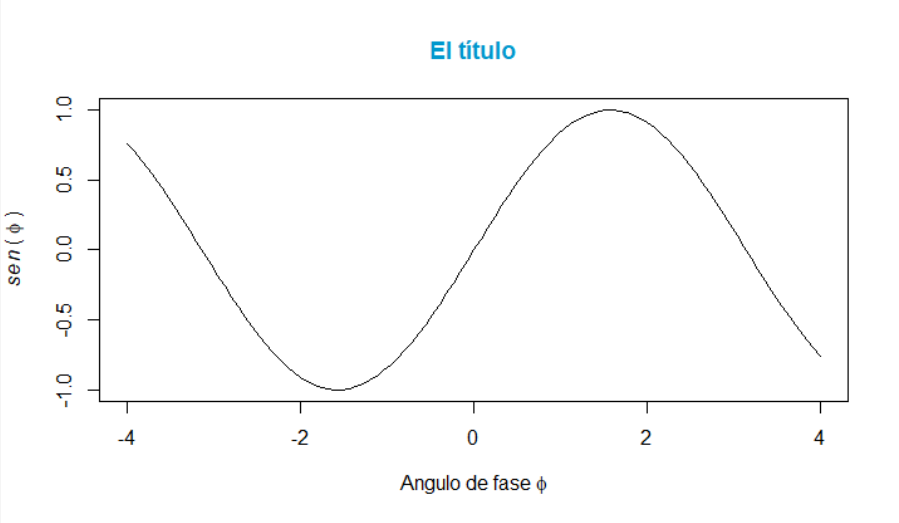


Figura 13: Título y nombre de los ejes, ejemplo 1.

En este ejemplo agregamos el título con a argumento `main = 'El título'`, además le cambamos su color especificando `col.main = "#0098CD"`. Además, le pusimos nombre a los dos ejes con `ylab` y `xlab`. Como habrán notado, utilizamos la función `expression()` para incluir expresiones matemáticas en los nombres que usamos. Para saber más sobre todas las opciones disponibles para utilizar la notación matemática puedes consular la ayuda de `plotmath` ejecutando `?plotmath`.

Función `axis`

Si en la función `plot()` establecemos en `axes = FALSE`, no se dibujarán los ejes, y así podemos agregarlos (o solo uno de ellos) con la función `axis` y personalizarlo/s. Pasar un 1 como argumento dibujará el eje X, pasar un 2 dibujará el eje Y, pasar un 3 agregará el eje superior y un 4 el derecho.

Ejemplo: función `axis()`

```
par(mfrow = c(1,2))
x <- seq(-4, 4, length.out = 101)
y <- sin(x)
plot(x, y, type = "l",
```

```
  axes = FALSE,
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = 'axis(1)',
  col.main = "#0098CD"
)
```

```
# agregamos el eje x
axis(1)

plot(x, y, type = "l",
  axes = FALSE,
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = 'axis(2)',
  col.main = "#0098CD"
)
# agregamos el eje y
axis(2)
```

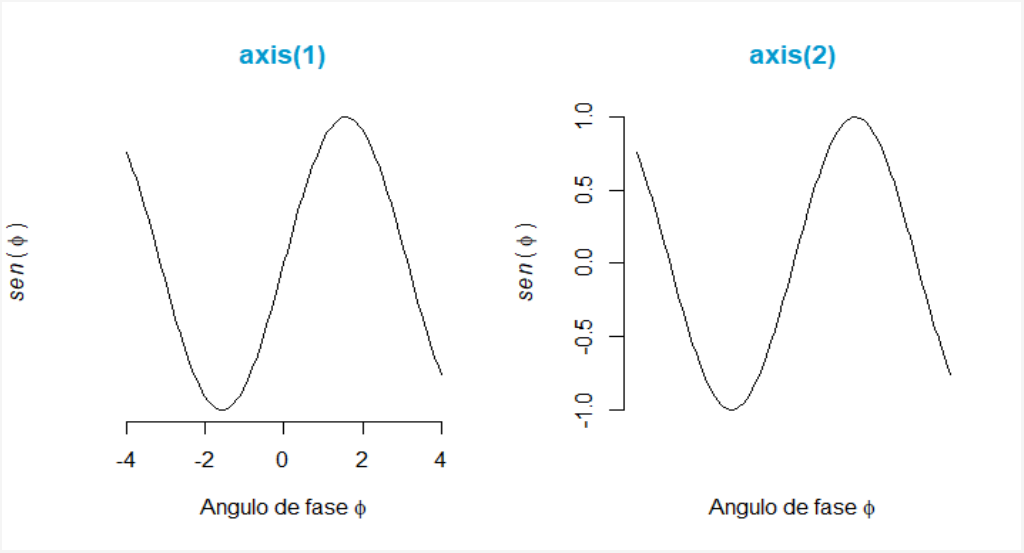


Figura 14: Agregar los ejes con la función `axis()`

Para modificar las posiciones de las marcas (*ticks*) en los ejes de abscisas y ordenadas podemos usar los parámetros `xaxp` e `yaxp`, respectivamente. Mediante la expresión `xaxp = c(a, b, m)`, imponemos que R dibuje m+1 marcas igualmente espaciadas entre los puntos a y b del eje de abscisas. La sintaxis para `yaxp` es la misma. Estas instrucciones no definen los rangos de los ejes de coordenadas, que se han de especificar con `xlim` e `ylim` si se quieren modificar. Veamos tres ejemplos:

Ejemplo: Modificar la posición de las marcas en los ejes

```
# Con las marcas por defecto:
```

```
x <- seq(-4, 4, length.out = 101)
y <- sin(x)
plot(x, y, type = "l",
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = 'El título',
  col.main = "#0098CD"
)
```

Figura 15: las marcas en los ejes por defecto.

```
# Definiendo las marcas sobre el eje de abscisas:
plot(x, y, type = "l",
  xaxp=c(-4, 4, 8),
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = 'El título',
  col.main = "#0098CD"
)
```

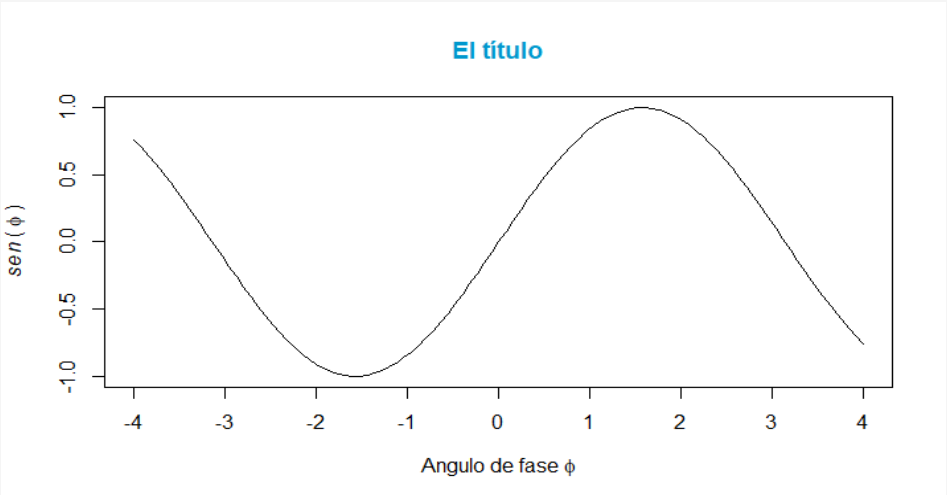


Figura 16: Modificamos las marcas en el eje X.

```
# Definiendo las marcas sobre ambos ejes:
```

```
plot(x, y, type = "l",
  xaxp=c(-4, 4, 8),
  yaxp=c(-1, 1, 5),
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = 'El título',
  col.main = "#0098CD"
)
```

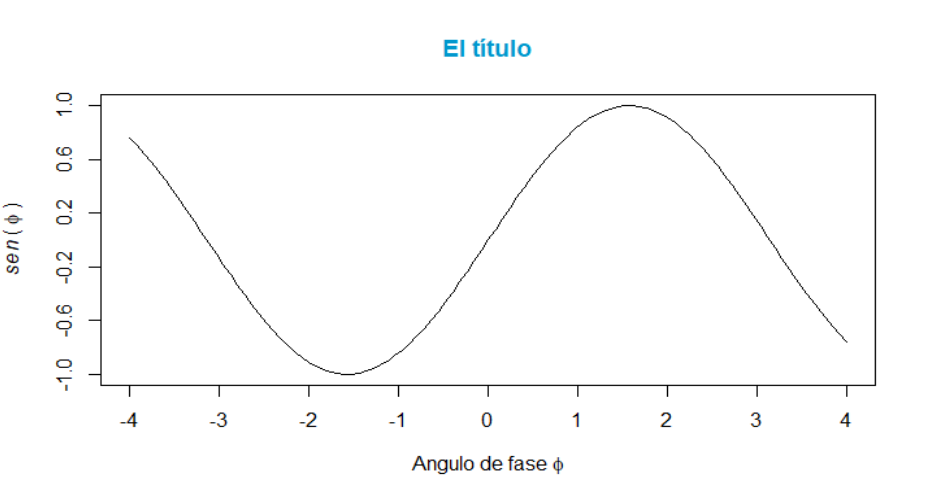


Figura 17: Modificamos las marcas en ambos ejes.

Personalizar las marcas desde la función `axis`

También es posible cambiar los *ticks* de los ejes modificando el argumento `at` de la función `axis()` permite indicar los puntos en los que se van a dibujar las etiquetas.

Ejemplo: modificar las marcas sobre los ejes utilizando la función `axis()`

```
# utilizando la función axis
plot(x, y, type = "l",
  axes = FALSE,
  ylab = expression(italic(sen) ~ '( ' * phi~')'),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = expression('Gráfico de la función'~italic(sen)~'('~phi~')'),
  col.main = "#0098CD"
)

# agregamos el eje x
axis(1, at = seq(-4, 4, length.out = 9))
# agregamos el eje y
axis(2, at = seq(-1, 1, length.out = 6))
```

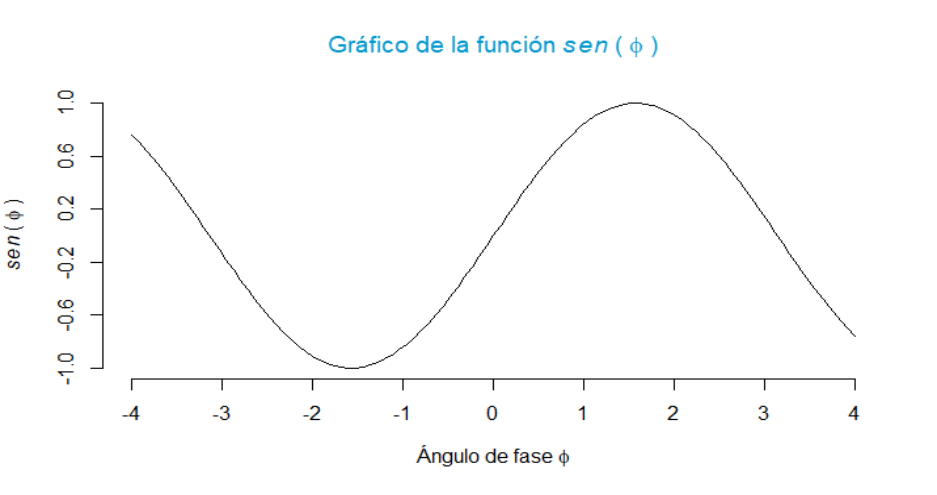


Figura 18: Modificamos las marcas en ambos ejes con axis().

Eliminar etiquetas de las marcas de los ejes

Estableciendo los argumentos `xaxt` o `yaxt` como `"n"` en la función `plot()`, no se dibujarán las marcas (ni sus etiquetas) de los ejes X e Y, respectivamente. Así, podemos luego utilizar la función `axis()`, estableciendo los valores que queremos que a través del argumento `labels`.

Ejemplo: Eliminar las marcas de los ejes con `xaxt` o `yaxt`.

```
# eliminar las marcas
par(mfrow = c(1,3))
plot(x, y,
  type = "l",
  xaxt = "n",
  ylab = expression(italic(sen) ~ "( " * phi ~ ")"),
  xlab = expression(paste("Angulo de fase ", phi)),
  main = "Sin las marcas en eje X",
  col.main = "#0098CD"
)

plot(x, y,
     type = "l",
     yaxt = "n",
     ylab = expression(italic(sen) ~ "( " * phi ~ ")"),
     xlab = expression(paste("Angulo de fase ", phi)),
     main = "Sin las marcas en eje Y",
     col.main = "#0098CD"
)

plot(x, y,
     type = "l",
```

```
     xaxt = "n",
     yaxt = "n",
     ylab = expression(italic(sen) ~ "( " * phi ~ ")"),
     xlab = expression(paste("Angulo de fase ", phi)),
     main = "Sin las marcas en eje X e Y",
     col.main = "#0098CD"
)
```

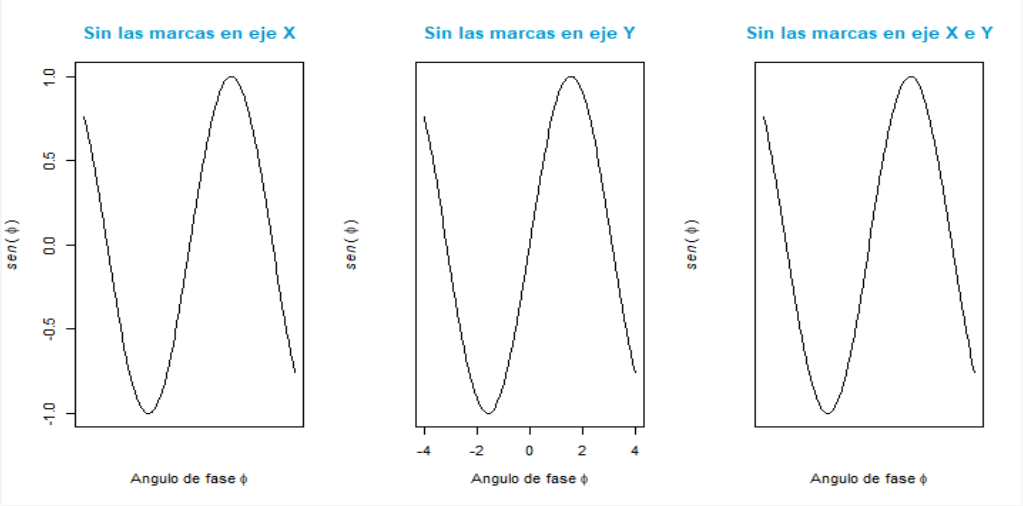


Figura 19: Eliminar las marcas (y sus etiquetas) de los ejes.

Cambiar etiquetas de las marcas de los ejes

Por defecto, las etiquetas de las marcas de los ejes se enumerarán para seguir la numeración de tus datos. Sin embargo, podemos modificar estas etiquetas si es necesario con el argumento `labels` de la función `axis()`. También tendremos que especificar dónde se mostrarán las etiquetas de los *ticks* con el argumento `at`.

Ejemplo: personalizar las etiquetas de los *ticks* en los ejes.

```
plot(x, y,
     type = "l",
     xaxt = "n",
     yaxt = "n",
     ylab = expression(italic(sen) ~ "( " * phi ~ ")"),
     xlab = expression(paste("Angulo de fase ", phi)),
     main = "Perssonalizar las etiquetas de las marcas en los ejes",
     col.main = "#0098CD"
)
axis(1,
  at = c(-pi, -pi / 2, 0, pi / 2, pi),
  labels = expression(-pi, -pi / 2, 0, pi / 2, pi)
)

axis(2,
```

```
     at = c(-1, -0.5, 0, 0.5, 1),
     labels = c(-1, '', 0, '', 1)
)
```

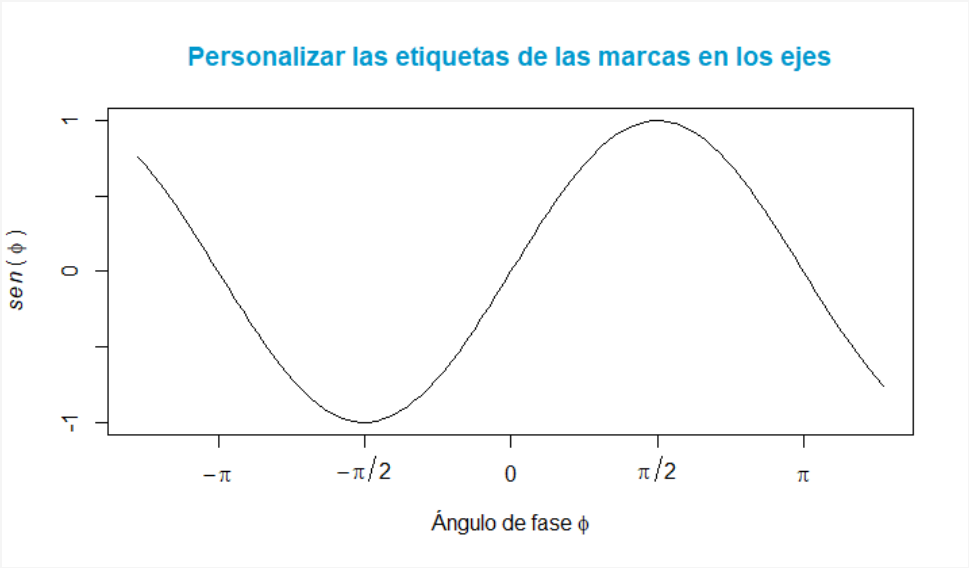


Figura 20: Personalizar las marcas (y sus etiquetas) de los ejes.

Modificar la fuente utilizada en los gráficos.

En muchas ocasiones se nos presentará la necesidad de modificar la fuente utilizada en el título, subtítulo o en las etiquetas de los ejes.

Podemos cambiar el tamaño de fuente de un gráfico de R con los argumentos `cex.main`, `cex.sub`, `cex.lab` y `cex.axis` para cambiar el título, los subtítulos, las etiquetas de los ejes X e Y y los *ticks* de los ejes, respectivamente. Valores mayores mostrarán textos más grandes.

También podemos modificar el estilo de la fuente utilizando el argumento `font`. Podemos especificar el estilo de cada uno de los textos del gráfico con los argumentos `font.main`, `font.sub`, `font.axis` y `font.lab`.

Tamaño del texto	
Argumento	Descripción
`cex.main`	Establece el tamaño del título
`cex.sub`	Establece el tamaño del subtítulo
`cex.lab`	Establece el tamaño de las etiquetas de los ejes
`cex.axis`	Establece el tamaño de las etiquetas de los ticks de los ejes

Tabla 2: Argumentos para cambiar tamaño de texto

Estilo de texto	
Valor en `font`	Descripción
1	Texto plano
2	Negrita
3	Cursiva
4	Negrita y cursiva

Tabla 3: Valores para argumento `font` para cambiar estilo de texto

Ejemplo: Cambiar tamaño y estilo de los textos en el gráfico

```
plot(x, y,
  type = "l",
  ylab = "Eje Y",
  xlab = "Eje X",
  main = "Título",
  sub = "Subtítulo",
  col.main = "#0098CD",
  col.sub = "#DB8100",
  col.lab = "#00566C",
  cex.axis = 0.8, # tamaño ticks en los ejes
  cex.lab = 1, # tamaño de las etiquetas de los ejes
  cex.main = 2, # tamaño texto del título
  cex.sub = 1.2, # tamaño del subtítulo
  font.main = 1, # Estilo de fuente del título
  font.sub = 2, # Estilo de fuente del subtítulo
  font.axis = 3, # Estilo de fuente en los ejes (ticks)
  font.lab = 4 # Estilo de fuente en los ejes (etiquetas)
)
```

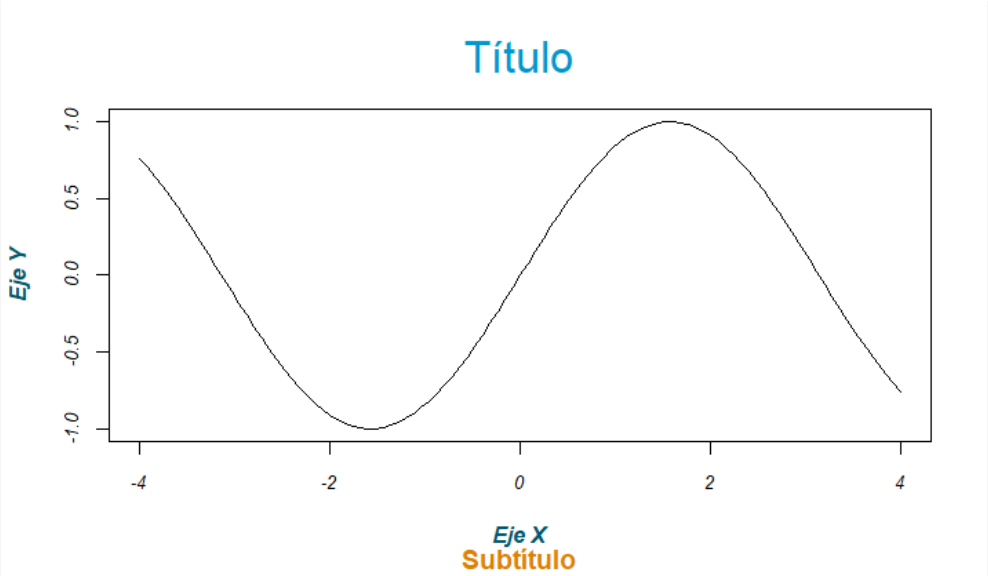


Figura 21: Personalizar los textos en el gráfico.

En el último ejemplo hemos hecho uso, además, de los argumentos `col.main`, `col.sub`, `col.lab` y `col.axis` que nos permiten cambiar el color del título, subtitulo, etiquetas de los ejes y marcas y sus etiquetas, respectivamente.

Cambiar el tipo y color de la caja exterior

El argumento `bty` permite cambiar el tipo de caja de los gráficos de R. Hay diversas opciones, resumidas en la Tabla 4:

Valor en `bty`	Descripción
"o"	Caja entera (valor por defecto)
"7"	Arriba y derecha
"L"	Abajo e izquierda
"U"	Izquierda, abajo y derecha
"C"	Arriba, izquierda y abajo.
"n"	Suprime la caja

Tabla 4: Valores posible para argumento `bty`.

También podemos utilizar el argumento `fg` para cambiar el color de la caja.

También es posible usar la función `box()` para cambiar el aspecto de la caja alrededor del gráfico. En el ejemplo, utilizamos los argumentos `which`, para especificar si la caja encierra al gráfico o al área gráfica; `col`, para especificar el color de la caja, `lwd`, para indicar el grosor de las líneas (los lados) de la caja y `lty` para elegir el tipo de línea que se dibuja.

```
par(mfrow = c(3, 3))
plot(x, y, type = "l", bty = "o", main = "Por defecto")
plot(x, y, type = "l", bty = "7", main = "bty = '7'")
plot(x, y, type = "l", bty = "L", main = "bty = 'L'")
plot(x, y, type = "l", bty = "U", main = "bty = 'U'")
plot(x, y, type = "l", bty = "C", main = "bty = 'C'")
plot(x, y, type = "l", bty = "n", main = "bty = 'n'")
plot(x, y, type = "l", bty = "o", fg = "#A5007B", main = "Color con
fg")
plot(x, y, type = "l", bty = "n", main = "Caja con box")
box(which = "plot", col= "#A5007B")
plot(x, y, type = "l", bty = "n", main = "Caja con box")
box(which = "plot", col= "#A5007B")
box(which = "figure", col= "#007BA5", lwd = 2, lty = 3)

par(mfrow = c(1, 1))
```

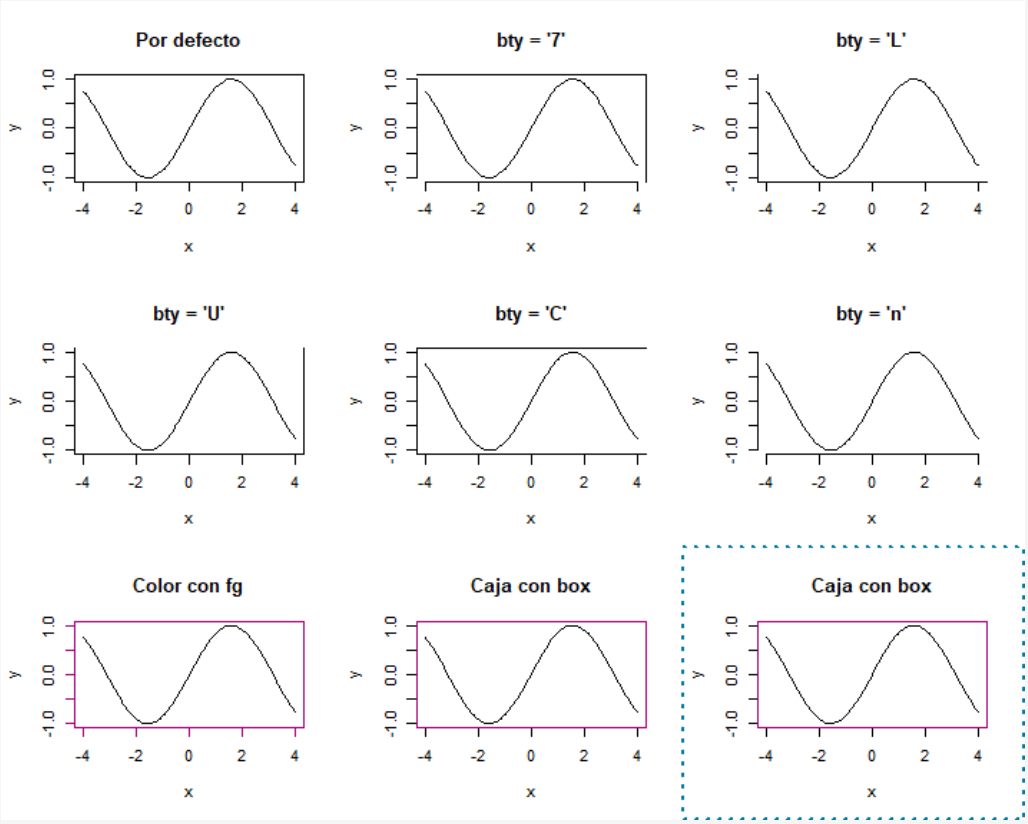


Figura 22: Distintas posibilidades para la caja alrededor del gráfico.

Añadir líneas o puntos a un gráfico

La función `plot()` permite dibujar una sola cosa (una familia de puntos o una función) con un único estilo. Si queremos dibujar varios objetos en un gráfico, los tenemos que añadir uno a uno, usando las funciones adecuadas. En esta sección veremos algunas funciones que permiten añadir elementos a un gráfico.

Es preciso decir que, cuando añadimos objetos a un gráfico, no podemos modificar su diseño general. Por ejemplo, los rangos de coordenadas de los ejes del gráfico final o sus etiquetas serán los del primer gráfico, aunque especifiquemos valores nuevos en el argumento de las funciones que usemos para añadir objetos.

Podemos agregar líneas a un gráfico ya existente utilizando la función `lines()` y puntos, con la función `points()`. También podemos utilizar la función `abline()` para añadir una recta. Otra opción es utilizar la función `curve()` para añadir la curva de una función al gráfico activo. Veamos cada una de ellas.

La instrucción

```
lines(x, y)
```

donde `x e y` son dos vectores numéricos de la misma longitud, añade gráfico una línea poligonal que une los puntos (x_i, y_i) sucesivos. El efecto es el mismo que si añadiéramos

```
plot(x, y, type = "l")
```

Podemos personalizar la apariencia de las líneas modificándola con los parámetros usuales de grosor, color, estilo, etc.

Ejemplo: función `lines()` para agregar líneas a gráfico existente

```
# Primero hacemos el gráfico de puntos (de dispersión)
set.seed(23)
x <- rnorm(50)
y <- 2*x +3 + rnorm(50)
par(pty="s")
```

```
plot(x, y,
  pch = 21,
  col = "#0098CD",
  bg = "#B3B3B3",
  cex = 1.2,
  bty = "n",
  xlim = c(-2,8),
  ylim = c(-2,8),
  xaxp = c(-4, 8, 6),
  yaxp = c(-4, 8, 6),
  main = 'Grafico de dispersión',
  cex.axis = 0.8,
  cex.main = 0.9,
  col.lab = "#00566C",
  las = 1
)

# añadimos la poligonal por los puntos
lines(x, y, col = "grey", lty = 2)

# añadimos la recta y = 2x + 3 utilizando la función lines
x <- seq(-2, 2, by = 0.1)
y = 2*x + 3
lines(x, y,
  col = "#EE9A00",
  lty = 2,
  lwd = 2
)
```

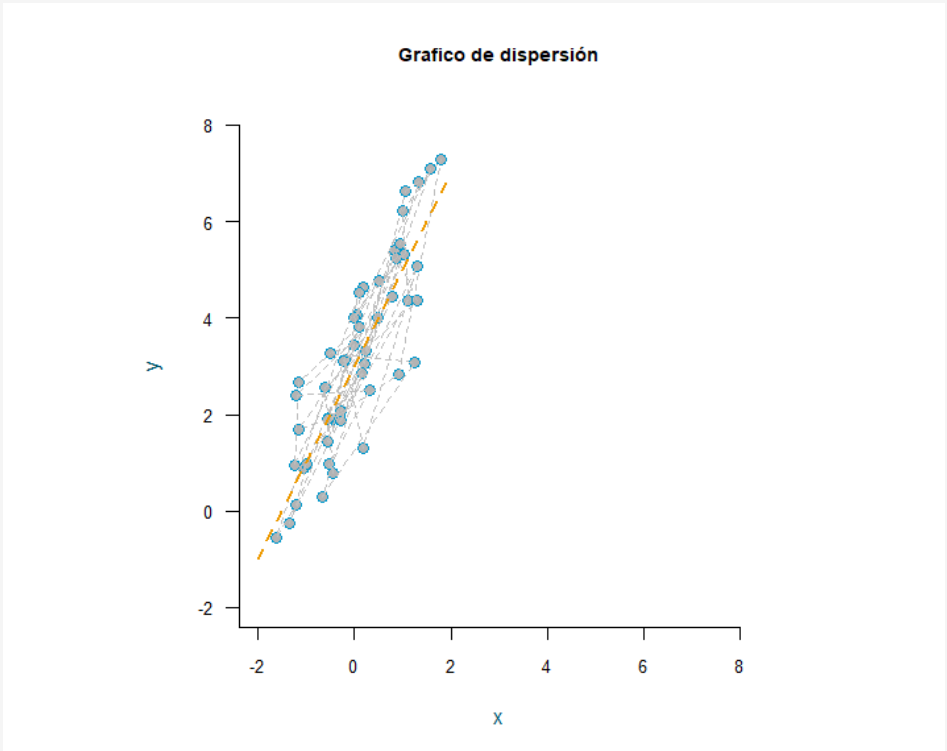


Figura 23: Agregar líneas a un gráfico.

Debemos insistir en que la función `lines()` agrega una línea poligonal uniendo los puntos en los vectores x e y, por lo tanto, si estos no están ordenados según el sentido creciente sobre el eje de las abscisas, el resultado es como una "tela de araña", como podemos observar en el ejemplo anterior.

Por otro lado, en el ejemplo, utilizamos la función `lines()` para trazar la recta $y = 2x + 3$ pero, sin duda, esto puede hacerse de forma más sencilla utilizando la función `abline()`.

La función `abline()` sirve para añadir una recta al gráfico. Esta función tiene tres variantes:

- `abline(a = a0, b = b0)` añade la recta $y = a_0 + b_0 x$.
- `abline(v = v0)` añade la recta vertical $x = v_0$.
- `abline(h = h0)` añade la recta horizontal $y = h_0$.

Podemos especificar las características de estas rectas, como su grosor, su estilo o su color, mediante los parámetros pertinentes.

Ejemplo: función abline() para agregar rectas

```
# Primero hacemos el gráfico de dispersión
set.seed(23)
x <- rnorm(50)
y <- 2*x +3 + rnorm(50)
par(pty="s")
plot(x, y,
     pch = 21,
     col = "#0098CD",
     bg = "#B3B3B3",
     cex = 1.2,
     bty = "n",
     xlim = c(-2,8),
     ylim = c(-2,8),
     xaxp = c(-4, 8, 6),
     yaxp = c(-4, 8, 6),
     main = 'Grafico de dispersión',
     cex.axis = 0.8,
     cex.main = 0.9,
     col.lab = "#00566C",
     las = 1
)
```

```r
# añadimos la poligonal por los puntos
abline(
  a = 3,
  b = 2,
  col = "#EE9A00",
  lwd = 2,
  lty = 3
)

# Agregamos una vertical
abline(v = 3, col = "blue", lty = 2, lwd = 2)

# y una horizontal
abline(h = 3, col = "violetred2", lty = 2, lwd = 2)

# También podemos agregar una grilla
abline(h = -2:8, v = -2:8, col = "lightgray", lty = 3)
```

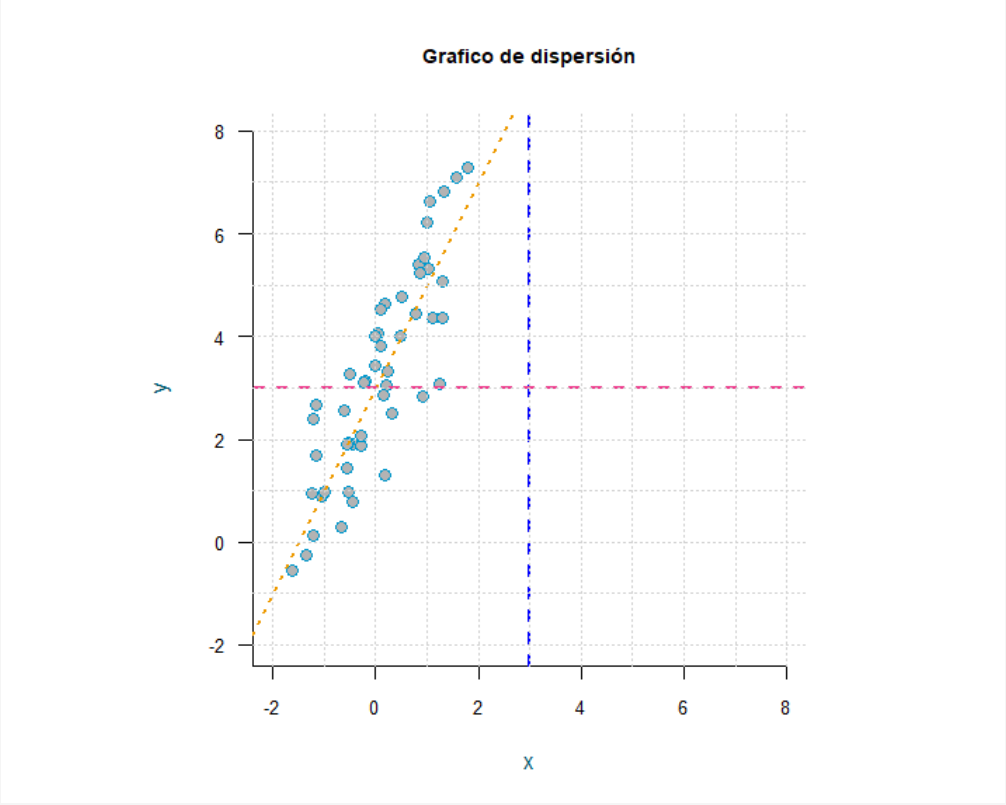


Figura 24: Agregar rectas a un gráfico con `abline()`.

La función `curve()` con el parámetro `add = TRUE` permite añadir la gráfica de una curva a un gráfico anterior. La curva se puede especificar mediante una expresión algebraica con variable `x`, o mediante su nombre si la hemos definido antes o es una función integrada en R.

Ejemplo: Añadir una línea con la función `curve()`

```
# Primero creamos el gráfico
x <- sample(x = seq(-3, 3, 0.1), size = 10)
y <- dnorm(x)
plot(x, y,
  pch = 19,
  col = '#0098CD',
  ylim = c(0, 0.4),
  xlim = c(-3,3)
)
# Ahora añadimos la densidad de la normal estándar
curve(dnorm(x), col = 'darkgreen', lty = 2, add = TRUE)
```

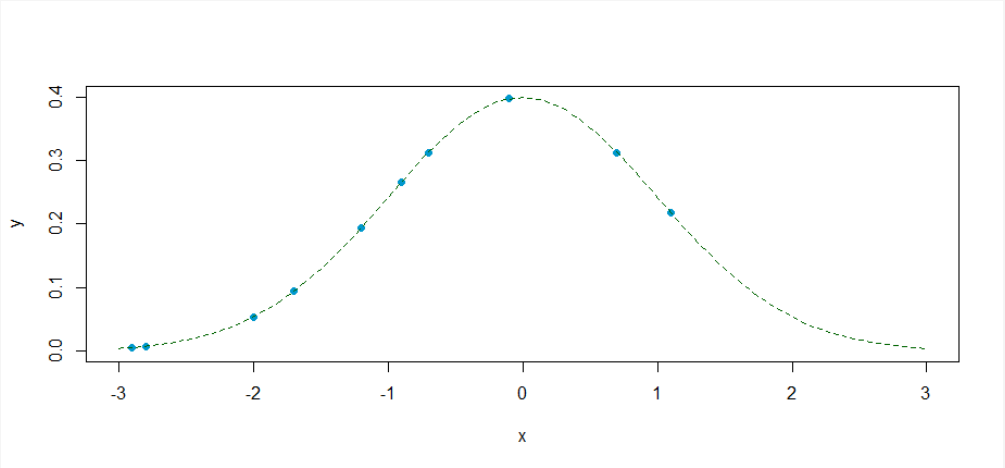


Figura 25: Agregar línea a un gráfico con `curve()`.

Veamos ahora como agregar puntos a un gráfico que ya existe. La instrucción

```
points(x, y)
```

añade un punto de coordenadas (x,y) al gráfico activo si x e y son números. Podemos declarar el color de este punto, el signo que lo represente, etc. mediante los parámetros usuales. Por ejemplo,

Ejemplo: Añadir puntos a un gráfico

```
# Primero creamos el gráfico
x <- seq(-4, 4, length.out = 101)
y <- sin(x)
plot(x, y,
  type = "l",
  main = "Gráfico de la función seno",
  ylab = expression(italic(sen) ~ "( " * phi ~ ")"),
  xlab = expression(paste("Angulo de fase ", phi)),
  col.main = "#0098CD",
  bty = "n",
  xaxt = "n",
```

```
)
axis(1,
  at = c(-pi, -pi / 2, 0, pi / 2, pi),
  labels = expression(-pi, -pi / 2, 0, pi / 2, pi)
)
# Añadimos los puntos (0, 0) y (pi/2, 1)
points(
  x = c(0, pi/2),
  y = c(0, 1),
  pch = 19,
  col = "#DB8100",
  cex = 1.5
)
```

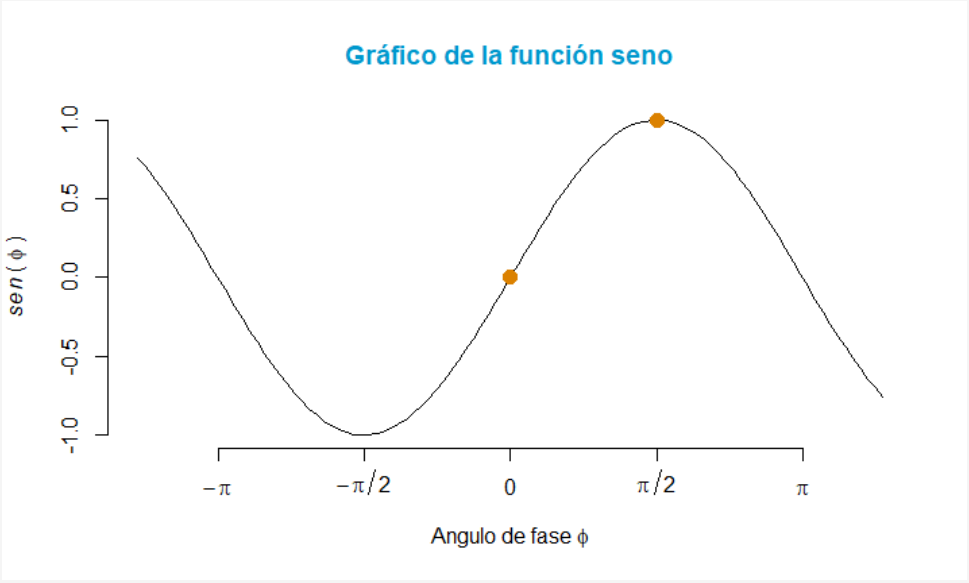


Figura 26: Agregar puntos a un gráfico con `points()`.

Añadir texto a un gráfico

Otra opción que puede resultar interesante es la de agregar texto a un gráfico. Para ello podemos utilizar las funciones `mtext()` y `text()`. La primera permite agregar texto a todos los lados del gráfico. Hay 12 combinaciones posibles (3 en cada lado de la caja: alineadas a la izquierda, al centro y a la derecha). Solo necesitas cambiar los argumentos `adj` y `side` para obtener la combinación que necesites. Por su parte, la función `text()`permite agregar texto o fórmulas dentro del gráfico en alguna posición que se le indique mediante sus coordenadas.

Ejemplo: Agregar texto al gráfico activo

```
# Primero creamos el gráfico
plot(x, y,
```

```
  type = 'n',
  xaxt = "n",
  yaxt = "n",
  main = ""
)

#---------------
# Función mtext
#---------------

# Abajo centro
mtext("Abajo", side = 1)

# Izquierda centro
mtext("Izquierda", side = 2)

# Arriba centro
mtext("Arriba", side = 3)

# Derecha centro
mtext("Derecha", side = 4)

# Abajo izquierda
mtext("Abajo izquierda", side = 1, adj = 0)

# Abajo derecha
mtext("Abajo derecha", side = 1, adj = 1, cex = 0.7)

# Arriba izquierda
mtext("Arriba izquierda", side = 3, adj = 0, font = 3)

# Arriba derecha
mtext("Arriba derecha", side = 3, adj = 1, font = 2)

# Arriba, con separación especificando argumento line
mtext("Texto arriba", side = 3, line = 2.5)

# Izquierda arriba, con separación especificando argumento line
mtext("Arriba izquierda", side = 2, adj = 1, line = 2, col = 'blue',
cex = 0.9)

# Derecha abajo, con separación especificando argumento line
mtext("Abajo derecha", side = 4, adj = 0, line = 1, font = 4, col =
'red')

# Derecha abajo, con separación especificando argumento line
mtext(expression(frac(pi,2)~'sen('~theta~')'), side = 4, adj = 1, line
= 1, col = 'darkgreen', cex = 0.7)

#--------------
# Función text
#--------------
```

```
# Texto simple en las coordenadas indicadas
text(-2, 0.5, "texto simple")
points(-2, 0.5, col= '#0098CD', pch = 19)

text(-2, 0, "texto simple, adj = 0.5", adj = 0.5)
points(-2, 0, col= '#0098CD', pch = 19)

text(-2, -0.5, "texto simple, adj = 1", adj = 1)
points(-2, -0.5, col= '#0098CD', pch = 19)

text(-2, 1, "texto simple, adj = 0.8", adj = 0.8)
points(-2, 1, col= '#0098CD', pch = 19)

text(2, 1, "texto simple, pos = 1", pos = 1)
points(2, 1, col= '#0098CD', pch = 19)

text(2, 0.5, "texto simple, pos = 2", pos = 2)
points(2, 0.5, col= '#0098CD', pch = 19)

text(2, 0, "texto simple, pos = 3", pos = 3)
points(2, 0, col= '#0098CD', pch = 19)

text(2, -0.5, "texto simple, pos = 4", pos = 4)
points(2, -0.5, col= '#0098CD', pch = 19)


# Fórmula en las coordenadas indicadas
text(0, 0.5, expression(frac(alpha[1], 4)))
points(0, 0.5, col= '#0098CD', pch = 19)

text(0, 0, expression(frac(alpha[1], 4)~', adj = 0'), adj = 0)
points(0, 0, col= '#0098CD', pch = 19)

text(0, -0.5, expression(frac(alpha[1], 4)~', pos = 4'), pos = 4)
points(0, -0.5, col= '#0098CD', pch = 19)
```

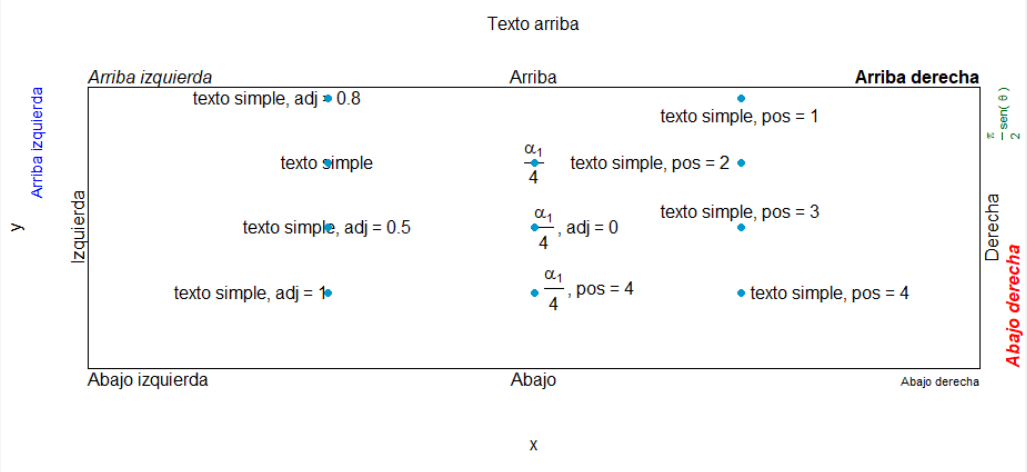


Figura 27: Agregar texto a un gráfico con `mtext()` y `text()`.

Añadir una leyenda al gráfico

Cuando dibujamos varias funciones en un mismo gráfico es conveniente usar una leyenda para identificarlas. Para añadirla, podemos utilizar la función `legend()`.

La sintaxis básica para esta función es

```
legend(x, y = NULL, legend, fill, col, border = "black", lty, lwd, pch)
```

donde:

- `x, y`: indica dónde queremos situar la leyenda, y puede ser o bien dos números para especificar las coordenadas de su esquina superior izquierda, o bien una de las palabras siguientes: `"bottomright"` (esquina inferior derecha), `"bottom"` (centrada abajo), `"bottomleft"` (esquina inferior izquierda), `"left"` (centrada a la izquierda), `"topleft"` (esquina superior izquierda), `"top"` (centrada arriba), `"topright"` (esquina superior derecha), `"right"` (centrada a la derecha) o `"center"` (en el centro del gráfico).
- `legend` es un vector que contiene los nombres (entre comillas o aplicándoles `expression`) con los que queremos identificar las curvas dentro de la leyenda.
- Con los parámetros `fill, col, border,` etc., personalizamos los elementos de la leyenda.
- Se puede usar también el parámetro `cex` dentro de la función `legend` para especificar el factor que queremos que se aplique al tamaño de la leyenda, si queremos modificar este último.

Puedes consultar este artículo si te interesa profundizar sobre como añadir y personalizar leyendas en R.

Ejemplo: Añadir leyenda a un gráfico.

```
color = c('#008FBF','#7EB433','#DB8100','#5A1560')
mus <- c(0,0,0,2)
sigmas <- c(0.5,1, 1.5, 1.5)

curve(dnorm(x, mus[1], sigmas[1]), main = expression(paste("Densidades
para distintos valores de ", mu, " y ", sigma)), lwd=2,bty='L',
      xlim = c(-3,7),ylim=c(0,.8), xlab = '', ylab = 'f(x)', col =
color[1],col.main="#1A0644",cex.axis=0.8,
```

```
      col.axis="#1A0644",cex=.75,las=1)
lines(c(mus[1],mus[1]),c(0,dnorm(mus[1],mus[1],sigmas[1])),lty=2,col=
'red')
for(j in 1:3){
  curve(dnorm(x, mus[j+1], sigmas[j+1]), main = '',xlab = '', ylab =
'', col = color[j+1],lwd=2,add=TRUE)

lines(c(mus[j+1],mus[j+1]),c(0,dnorm(mus[j+1],mus[j+1],sigmas[j+1])),
lty=2,col='red')
}
leg <- sprintf("μ = %1d, σ = %.1f \n",mus, sigmas)
legend(
  4,0.85,
  legend = leg,
  lwd=2,
  lty=1,
  col=color,
  bty='n',
  cex=.8
)
```

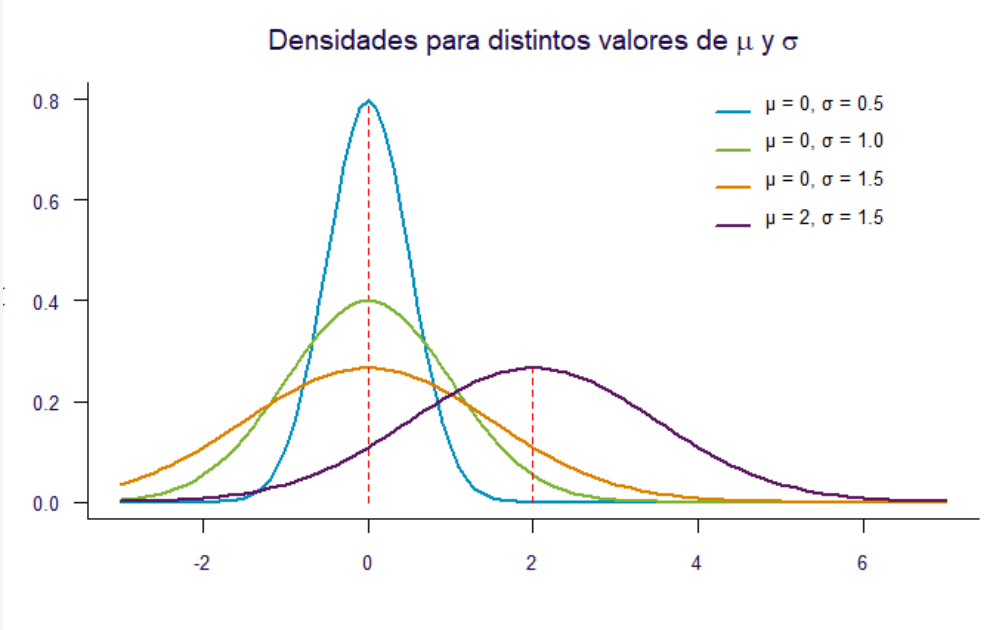


Figura 28: Añadir leyenda a un gráfico.

6.3. La función `hist()`

El histograma es quizá el gráfico que se utiliza con mayor frecuencia para representar una variable continua.

Para hacer un histograma usamos la función `hist()`, que produce un histograma del vector numérico que recibe como argumento. El número de clases (intervalos) es seleccionado automáticamente, pero puede ser especificado, así como los límites de las clases.

Ejemplo: Mi primer histograma.
El conjunto de datos `NutritionStudy.csv` (provisto como parte del material de curso), contiene datos de 315 pacientes sometidos a cirugía electiva de un estudio transversal para investigar la relación entre las características personales y los factores dietéticos, y las concentraciones plasmáticas de retinol, betacaroteno y otros carotenoides. Los sujetos del estudio fueron pacientes que se sometieron a un procedimiento quirúrgico electivo durante un período de tres años para realizar una biopsia o extirpar una lesión de pulmón, colon, mama, piel, ovario o útero que no resultó cancerosa.
Una de las variables en el conjunto de datos es `Fiber`, que corresponde a la cantidad, en gramos, de fibras consumidas por el paciente por día. Esta es una variable cuantitativa continua. Veamos cómo es la distribución de estos datos construyendo el histograma de la variable.

```
NutritionStudy <- read_csv("datos/NutritionStudy.csv")
attach(NutritionStudy)
hist(Fiber)
```

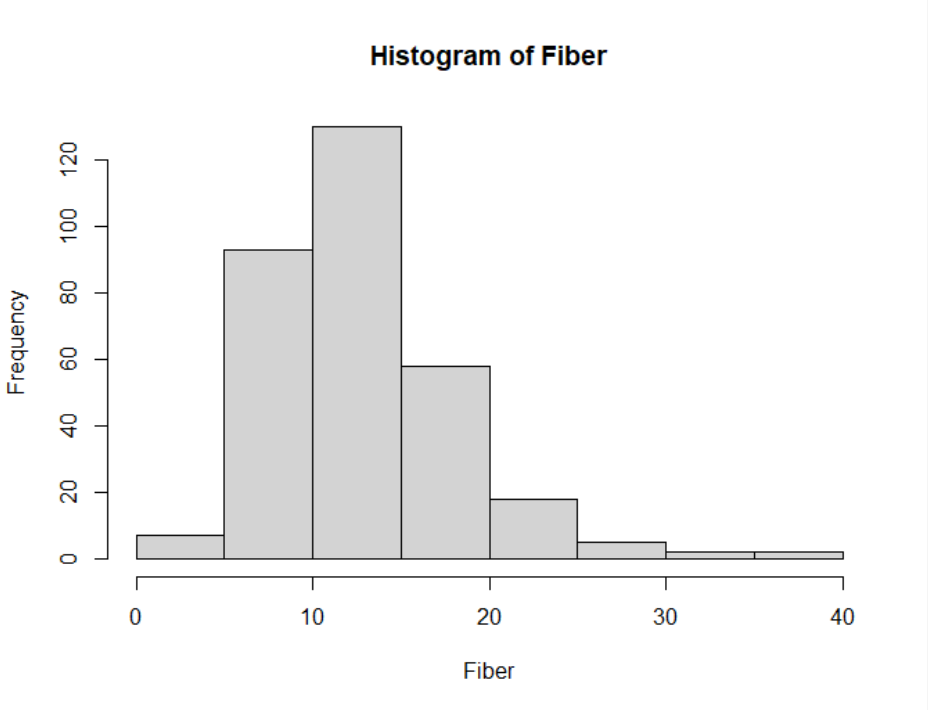


Figura 29: Histograma: valores por defecto.

Al utilizar `hist(Fiber)` obtenemos el histograma de la variable `Fiber`, tomando los valores de los parámetros por defecto. Como podemos observar, se muestran las frecuencias absolutas de cada clase. Esto es, las alturas de las barras están dadas por las frecuencias absolutas de cada intervalo. Podemos modificar este comportamiento especificando el argumento `freq = FALSE` (o `prob = TRUE`), en cuyo el histograma de construye en escala de densidades, significando que el área de cada barra es igual a la proporción o frecuencia relativa del intervalo correspondiente. De este modo, el área total del histograma es igual a 1.

Muchos de los parámetros que ya vimos para la función `plot()`, pueden utilizarse aquí también. Puedes consultar todos los argumentos que tiene la función accediendo a la ayuda ejecutando `?hist`.

Ejemplo: Modificando el aspecto del histograma.

```
hist(Fiber,
     freq = FALSE,
     border = 'grey',
     col = '#E5F8FF',
     xlab = 'Fibras (gr/sem)',
     ylab = 'Densidad',
     labels = TRUE,
     ylim = c(0,0.1),
     main = 'Hstograma de la variable Fiber',
     cex.main = 0.9,
     font.main = 4,
     cex.lab = 0.9,
     cex.axis = 0.8,
     font.lab = 2,
     font.axis = 3,
     col.lab = 'gray27',
     col.axis = 'midnightblue'
)
lines(density(Fiber), col = 'red')
```

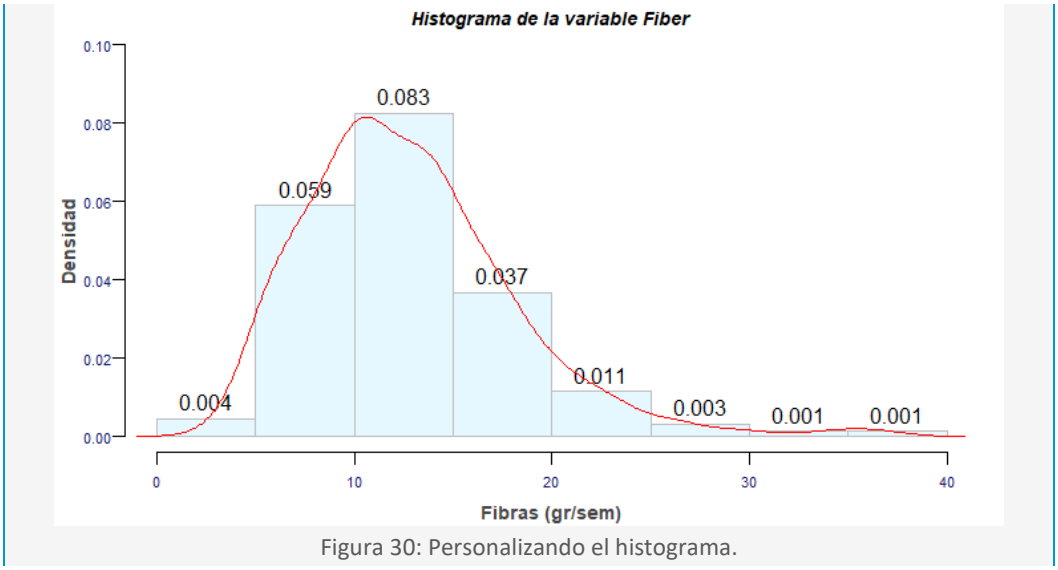


Figura 30: Personalizando el histograma.

Como podemos observar en el ejemplo, también podemos añadir elementos al histograma (en el ejemplo, una línea con la función `lines()`), tal como ya lo vimos con los gráficos obtenidos con la función `plot()`.

El argumento `labels = TRUE`, añade una etiqueta con la altura de cada barra.

Argumento `breaks`

Los histogramas son una herramienta valiosa para resumir gráficamente la distribución de un conjunto de datos siempre que el **número de barras o clases se seleccione correctamente.** Sin embargo, la selección del número de barras (o el ancho de las barras) puede ser complicada:

- Pocas clases agruparán demasiado las observaciones.
- Con demasiadas clases habrá pocas observaciones en cada una de ellas aumentando la variabilidad del gráfico obtenido.

Hay varias reglas para determinar el número de barras. En R, *el método de Sturges* (`breaks = "Sturges"`) se usa por defecto. Si queremos cambiar el número de barras, debemos especificar el valor del argumento `breaks` con una de las siguientes opciones:

- un vector que proporciona los puntos de corte entre los intervalos del histograma,

- una función para calcular el vector de puntos de corte,
- un solo número que da el número de intervalos para el histograma,
- una cadena de caracteres que nombra un algoritmo para calcular el número de intervalos ("Sturges", "Scott" o "FD"/"Freedman-Diaconis"),
- una función para calcular el número de intervalos.

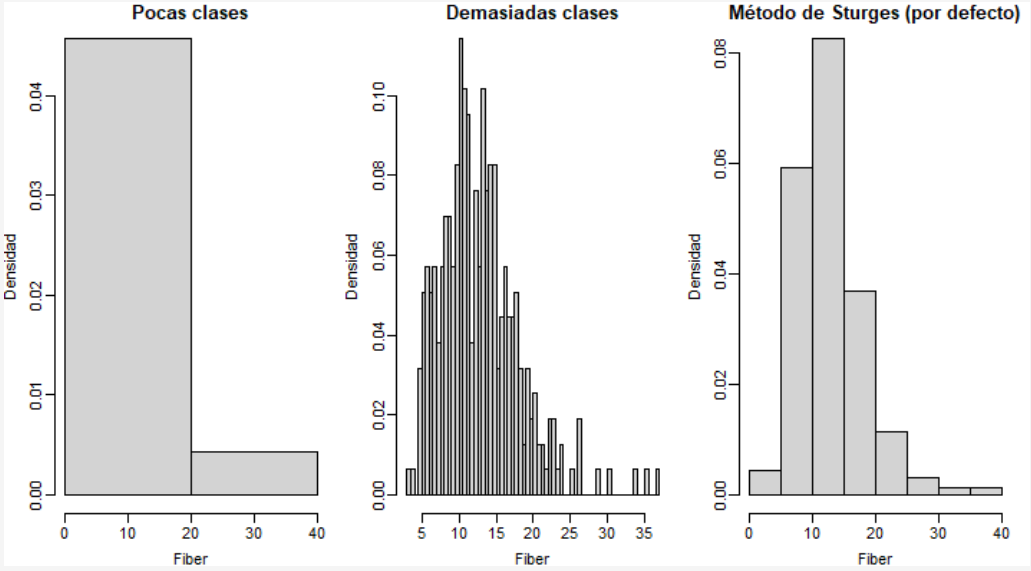
Ejemplo: Modificar los intervalos con `breaks`.

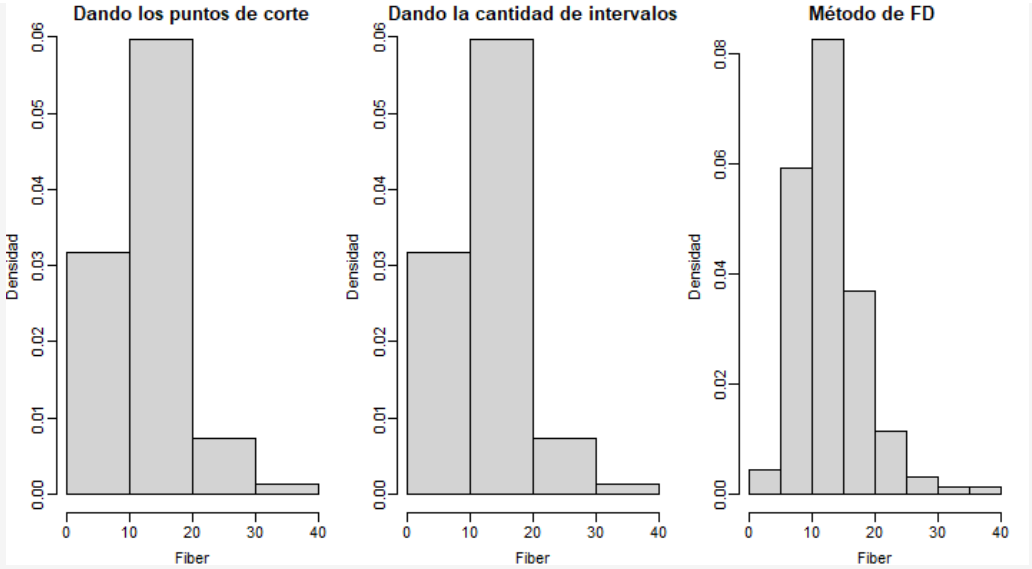
```
par(mfrow = c(1, 3), mar = c(2.9,2.8,1,1), mgp = c(2,0.55,0), las = 0)

hist(Fiber, breaks = 2, freq = FALSE, main = "Pocas clases", ylab =
"Densidad")
hist(Fiber, breaks = 50, freq = FALSE, main = "Demasiadas clases", ylab
= "Densidad")
hist(Fiber, freq = FALSE, main = "Método de Sturges (por defecto)",
ylab = "Densidad")

hist(Fiber, breaks = c(0,10,20,30,40), freq = FALSE, main = "Dando los
puntos de corte", ylab = "Densidad")
hist(Fiber, breaks = 4, freq = FALSE, main = "Dando la cantidad de
intervalos", ylab = "Densidad")
hist(Fiber, freq = FALSE, main = "Método de FD ", ylab = "Densidad")

par(mfrow = c(1, 1))
```





Figuras 31 y 32: Modificar la cantidad de intervalos/barras usando el argumento `breaks`.

6.4. La función `boxplot()`

Un diagrama de caja y bigotes, o *box plot*, es un gráfico que permite resumir las principales características de datos de una variable cuantitativa (simetría, posición, dispersión) así como también, identificar si hay datos atípicos. Este gráfico marca básicamente cinco valores:

- Los lados inferior y superior de la caja representan el **primer y el tercer cuartil**, por lo que la altura de la caja es igual al rango intercuartílico. Así, la caja demarca al 50% central de los casos.
- La línea central de este gráfico representa a la **mediana**, es decir al valor que señala el 50% de la distribución de la variable en estudio.
- Los valores b_{inf} y b_{sup}, son los extremos de los bigotes. Estos valores se calculan de la siguiente manera:
  - Sea m el mínimo del conjunto de datos. Si $m \geq Q_1 - 1.5 \cdot (Q_3 - Q_1)$, entonces $b_{inf} = m$. Si, en cambio, $m < Q_1 - 1.5 \cdot (Q_3 - Q_1)$, entonces b_{inf} es el menor elemento del conjunto de datos que es mayor o igual que $Q_1 - 1.5 \cdot (Q_3 - Q_1)$.
  - Sea M el máximo del conjunto de datos. Si $M \leq Q_3 - 1.5 \cdot (Q_3 - Q_1)$, entonces $b_{sup} = M$. Si, en cambio, $M > Q_3 - 1.5 \cdot (Q_3 - Q_1)$, entonces b_{sup} es el mayor elemento del conjunto de datos que es menor o igual que $Q_3 - 1.5 \cdot (Q_3 - Q_1)$.

Es decir, los bigotes marcan el mínimo y el máximo de la variable, excepto cuando están muy alejados de la caja intercuartílica; en este caso, el bigote inferior marca el menor valor por debajo de la caja intercuartílica a distancia menor o igual que 1.5 veces la altura de dicha caja, y el superior marca el mayor valor por encima de la caja intercuartílica a distancia menor o igual que 1.5 veces la altura de dicha caja.

- Si hay datos más allá de los bigotes (menores que b_{inf} o mayores que b_{sup}) se marcan como puntos aislados: son los valores atípicos (outliers) de la variable.

Ejemplo: Diagrama de cajas para la variable Fiber

```
boxplot(Fiber,
        col = "#E5F8FF",
        horizontal = FALSE,
        border="#1D6786",
        boxwex = 0.3,staplewex = 0.3, width=4,
        col.axis="#0A335D",
        cex.axis=0.8)
title("Fibras consumidas (g/sem)",adj = 0.4,line=1)
```

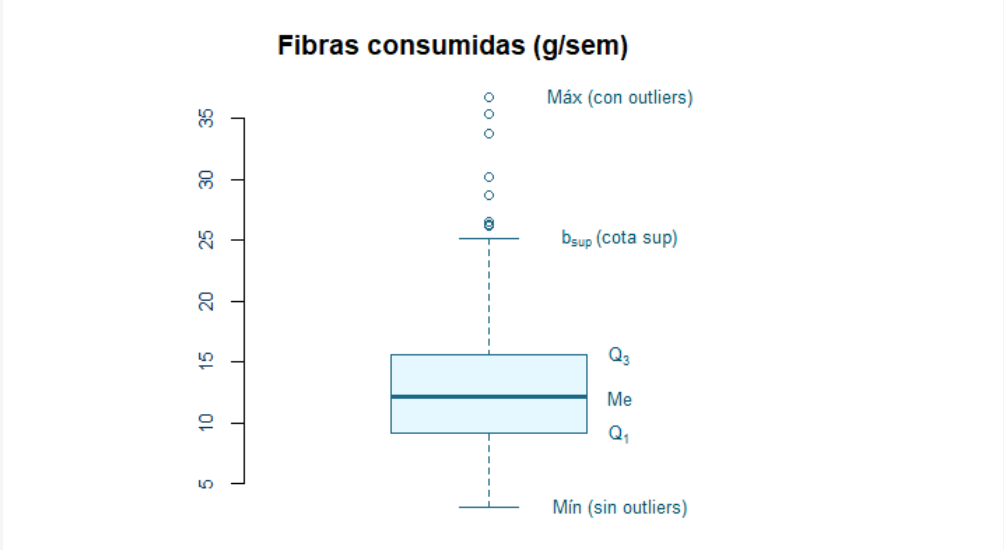


Figura 33: Diagrama de caja y bigotes para los datos de Fiber.

Para dibujar varios diagramas de caja en un mismo gráfico, por ejemplo, para poder compararlos, basta aplicar la instrucción `boxplot()` a todos los vectores simultáneamente. Por ejemplo:

Ejemplo: Dibujar más de un boxplot en el mismo gráfico.

```
boxplot(
  Fiber,
```

```
  Fat,
  col = "#E5F8FF",
  border="#1D6786",
  names = c("Fibras", "Grasas")
)
```

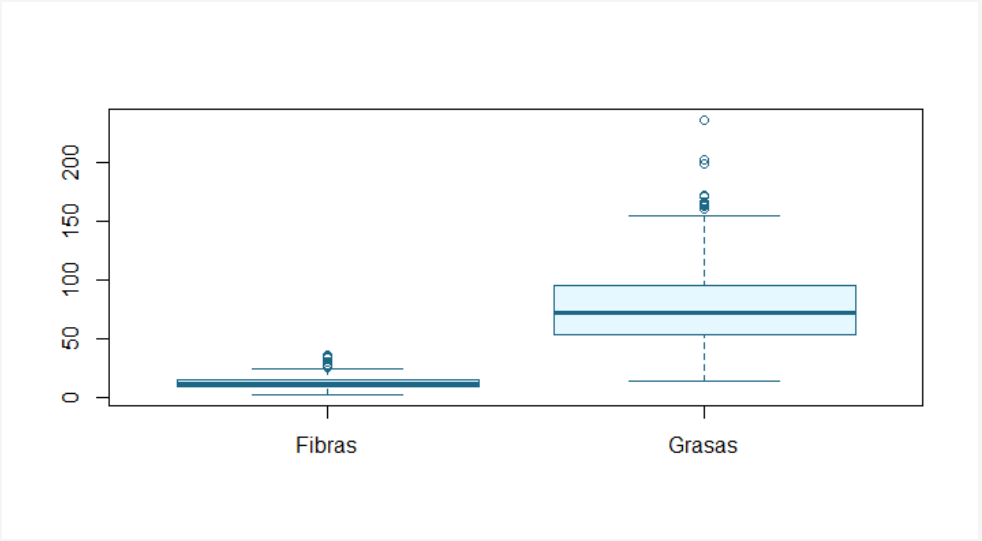


Figura 34: Múltiples diagramas de caja en un mismo gráfico.

El objetivo de agrupar varios diagramas de caja en un único gráfico suele ser el poder compararlos visualmente, y esto tiene sentido cuando las variables tienen significados muy similares o cuando son la misma variable sobre poblaciones diferentes. Esto es lo que ocurre cuando queremos producir diagramas de caja de una variable cuantitativa agrupada por un factor, porque esto nos permitirá comparar el comportamiento de esta variable sobre cada uno de los niveles del factor. En tales situaciones solemos decir que la variable cuantitativa es la respuesta de interés y el factor la variable predictora.

La instrucción básica para dibujar en un único gráfico los diagramas de caja de una variable numérica segmentada por un factor es

```
boxplot(respuesta ~ predictora)
```

Veamos como ejemplo, como graficar los diagramas de caja de la variable Fiber según el sexo de los individuos.

Ejemplo: Boxplot de una variable cuantitativa en fusión de una cualitativa (factor)

```
boxplot(
```

```
  Fiber ~ Sex,
  col = rainbow(2, alpha = 0.5),
  names = c("Mujeres", "Varones"),
  xlab = 'Sexo',
  ylab = "Fibras consumidas (g/sem)"
)
```

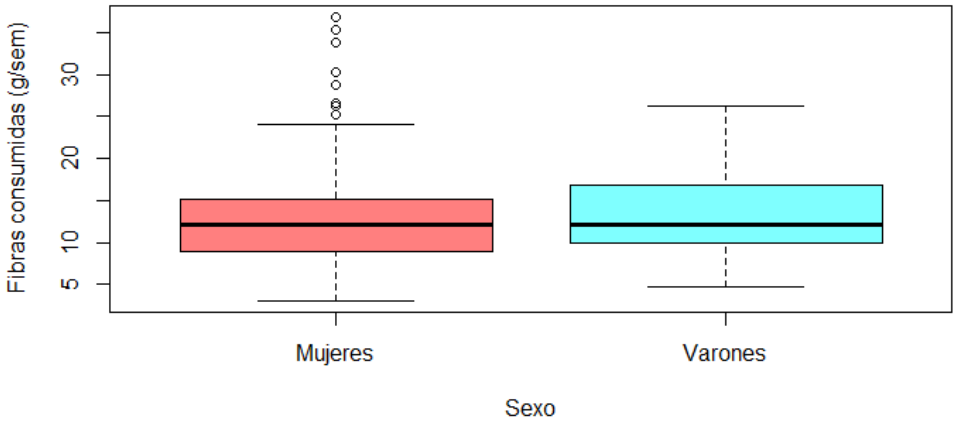


Figura 35: Diagramas de caja de una variable cuantitativa en función de otra cualitativa.

6.5. Gráfico de barras: la función `barplot()`

El tipo de gráfico más usado para representar variables cualitativas son los gráficos o diagramas de barras (*bar plots*). Un diagrama de barras contiene, para cada nivel de la variable cualitativa, una barra de altura igual su frecuencia (absoluta o relativa). Para crear un gráfico de barras en R, podemos utilizar la función `barplot()` de R base.

La sintaxis básica de esta función es:

```
barplot(
  height,
  names.arg = NULL,
  legend.text = NULL,
  beside = FALSE,
  horiz = FALSE
)
```

donde

- `height`: son las alturas de las barras que se dibujarán, usualmente, una tabla de frecuencias absolutas o relativas.
- `names.arg`: un vector de etiquetas de cada barra o grupo de barras. Si se omite este argumento, los nombres se toman del atributo *names* de `height` (si se trata de un vector), o de los nombres de las columnas si se trata de una matriz.
- `legend.text`: un vector donde puedes especificar los nombres que quieres agregar a la leyenda y un valor lógico (`TRUE` o `FALSE`) para indicar si se debe añadir o no una leyenda. En caso de `legend.text = TRUE`, los nombres de las filas de `height` se utilizarán como etiquetas para la leyenda.
- `beside`: valor lógico (`FALSE` por defecto) que especifica si las columnas de `height` se deben graficar apiladas o adosadas. Esto aplica cuando estamos graficando dos variables cualitativas simultáneamente.
- `horiz`: valor lógico (`FALSE` por defecto) para especificar si el diagrama se debe dibujar horizontal (`TRUE`) o no (`FALSE`).

Nuevamente, muchos de los parámetros ya estudiados pueden aplicarse a la función `barplot()` para personalizar el gráfico final. No olvides consultar la ayuda de la función (`?barplot`).

Para ejemplificar el uso de esta función, vamos a trabajar con los datos `Cars2020.csv`, proporcionados como material del curso. El conjunto de datos contine información sobre nuevos modelos de automóviles en 2020. El dataset cuenta con 110 observaciones de 24 variables, entre las cuales nos interesa estudiar:

- `Drive`: tipo de tracción del vehículo. Cualitativa con tres niveles (`"FWD"`: delantera, `"RWD"`: trasera, `"AWD"`: tracción total o en las 4 ruedas)
- `Size`: Tamaño del vehículo. Cualitativa con tres niveles (`Small, Midsized` o `Large`).

> Ejemplo: Gráfico o diagrama de barras.
> Vamos a crear primero el diagrama de barras de la variable `Drive`, para ver distribución entre sus categorías.
> Para ello, comenzamos por obtener la tabla de frecuencias (absolutas o relativas) de la variable.

```
tabla_abs <- table(Drive)
tabla_abs
## Drive
## AWD FWD RWD
##  80  25   5

tabla_prop <- prop.table(table(Drive))
tabla_prop
## Drive
##    AWD    FWD    RWD
## 0.7273 0.2273 0.0455
```

Ahora, vamos a crear el gráfico de barras. Utilizaremos las frecuencias relativas.

```
bp <- barplot(
        tabla_prop,                      # datos
        main = "Gráfico de barras",      # título del gráfico
        xlab = "Tracción",               # Etiqueta del eje X
        names.arg = c("Total", "Delantera", "Trasera"),  # Etiquetas d
las barras
        ylab = "Frecuencia relativa",    # Etiqueta eje Y
        col = c("#C9A340", "#008FBF", "#F50A8BD4"),  # Color para cada
barra
        density = 50, angle = 45,        # Densidad e inclinación de
las líneas de sombreado de cada barras
        border = "white",              # color del borde de las barras
        cex.names = 0.8
      )
text(bp, 0, round(tabla_abs, 1), cex=0.9, pos=3)     # agregamos texto
con las frecuencias absolutas de cada barra
```

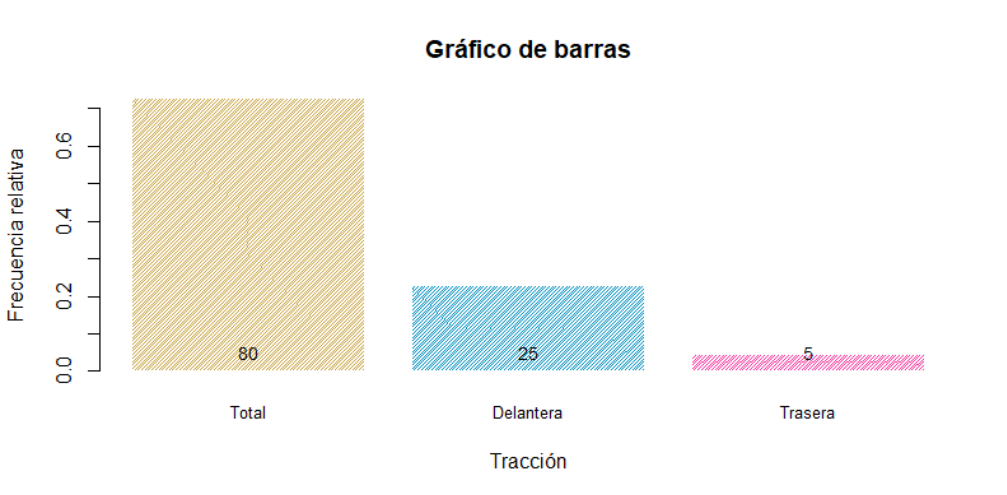


Figura 36: Gráfico de barras para una variable cualitativas.

Si se aplica `barplot()` a una tabla bidimensional, por defecto dibuja las barras de la segunda variable cortadas por la frecuencia de la primera variable: es lo que se llama

un diagrama de barras *apiladas*. Especificando `beside = TRUE`, se dibujarán las barras adosadas en lugar de apiladas.

Ejemplo: Diagrama de barras apiladas con dos variables cualitativas

Supongamos que nos interesa comparar tamaño de los vehículos de acuerdo con su tamaño. En particular nos gustaría saber si es mayor la proporción de autos pequeños entre aquellos que poseen tracción. Para ellos, debemos estudiar las variables `Drive` y `Size`.

En estadística decimos que nos interesa la relación funcional `Size` en función de `Drive`, es decir que, `Size` es la variable respuesta y `Drive` la predictora. Esta identificación es importante pues determina la forma en que se calculan las proporciones. Además, aquí resulta fundamental construir el gráfico utilizando las frecuencias relativas para que sea posibles comparar las barras.

```
# Primero creamos las tablas de frecuencias
tabla2C <- table(Drive, Size)  # absolutas
tabla2C_prop <- prop.table(tabla2C, 1)  # relativas
color <- c("#C9A340", "#008FBF", "#F50A8BD4")
leyenda <- rownames(tabla2C_prop)
leyenda <- ifelse(leyenda == 'AWD', "Total", ifelse(leyenda == "FWD",
"Delantera", "Trasera"))

barplot(
  tabla2C_prop,
  main = "Gráfico de barras apiladas",
  xlab = "Size",
  names.arg = c("Grande", "Mediano", "Pequeño"),
  col = color,
  density = 50, angle = 45,
  border = "white",
  cex.names = 0.8,
  legend.text = leyenda,
  args.legend = list(
                  x = "topleft",
                  bty = 'n',
                  cex = 0.9,
                  border = "white",
                  density = 50,
                  angle = 45
                )
)
```

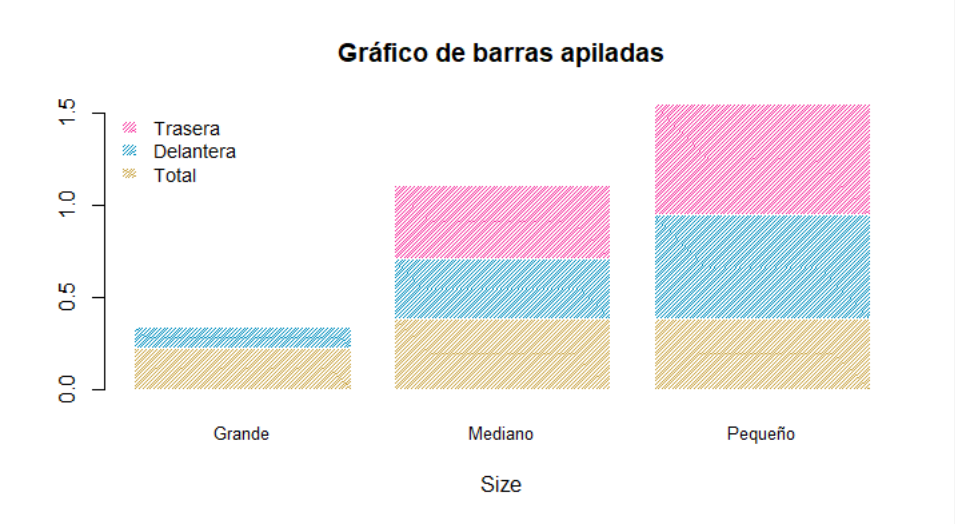


Figura 37: Gráfico de barras apiladas (dos variables cualitativas).

Poniendo el valor del argumento `beside` en `TRUE`, obtenemos el diagrama de barras *adosadas*.

Ejemplo: Diagrama de barras adosadas

```
barplot(
  tabla2C_prop,
  beside = TRUE,
  main = "Gráfico de barras adosadas",
  xlab = "Size",
  names.arg = c("Grande", "Mediano", "Pequeño"),
  col = color,
  density = 50, angle = 45,
  border = "white",
  cex.names = 0.8,
  legend.text = leyenda,
  args.legend    = list(
                     x = "topleft",
                     bty = 'n',
                     cex = 0.9,
                     border = "white",
                     density = 50,
                     angle = 45
                   )
)
```

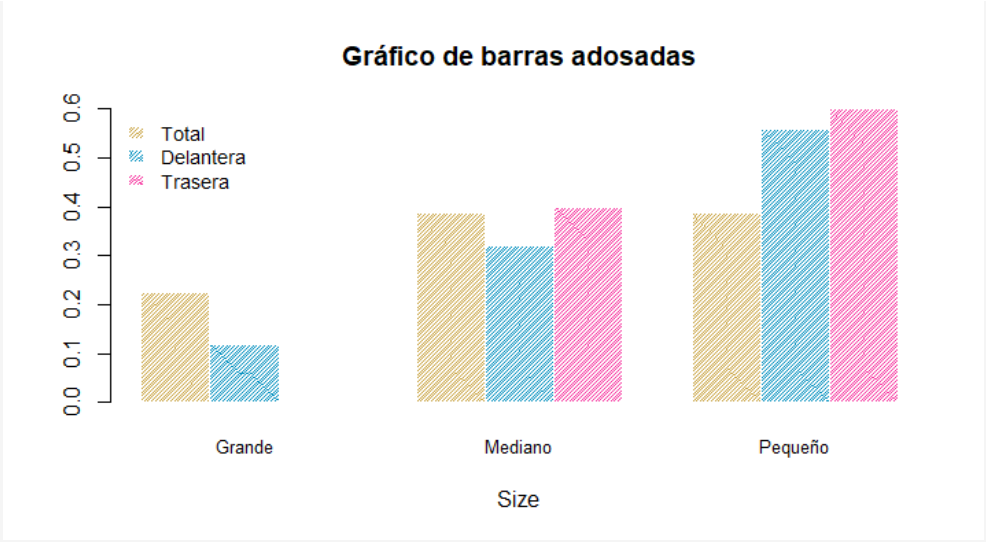


Figura 38: Gráfico de barras adosadas (dos variables cualitativas).

6.6. Otros gráficos

Hay muchos otros gráficos, pero los que acabamos de ver son la base. Para obtener más información e ideas para representar sus datos, puedes consultar el magnífico sitio https://www.data-to-viz.com/ o la galería gráfica R https://www.r-graph-gallery.com/, que incluye ejemplos interesantes de visualización de datos junto con sus respectivas sintaxis de configuración. Es recomendable explorar este tipo de recursos cuando se desee implementar una visualización cuyo código no se maneje.

6.7. El paquete ggplot2

El paquete `{ggplot2}` es una alternativa a las funciones básicas de R para realizar gráficos. `{ggplot2}` divide los gráficos en componentes o capas de manera que le permite a los principiantes crear gráficos relativamente complejos y estéticamente agradables utilizando una sintaxis intuitiva y relativamente fácil de recordar. Así que `{ggplot2}`es un paquete interesante porque ofrece una alternativa con una filosofía diferente en la construcción de gráficos, pero no reemplaza lo que hemos aprendido

hasta ahora. En la práctica podemos utilizar uno u otro en función de los datos y de las manipulaciones que queremos hacer.

Una razón por la cual `{ggplot2}` es generalmente más intuitiva es porque usa una *gramática de gráficos* (Wilkinson, 2005) y de ahí el gg de ggplot2. Esto resulta análogo a cómo construimos y formamos oraciones en español al combinar diferentes elementos, como sustantivos, verbos, artículos, sujetos, objetos, etc. No podemos simplemente combinar estos elementos en un orden arbitrario; debemos hacerlo siguiendo un conjunto de reglas conocido como gramática lingüística. Del mismo modo, al aprender una pequeña cantidad de los componentes básicos de `{ggplot2}` y de su gramática, podremos crear cientos de gráficos diferentes.

Quizás una limitación de `{ggplot2}` es que está diseñado para trabajar exclusivamente con tablas de datos en formato *tidy* (recordemos que esto significa que las filas son observaciones y las columnas son variables). Sin embargo, la mayor parte de *datasets* con los que los principiantes trabajan están en este formato o pueden convertirse a tal, como ya lo hemos visto. Una ventaja de este enfoque es que, con tal que nuestros datos estén *tidy*, ggplot2 simplifica el código de graficar y el aprendizaje de gramática para una variedad de gráficos.

Para usar ggplot2, tendrán que aprender varias funciones y argumentos. Estos son difíciles de memorizar, por lo que les recomendamos que tengan a mano la hoja de referencia o guía rápida (*cheat sheet*) de `{ggplot2}`. Pueden obtener una copia en línea aquí o simplemente realizar una búsqueda en internet de "ggplot2 cheat sheet." Otra posibilidad es acudir a la Galería de gráficos de R, que ya hemos mencionado antes.

6.7.1. Primeros pasos

El primer paso para aprender a utilizar `{ggplot2}` es poder separar un gráfico en componentes. Empezaremos analizando el gráfico anterior e introduciendo algo de la terminología de `{ggplot2}`. Los tres componentes principales para considerar son:

- **Datos**: Se está resumiendo el conjunto de datos `Cars2020`. Nos referimos a esto como el componente `data` (es la fuente de datos).
- **Geometría**: son los elementos visuales que se posicionarán en la gráfica para representar los datos que interesa visualizar. El gráfico anterior es un diagrama de dispersión. Esto se denomina el componente de geometría. Otras posibles geometrías son diagrama de barras, histograma, densidades suaves (smooth densities en inglés), gráfico Q-Q y diagrama de cajas.
- **Mapeo estético**: Mapeo de las variables del conjunto de datos y las propiedades visuales a valores estéticos de la gráfica. Aquí hacemos la definición de los ejes donde se posicionarán los datos a visualizar y especificamos cómo queremos que los datos se vean en el gráfico. Las estéticas incluyen cosas como el tamaño, la forma o el color de tus puntos. La forma en que definimos el mapeo depende de qué geometría estamos usando.

En resumen, la gramática nos dice que un gráfico asigna los **datos** a los atributos **estéticos** (color, forma, tamaño) de objetos **geométricos** (puntos, líneas, barras).
Además de estas tres componentes principales, opcionalmente, el gráfico también puede incluir *transformaciones estadísticas* de los datos e información sobre el *sistema de coordenadas* del gráfico. La creación de *facetas* se puede utilizar para trazar diferentes subconjuntos de datos.

Para construir un gráfico con `{ggplot2}` comenzamos con la siguiente estructura de código, usando + como nexo entre argumentos:

```
ggplot(datos, aes() ) + geom_tipo()
```

A partir de esta estructura básica puede mejorarse la presentación de los gráficos introduciendo, por ejemplo, características estéticas en los objetos geométricos, rotulando los gráficos, etc.

> Ejemplo: mi primer gráfico con `ggplot()`.
> Como nuestro primer ejemplo vamos a utilizar los datos `Cars2020.csv` que ya utilizamos en la sección anterior. En particular, vamos a construir una

visualización que nos permita responder una pregunta: ¿los automóviles más pesados consumen más combustible que los automóviles livianos?
Para responder a la pregunta de interés, vamos a utilizar las variables:

- `Weight`: peso en vacío del vehículo (es una variable cuantitativa continua)
- `HwyMPG`: eficiencia del uso de combustible del vehículo en autopista, en millas por galón (también es una variable cuantitativa continua).

Como tenemos dos variables cuantitativas y queremos estudiar la eficiencia del automóvil según el peso de este, vamos a construir un diagrama de dispersión poniendo la variable `Weight` en el eje horizontal y `HwyMPG` en el vertical. Para ello ejecutamos el siguiente código:

```
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG))
```

Y ya tenemos nuestro primer gráfico con `ggplot()`:

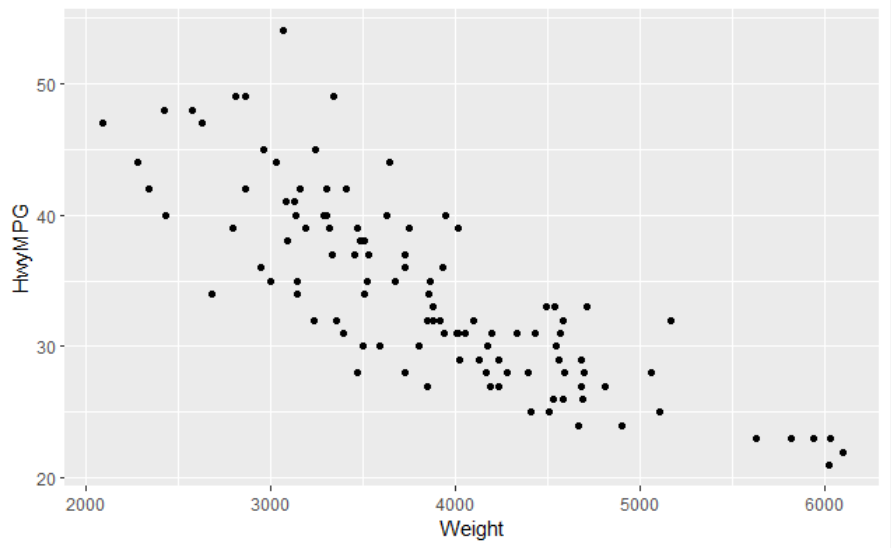


Figura 39: Gráfico de dispersión básico con la función `ggplot()`.

A partir del gráfico vemos que parece existir una relación aproximadamente lineal negativa entre el peso del vehículo y la eficiencia en autopista, es decir, autos más pesados consumen más combustible (es menor la eficiencia a mayor peso).

6.7.2. Construcción de gráficos usando la función `ggplot()`

En este apartado replicaremos los gráficos construidos con las funcionalidades básicas de R, pero ahora aplicando la función `ggplot()`.

Ya hemos creado nuestro primer gráfico usando `ggplot()`, así que continuemos con el mismo.

`geom_point()` o gráficos de dispersión (*scatter plots*)

Ya hemos construido el gráfico base. Ahora, agregando *capas* a nuestro gráfico, podremos ir agregando información que contienen nuestros datos. Por ejemplo, podemos agregar una tercera variable, como `Type`, a un diagrama de dispersión bidimensional asignándolo a un parámetro estético (color, forma o tamaño). Para mapear (o asignar) una estética a una variable, debemos asociar el nombre de la estética (por ejemplo `colour`) al de la variable dentro de `aes()`. Este proceso es conocido como **escalamiento** (*scaling*). {`ggplot2`} acompañará el gráfico con una **leyenda** que explica qué niveles corresponden a qué valores.

Ejemplo: Añadir una variable utilizando mapeo estético de color (`colour`).

```
# Agregando una tercera variable
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG, colour = Type))
```

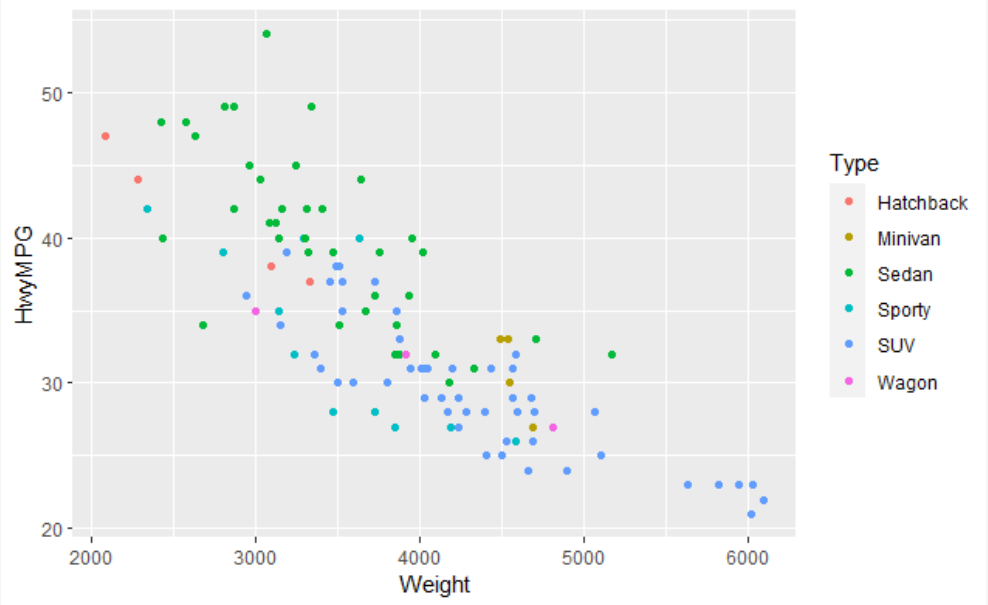


Figura 40: Agregar parámetro estético de color.

En el ejemplo anterior asignamos la variable `Type` a la estética de color, pero podríamos haberla asignado a la estética del tamaño del mismo modo. En este caso, el tamaño de cada punto revela a qué clase pertenece. En este caso recibimos una

advertencia (`warning`), porque mapear una variable no ordenada (`Type`) a una estética ordenada (`size`) no es aconsejable.

Ejemplo: Añadir una variable utilizando mapeo estético de tamaño (`size`).

```
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG, size = Type))
Warning message:
Using size for a discrete variable is not advised.
```

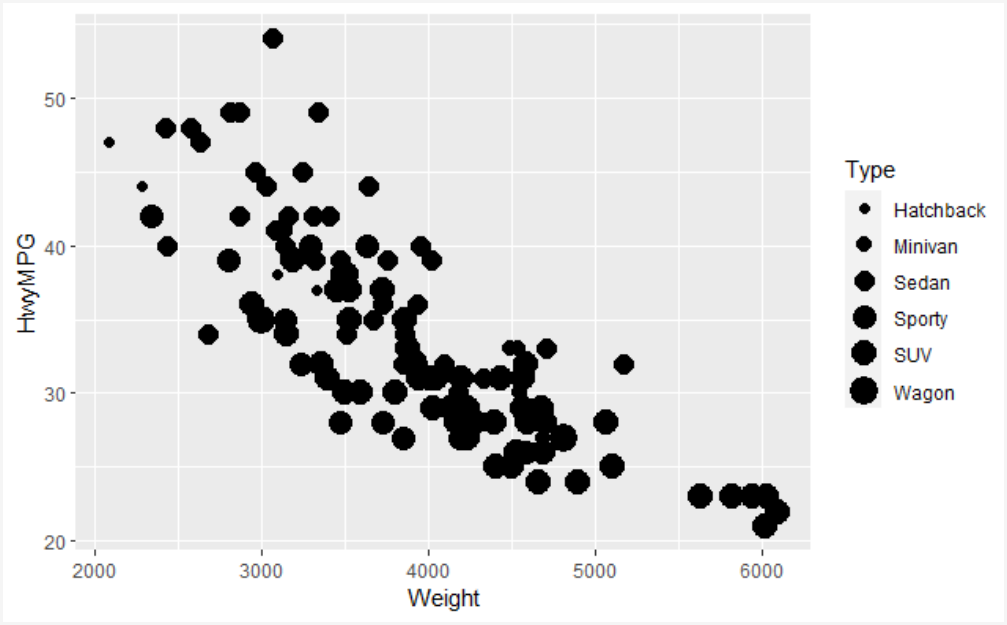


Figura 41: Agregar parámetro estético de tamaño.

También podríamos haber asignado la variable `Type` a la estética `shape` que controla la forma (*shape*) de los puntos, o a la estética `alpha`, que controla la transparencia de los puntos. En el último caso, de nuevo se nos mostrará un mensaje con la misma advertencia que para `size`.

Ejemplo: Añadir una variable utilizando mapeo estético de forma (`shape`) y transparencia (`alpha`).

```
# estética de forma
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG, shape = Type))
```

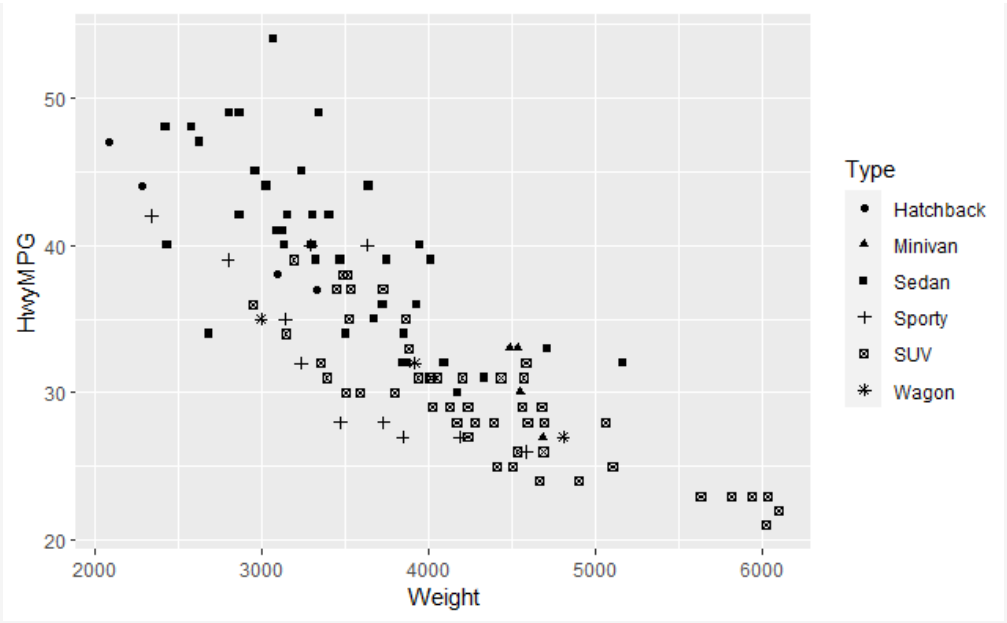


Figura 42: Agregar parámetro estético de forma.

```
# estética de transparencia
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG, alpha = Type))
Warning message:
Using size for a discrete variable is not advised.
```

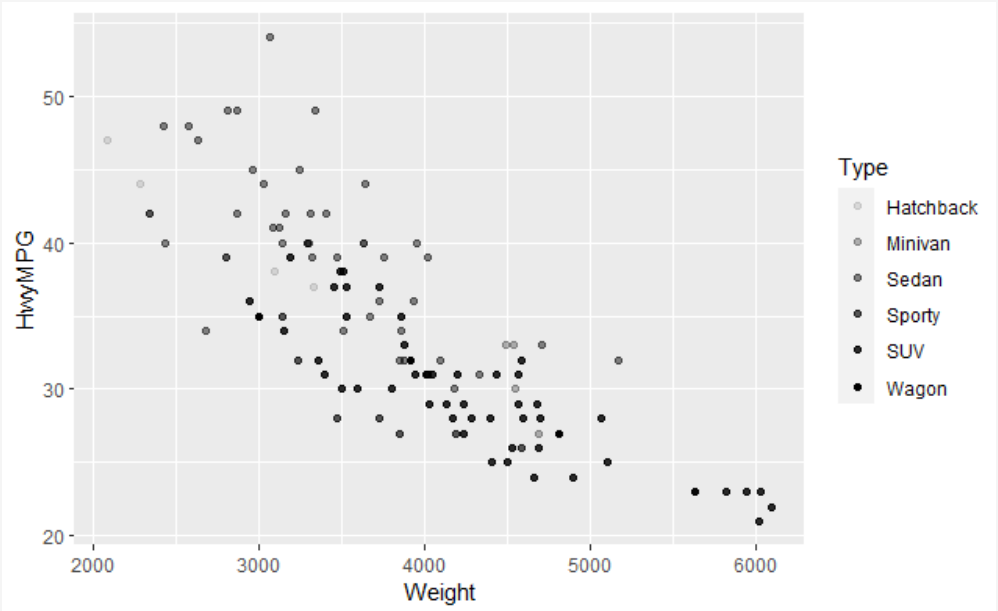


Figura 43: Agregar parámetro estético de transparencia.

Debemos tener presente que `{ggplot2}` solo puede usar seis formas a la vez. De forma predeterminada, los grupos adicionales no se grafican cuando se emplea la estética de la forma.

Otra posibilidad es fijar las propiedades estéticas del `geom` manualmente (y no dentro de la función `aes()`). Por ejemplo, podemos hacer que todos los puntos del gráfico sean de determinado color, forma y tamaño:

Ejemplo: Cambiar estética manualmente

```
# Cambiar estética manualmente
ggplot(data = Cars2020) +
  geom_point(mapping = aes(x = Weight, y = HwyMPG), shape = 21, colour
= "#0098CD", fill = "#C1F5BF", size = 3)
```

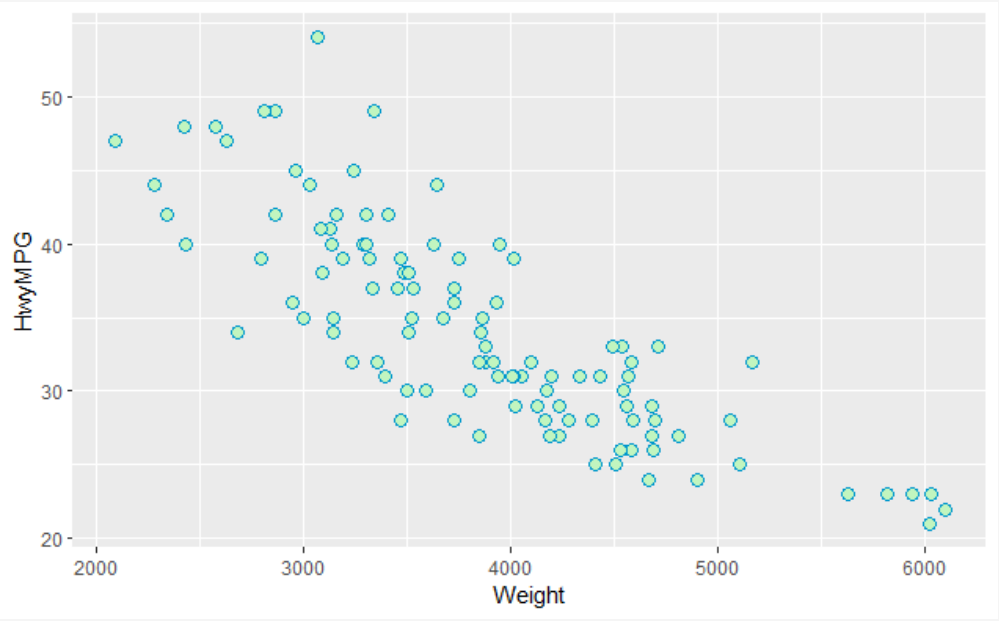


Figura 44: Cambiar estética manualmente.

La diferencia es que ahora el color no transmite información sobre una variable, sino que cambia la apariencia del gráfico. Para establecer una estética de forma manual, debes usar el nombre de la estética como un argumento de la función `geom`; es decir, va fuera de `aes()`. Notar que para la forma podemos utilizar los mismo valores que ya vimos con `pch` (de hecho, podemos usar `pch`, en lugar de `shape`) para la función `plot()` de R base: las formas vacías (0–14) tienen un borde determinado por `colour`; las formas sólidas (15–18) están rellenas con `colour`; las formas rellenas (21–24) tienen el borde de `colour` y están rellenas por `fill`.

Objetos geométricos

Un `geom` es el objeto geométrico usado para representar datos de forma gráfica. Es usual llamar a los gráficos por el tipo de `geom` que utiliza. Por ejemplo, los diagramas de barras usan geoms de barra (`geom_bar()`), los diagramas de líneas usan geoms de línea (`geom_line()`), los diagramas de caja usan geoms de diagrama de caja (`geom_boxplot()`), y así sucesivamente. Podemos usar diferentes geoms para graficar los mismos datos. Por ejemplo, en la Figura 45, la gráfica de la izquierda usa el geom de punto (`geom_point()`), y la gráfica de la derecha usa el geom suavizado (`geom_smooth()`), una línea suavizada ajustada a los datos.

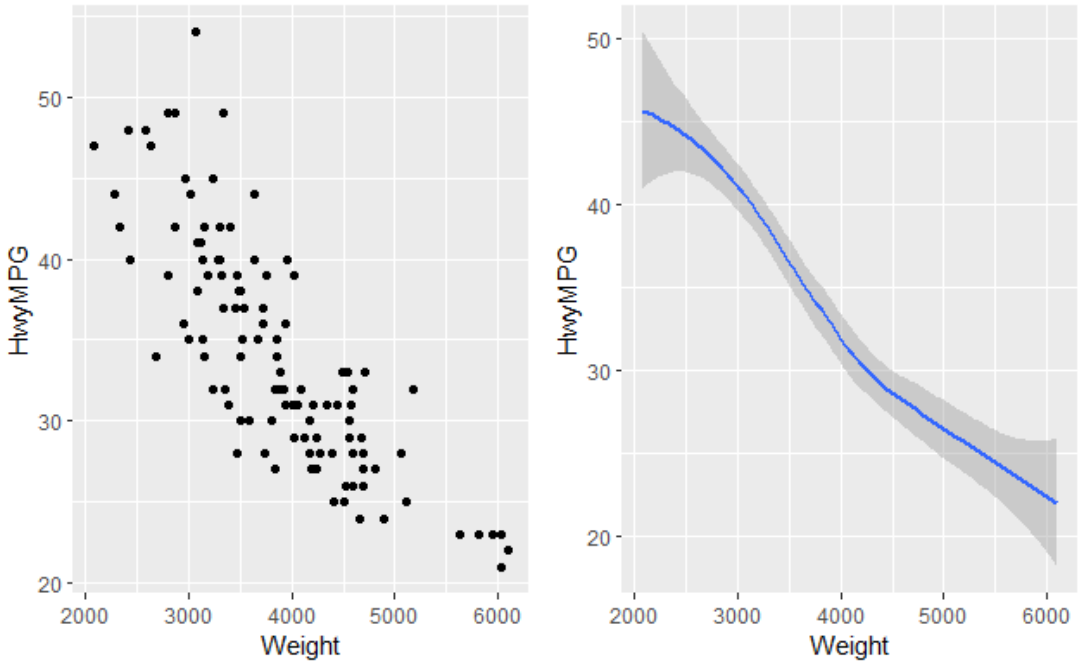


Figura 45: Diferentes geoms para los mismos datos.

Como dice en el mensaje, por defecto `geom_smooth()` suaviza los datos usando el método *loess* (**regresión lineal local**) cuando hay menos de 1000 datos. Seguramente va a ser muy común que quieras ajustar una **regresión lineal *global***. En ese caso, hay que poner `method = "lm"`.

Cada función `geom` en {ggplot2} toma un argumento de `mapping`. Sin embargo, no todas las estéticas funcionan con todos los `geom`. Por ejemplo, podemos establecer la

forma para un punto, pero no puedes establecer la "forma" de una línea. Y análogamente, podemos elegir el tipo de línea (`linetype` o `lty`) pero esta estética no aplica para los puntos.

Hemos dicho que {ggplot2} divide a los gráficos en componentes o capas, en el sentido de que podemos ir añadiendo elementos agregando capas a nuestro dibujo. En particular, podemos hacer esto para graficar más de un geom en un mismo gráfico.

Ejemplo: Grafico con más de un geom.

```
ggplot(data = Cars2020, aes(x = Weight, y = HwyMPG)) +
  geom_smooth(mapping = aes(linetype = Size, colour = Size)) +
  geom_point(mapping = aes(colour = Size))
```

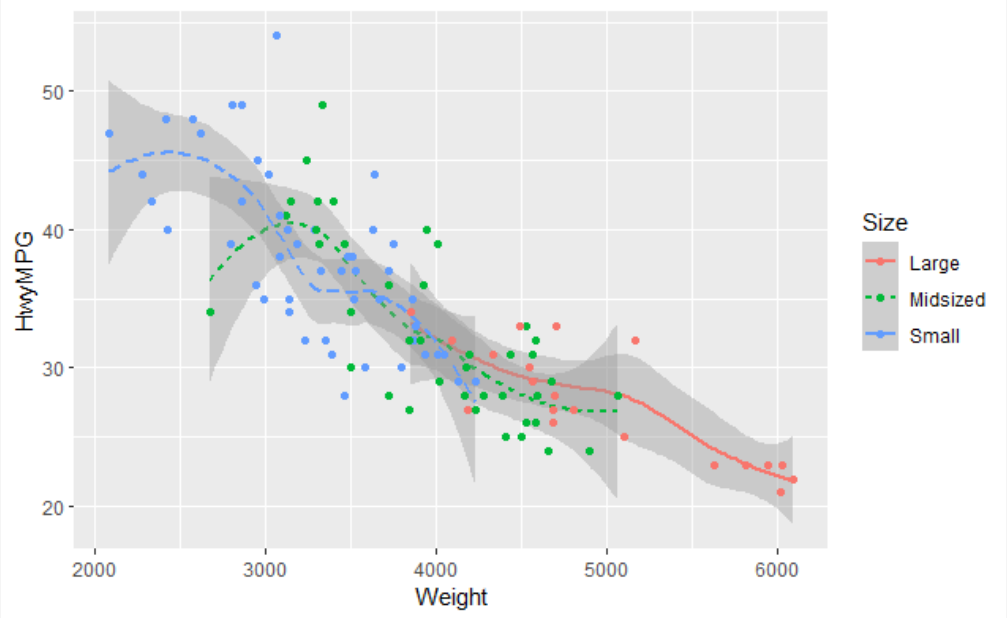


Figura 46: Múltiples geoms en un mismo gráfico.

Como podemos observar, basta con agregar los distintos geoms utilizando nuevas capas precedidas con el símbolo +.

Notemos que, a diferencia de los ejemplos anteriores, hemos incluido el mapeo estético `x` e `y` dentro de la función `ggplot()` (y no dentro de cada `geom_X()`). De esta forma evitamos la repetición pasando un conjunto de mapeos directamente a `ggplot()`. {ggplot2} trata estos mapeos como mapeos globales que se aplican a cada geom en el gráfico. En cambio, si ponemos mapeos en una función `geom_X()`, {ggplot2} los tratará como mapeos locales para la capa en particular. Estas

asignaciones serán usadas para extender o sobrescribir los mapeos globales solo para esa capa. Esto permite mostrar diferentes estéticas en diferentes capas.

Ejemplo: Estéticas globales vs locales

```
ggplot(data = Cars2020, aes(x = Weight, y = HwyMPG)) +
  geom_smooth(mapping = aes(linetype = Size)) +
  geom_point(mapping = aes(colour = Drive))
```

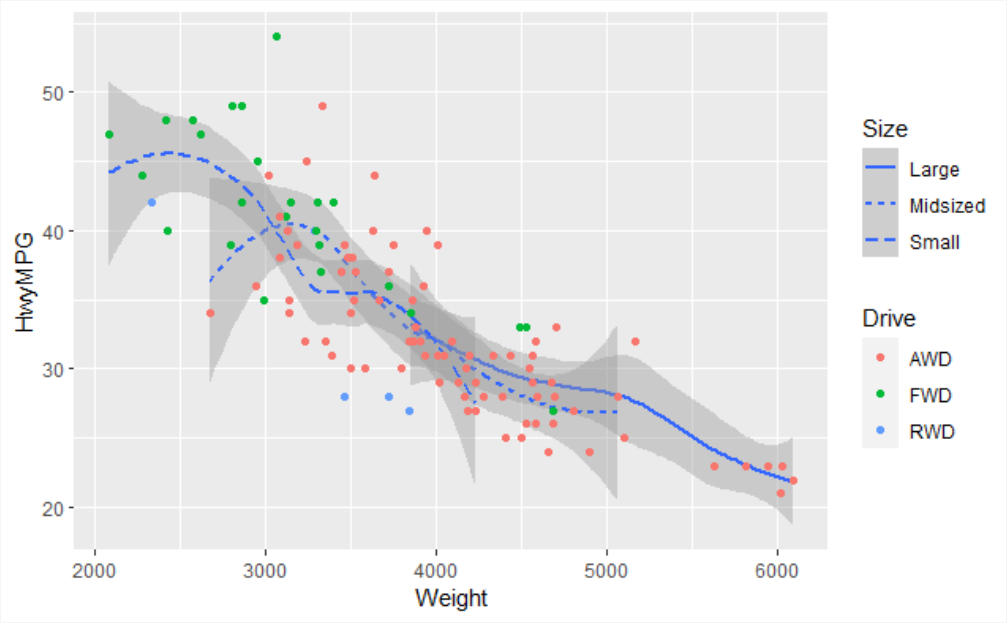


Figura 47: estéticas locales y globales.

De forma análoga, podemos especificar distintos conjuntos de datos (`data`) para cada capa. Por ejemplo, podemos utilizar todo el conjunto de datos para obtener el suavizado y agregare luego los puntos correspondientes solo a un subconjunto de los datos. En el siguiente ejemplo, la línea de suavizado aplica sobre todo el conjunto de datos, pero para `geom_points()` solo consideramos los autos con tracción delantera. El argumento local de datos en `geom_points()` anula el argumento de datos globales en `ggplot()` solo para esa capa. Notar además que el argumento `se = FALSE` de `geom_smooth()` especifica que no se grafique la región de confianza alrededor de la línea de suavizado.

Ejemplo: Graficar capas con diferentes datos

```
ggplot(data = Cars2020, mapping = aes(x = Weight, y = HwyMPG)) +
  geom_point(data = filter(Cars2020, Drive == "FWD"), mapping =
aes(colour = Size)) +
  geom_smooth(se = FALSE, colour = '#810BB8', size = 0.8, linetype =
2)
```

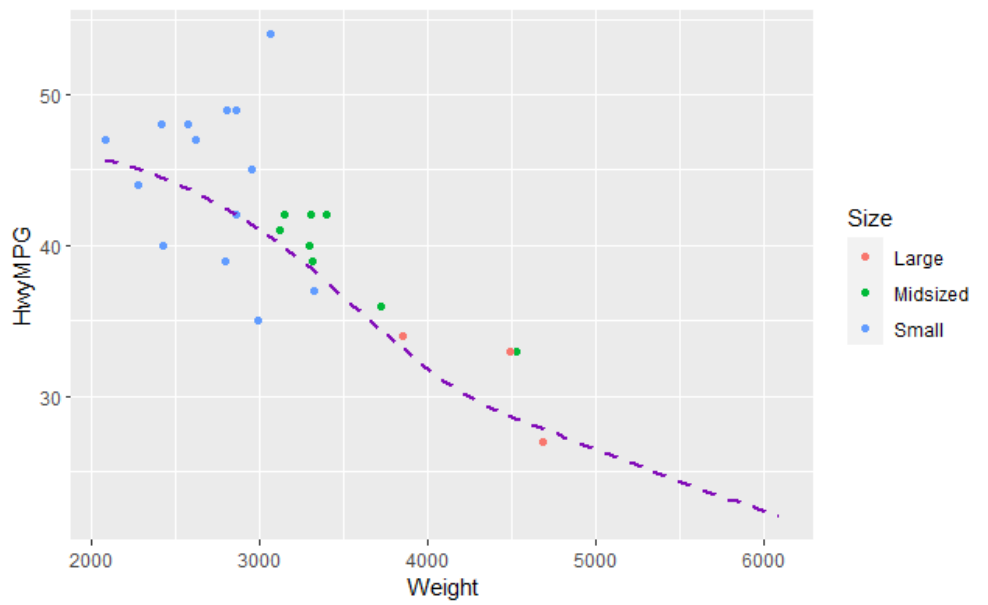


Figura 48: Capas con diferentes conjuntos de datos.

geom_line() o gráficos de líneas

Para hacer un gráfico de líneas usaremos una sintaxis muy parecida a la que vimos para el de dispersión, teniendo en cuenta que `{ggplot2}` trazará una línea de punto a punto. Para ejemplificar el comportamiento de `geom_line()`, vamos a utilizar el conjunto de datos `paises`, contenido en el paquete {datos}. Estos son la versión en español de los datos de `gapminder`, que resumen la progresión de países a través del tiempo, mirando estadísticos como esperanza de vida y PIB.

Comencemos creando un gráfico de líneas ara mostrar la evolución de la esperanza de vida en Argentina.

Ejemplo: Gráfico de líneas con `geom_line()`

```
paises %>% filter(pais == "Argentina") %>%
  ggplot(aes(y = esperanza_de_vida, x = anio)) +
  geom_line()
```

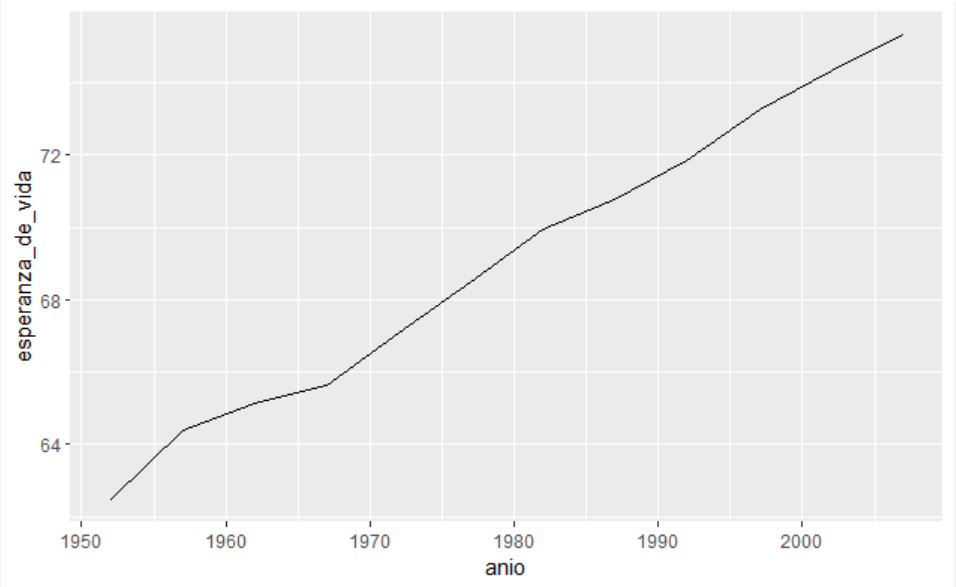


Figura 49: Gráfico de líneas usando la función geom_line().

El gráfico luce bien porque para cada año (que además están ordenados) tenemos un único valor de esperanza de vida. Pero ¿qué ocurre si tenemos más de un país (esto es, más de un valor de la variable `esperanza_de_vida`) para cada año? Para ver eso, repitamos el gráfico, pero considerando ahora todos los países correspondientes al continente americano (`continente == "Américas"`).

Ejemplo: varias observaciones en un mismo punto.

```
paises %>% filter(continente == "Américas") %>%
  ggplot(aes(y = esperanza_de_vida, x = anio)) +
  geom_line()
```

Figura 50: Gráfico de líneas cuando hay varias observaciones para un mismo punto.

Esto no luce muy bien. ¿qué ha pasado? Pues que `{ggplot2()}`, al graficar punto a punto, ha juntado todos los valores de un mismo año. ¿Cómo podemos arreglar esto? Necesitamos que agrupe las observaciones por la variable `país` (es decir, una línea por cada país). Esto lo podemos hacer usando, por ejemplo, la estética `colour` o la estética `group`. La diferencia entre estas dos es que la estética del grupo en sí misma no agrega una leyenda ni una característica distintiva a los geoms. Veámoslo en un ejemplo.

Ejemplo: agrupar observaciones.

```
# con la estética de color
paises %>% filter(continente == "Américas") %>%
  ggplot(aes(y = esperanza_de_vida, x = anio)) +
  geom_line(aes(colour = pais))
```

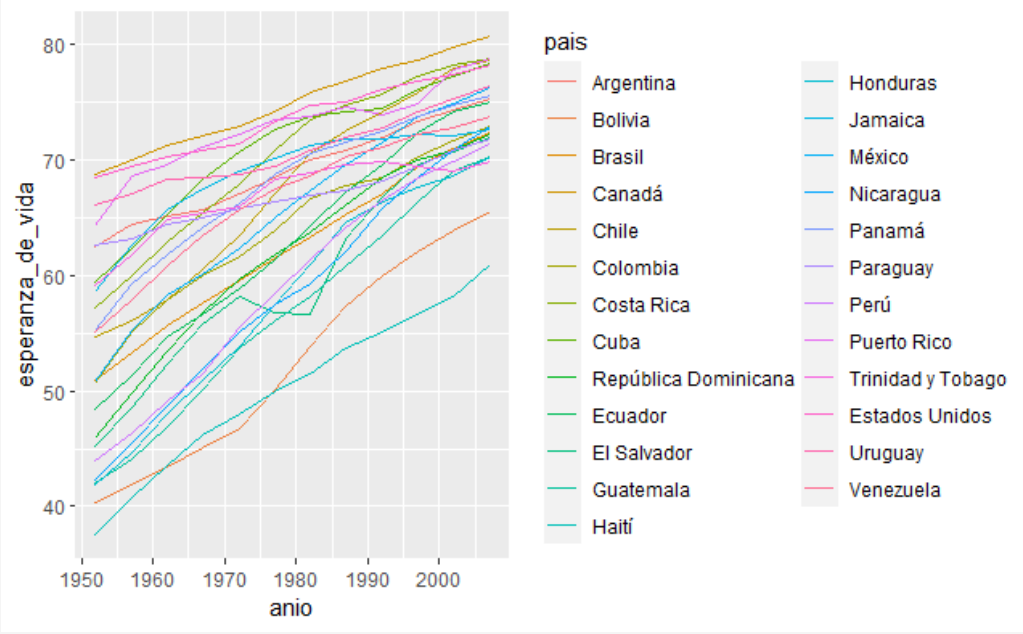


Figura 51: Gráfico de líneas agrupando con la estética de color.

```
# con la estética de grupo
paises %>% filter(continente == "Américas") %>%
  ggplot(aes(y = esperanza_de_vida, x = anio)) +
  geom_line(aes(group = pais))
```

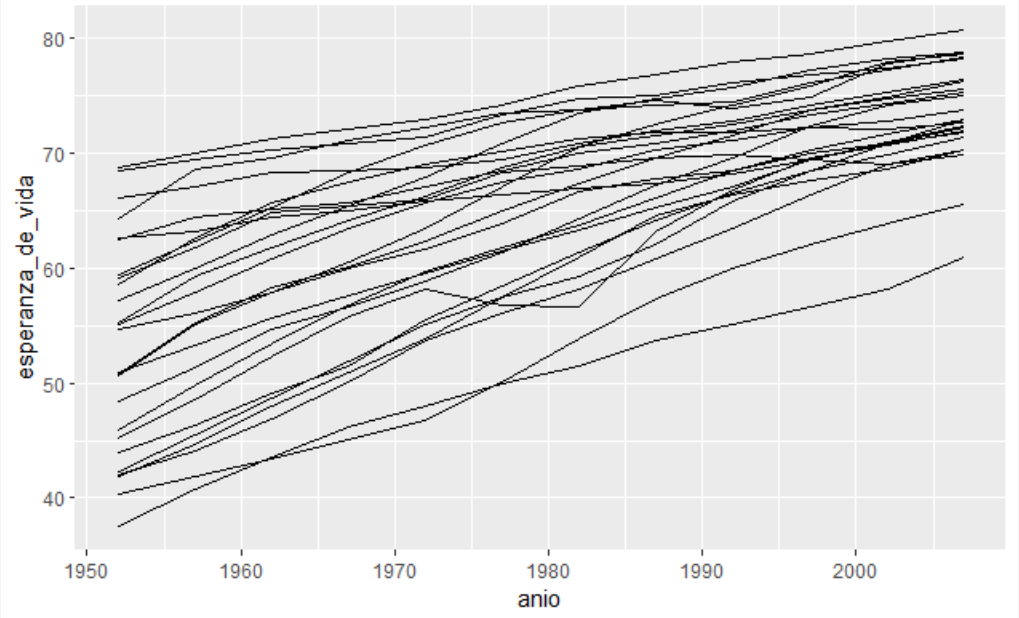


Figura 52: Gráfico de líneas agrupando con la estética de grupo.

Relación entre variables

Muchas veces no es suficiente con mirar los datos crudos para identificar la relación entre las variables, sino que es necesario usar alguna transformación estadística que resalte esas relaciones (por ejemplo, ajustando una recta o calculando promedios, etc.).

Para algunas transformaciones estadísticas comunes, `{ggplot2}` tiene geoms ya programados, pero muchas veces es posible que necesitemos manipular los datos antes de poder hacer un gráfico. A veces esa manipulación será compleja y entonces para no repetir el cálculo muchas veces, guardaremos los datos modificados en una nueva variable. También podemos encadenar la manipulación de los datos y el gráfico resultante.

Por ejemplo, calculemos la **esperanza de vida *media*** por continente y para cada año y luego grafiquemos la `esperanza_de_vida_media` a lo largo de los `anios`:

Ejemplo: Relación entre variables

```
paises %>%
  group_by(continente, anio) %>%
```

```
  summarise(esperanza_de_vida_media = mean(esperanza_de_vida)) %>%
  ggplot(aes(anio, esperanza_de_vida_media)) +
  geom_point()
```

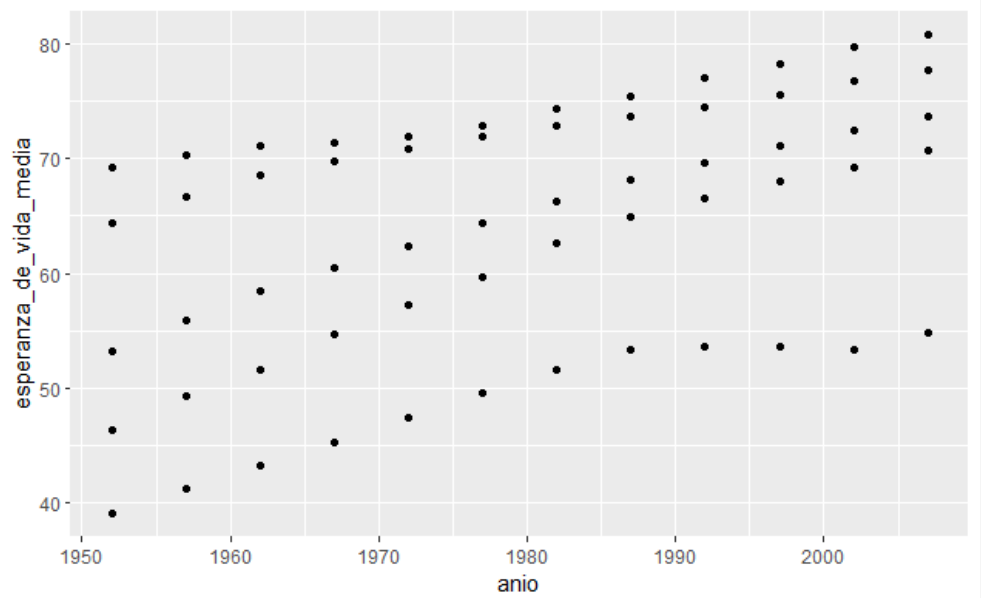


Figura 53: Relación entre variables.

Este gráfico parece mostrar que la esperanza de vida media fue aumentado a lo largo de los años. Podríamos ver esa relación más explícitamente agregando una nueva capa con `geom_smooth()` y vamos a especificar que ajuste un modelo lineal global:

Ejemplo: Relación entre variables (continuación)

```
paises %>%
  group_by(continente, anio) %>%
  summarise(esperanza_de_vida_media = mean(esperanza_de_vida)) %>%
  ggplot(aes(anio, esperanza_de_vida_media)) +
  geom_point(aes(colour = continente)) +
  geom_smooth(method = "lm", colour = '#67BF4C')
```

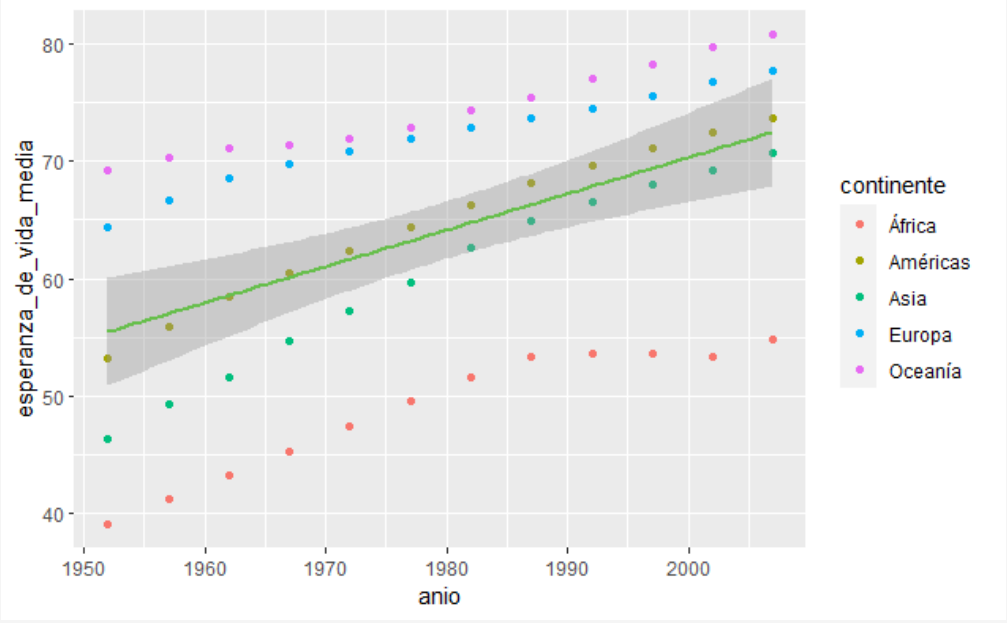


Figura 54: Relación entre variables *(continuación)*.

`geom_boxplot()` o diagramas de caja y bigotes

Para realizar un diagrama de caja utilizamos la geometría `geom_boxplot()`.

Recordemos que los diagramas de cajas son gráficos apropiados para variables de tipo cuantitativas.

Ejemplo: Diagrama de caja

Volvamos a los datos `Cars2020.csv`. Vamos a construir el diagrama de caja de la variable `Length` (que, recordemos, es el largo del vehículo, medido en pulgadas).

```
# diagrama de caja básico
ggplot(data = Cars2020, aes(y = Length)) +
  geom_boxplot()
```

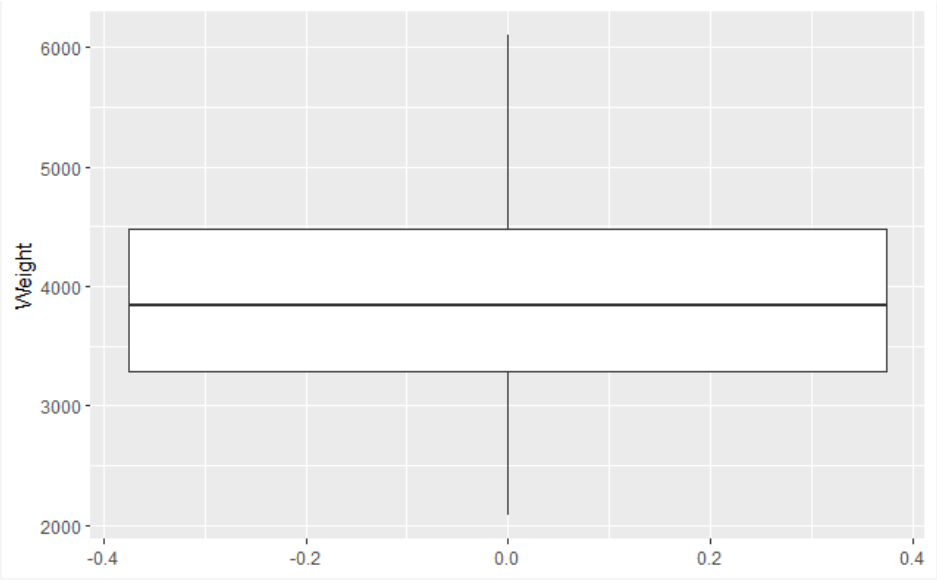


Figura 55: Diagrama de caja y bigotes básico.

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

```
# eliminamos los valores en el eje X
ggplot(data = Cars2020, aes(x = "", y = Length)) +
  geom_boxplot()
```

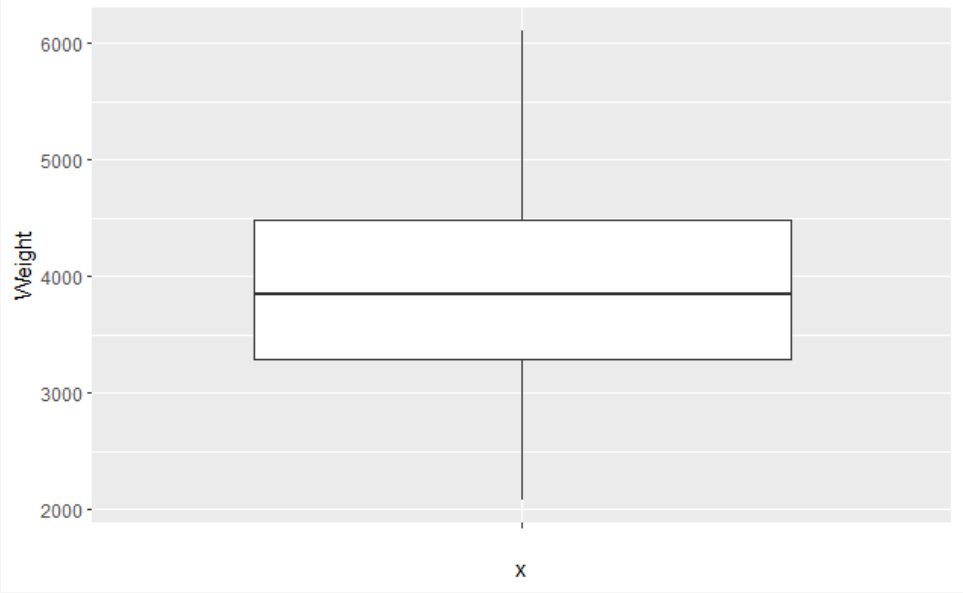


Figura 56: Diagrama de caja y bigotes básico: eliminar escala del ejeX.

Agregando barras de error con `stat_boxplot`

El diagrama de caja y bigotes de `{ggplot2}` no añade las líneas horizontales de los bigotes (como lo hace `boxplot()` de R base), pero puedes agregarlas con `stat_boxplot`, estableciendo `geom = "errorbar"`. Además podemos cambiar su ancho con `width`.

```
# Agregamos las barras horizontales en los bigotes
ggplot(data = Cars2020, aes(x = "", y = Length)) +
```

```
  stat_boxplot(geom = "errorbar", width = 0.15) +
  geom_boxplot()
```

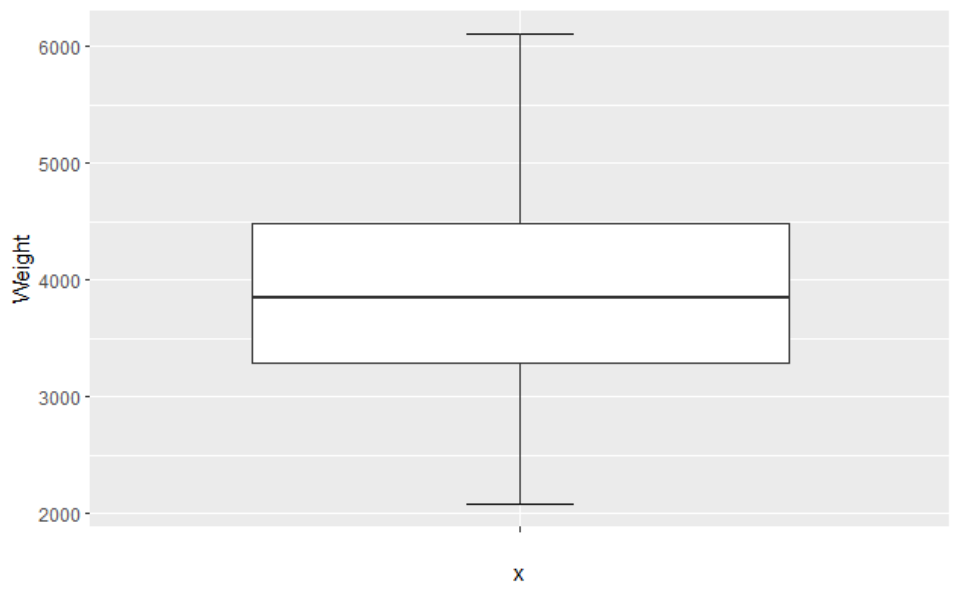


Figura 57: Diagrama de caja y bigotes básico: líneas horizontales en los bigotes.

Personalización del diagrama de caja

Los *box plots* hechos con `{ggplot2}` se pueden personalizar haciendo uso de los argumentos de las funciones `stat_boxplot` y `geom_boxplot`. Veamos ejemplos en los que cambiamos los colores y los tipos de líneas de los gráficos y modificamos el ancho de la caja.

Ejemplo: personalizar *box plots*.

```
# personalizando los boxplot
ggplot(Cars2020, aes(x = "", y = Length)) +
  stat_boxplot(geom = "errorbar",
               width = 0.1,
               color = "#090D45") +  # Color barras error
  geom_boxplot(fill = "#E0F2FC",     # Color de relleno de la caja
               alpha = 0.5,          # Transparencia
               color = "#090D45",    # Color del borde de la caja
               outlier.colour = "#0098CD",   # Color atípicos
```

```
width = 0.2)            # Ancho de la caja
```

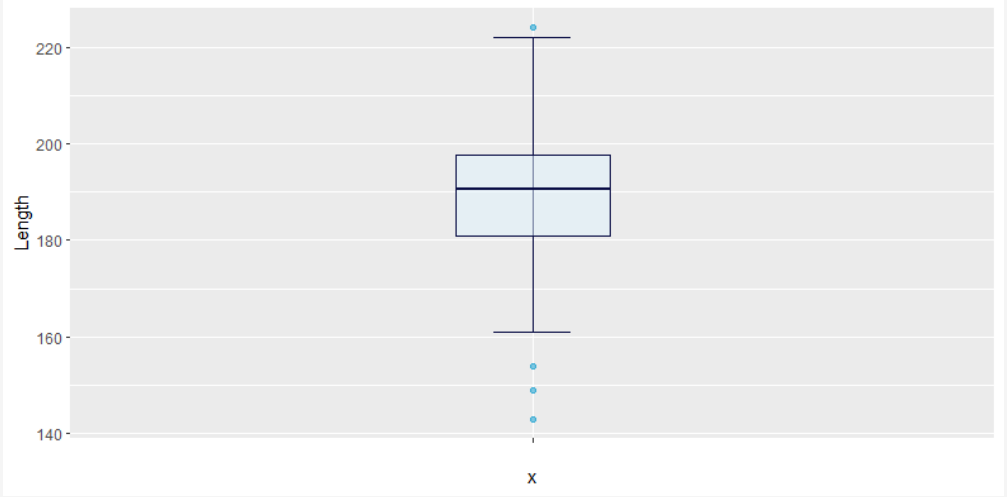


Figura 58: Personalizar el diagrama de caja y bigotes.

Otra posibilidad interesante es añadir estadísticas a un gráfico. Una *stat* (o transformación estadística), transforma los datos, normalmente resumiéndolos de alguna manera. Por ejemplo, una estadística útil es el suavizador, que calcula la media suavizada de `y`, condicional a `x`. Sin saberlo, ya ha usado algunas estadísticas de `{ggplot2}` porque se usan detrás de escena para generar muchos *geoms* importantes. Algunos ejemplos de stats y geoms asociados se muestran en la Tabla 5.

stats	geoms
`stat_bin()`	`geom_bar(), geom_histogram, geom_freqpoly()`
`stat_boxplot()`	`geom_boxplot()`
`stat_quantile()`	`geom_quantile()`
`stat_smooth()`	`geom_smooth()`
`stat_sum()`	`geom_count()`

Tabla 5: Stats y geoms compplementarios.

Hay dos formas de utilizar estas funciones. Podemos agregar una la función `stat_()` y anular la `geom` predeterminada, o agregar una `geom_()` y anular la estadística predeterminada:

Ejemplo: Agregar estadística a un gráfico

```
ggplot(Cars2020, aes(x='', Length)) +
  geom_boxplot(width = 0.2) +
  stat_summary(geom = "point", fun = "mean", colour = "red", size = 4)

ggplot(Cars2020, aes(x='', Length)) +
  geom_boxplot(width = 0.2) +
  geom_point(stat = "summary", fun = "mean", colour = "red", size = 4)
```

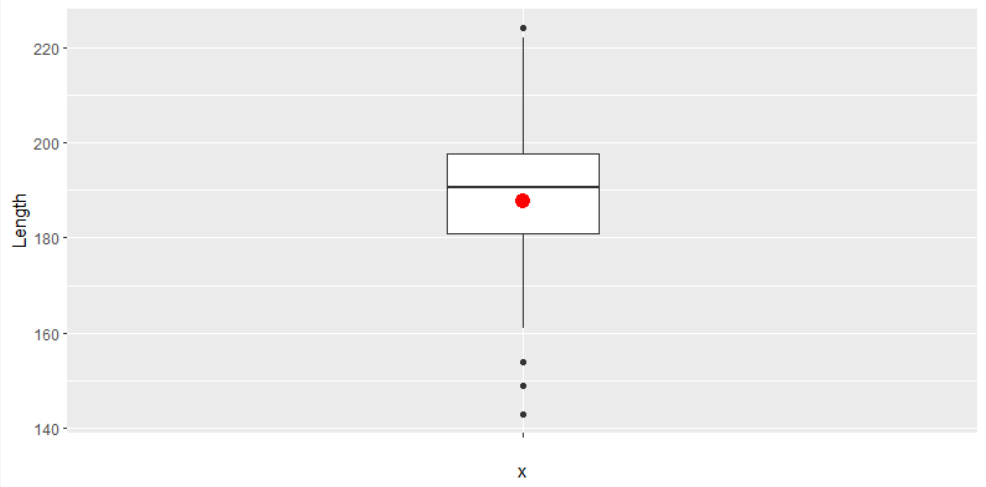


Figura 59: Añadir la media al boxplot.

También podemos crear un gráfico de cajas para una variable cuantitativa en función de otra cualitativa. Ya vimos que esto implica construir un diagrama de cajas por cada categoría o nivel de la variable cualitativa.

Ejemplo: Boxplot para una variable cuantitativa en función de otra cualitativa

```
ggplot(Cars2020, aes(x = Size, y = Length)) +
  stat_boxplot(geom = "errorbar",
               width = 0.1,
               color = "#090D45") +  # Color barras error
  geom_boxplot(fill = "#E0F2FC",     # Color de relleno de la caja
               color = "#090D45",    # Color del borde de la caja
               outlier.colour = "#0098CD",   # Color atípicos
               width = 0.4) +
  geom_point(stat = "summary", fun = "mean", colour = "red", size = 3)
+
  scale_x_discrete(labels=c("Grande","Mediano", "Pequeño"))
```

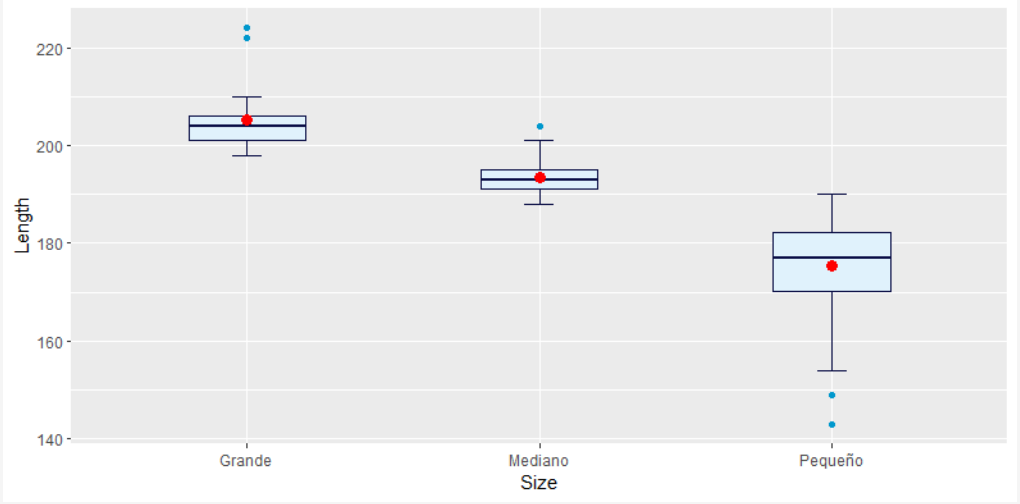


Figura 60: *Boxplot* por categoría de una variable cualitativa.

geom_histogram() o histograma

Sabemos que, en el caso de variables cuantitativas, podemos también representar la distribución de los datos utilizando histogramas. En `{ggplot2}` la función que nos permite crear histogramas es `geom_histogram()`.

En `{ggplot2}` resulta muy fácil construir histogramas por grupos (lo que es bastante más laborioso de conseguir trabajando con las funcionalidades gráficas de R base). Continuaremos con el ejemplo de `Length`, del conjunto de datos `Cars2020`, para demostrar la función de `geom_histogram()` a continuación.

Ejemplo: Creación de histogramas con `geom_histogram()`

```
# histogramas
iz <- ggplot(Cars2020, aes(x = Length)) +
  geom_histogram(colour = "white", fill = "blue") +
  labs(x = "Longitud del vehículo", y = "Frecuencia Absoluta") +
  theme(axis.title = element_text(size = 10, face = "bold"))

der <- ggplot(Cars2020, aes(x = Length)) +
  geom_histogram(binwidth = 10, colour = "white", fill = "blue") +
  labs(x = "Longitud del vehículo", y = "Frecuencia Absoluta") +
  theme(axis.title = element_text(size = 10, face = "bold"))

# los disponemos en un arreglo de dos columas
ggpubr::ggarrange(iz, der)
```

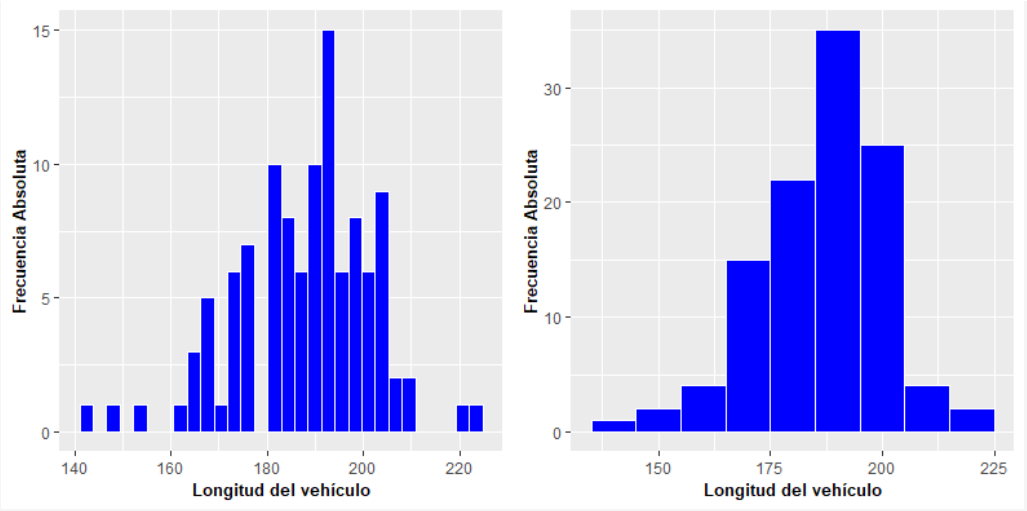


Figura 61: Histogramas

Si en lugar de frecuencias absolutas, queremos hacer el histograma en escala de densidades, deberemos pasar `aes(y = ..density..)` a `geom_histogram()`.

```
ggplot(Cars2020, aes(x = Length)) +
  geom_histogram(aes(y = ..density..), colour = "white", fill = "blue")
+
  labs(x = "Longitud del vehículo", y = "Densidad")
```

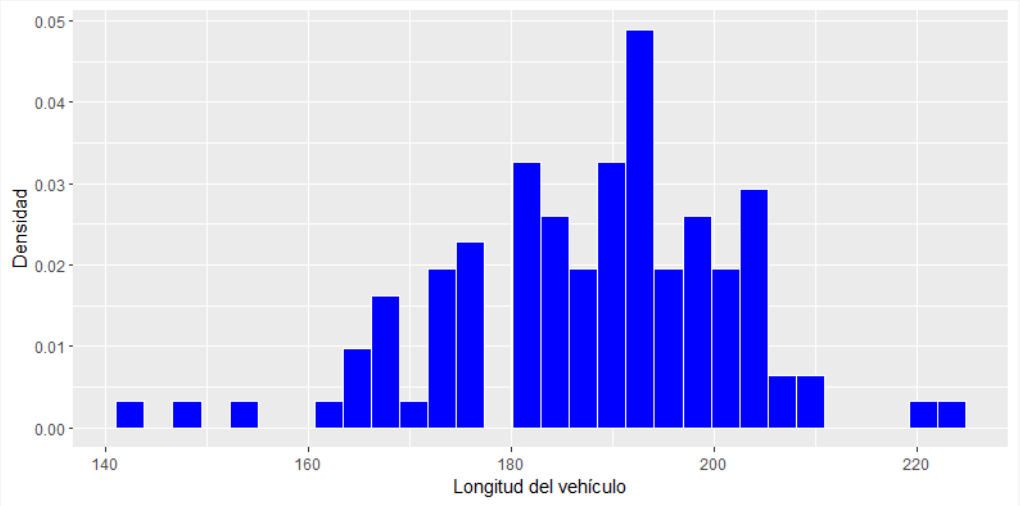


Figura 62: Histograma en escala de densidad.

En el primer caso, `geom_histogram()` representa cantidad de observaciones (datos) en cada intervalo (es decir, el alto de cada barra es igual a la frecuencia del intervalo). En el segundo caso, para mostrar menos barras, especificamos el parámetro `binwidth` dentro de la función `geom_histogram()`. Como podemos apreciar, el número de intervalos se cambia utilizando el parámetro `binwidth`, donde su valor representa el ancho del intervalo (en nuestro caso es igual a 10). El número de `bins` (intervalos)

predeterminado es 30, pero no necesariamente todos tendrán una barra ya que eso depende de la distribución de los datos.

En el tercer caso, el histograma se construye en escala de densidad, lo que significa que ahora el área de cada barra es igual a la proporción de observaciones en el intervalo.

En los tres casos indicamos `colour = "white"` para especificar el color blanco del borde de cada barra y `fill = "blue"` para elegir el color de relleno de las barras.

Notar que al utilizar la configuración predeterminada de `geom_hitogram()` R retorna el mensaje `` `stat_bin()` using `bins = 30`. Pick better value with `binwidth` ``. Esto es simplemente porque no especificamos la cantidad de intervalos (o el ancho de estos) y por lo tanto se utilizó la cantidad de `bins` por defecto.

Histogramas por grupos

Para crear un histograma por grupo en `{ggplot2}` simplemente tenemos que pasar la variable numérica al eje X y la categórica como estética de color (relleno) dentro de `aes()`.

Ejemplo: Histograma por grupo

```
ggplot(Cars2020, aes(x = Length, fill = Size)) +
  geom_histogram(binwidth = 10, colour = "white") +
  xlab("Longitud del vehículo") +
  ylab("Frecuencia")
```

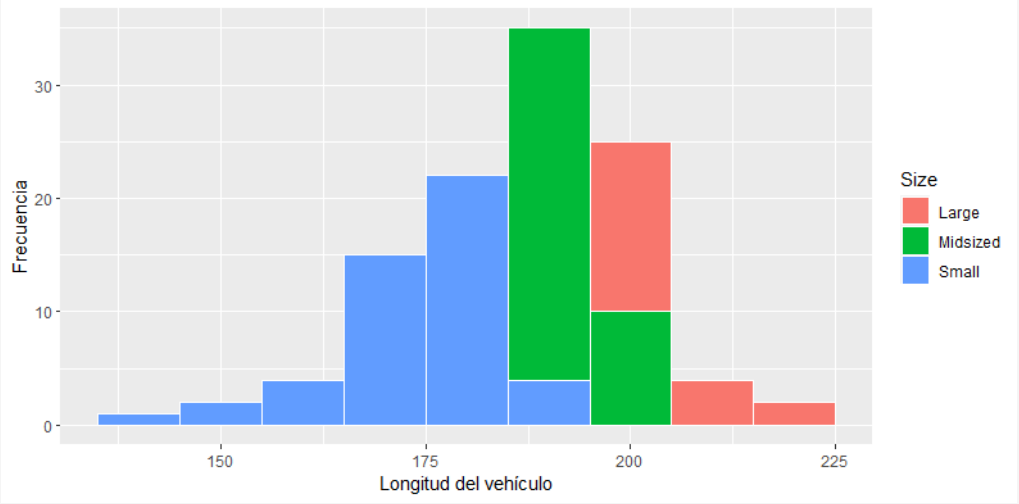


Figura 63: Histogramas por grupo.

Por defecto, si los histogramas se solapan, las barras se superponen. Podemos modificar este comportamiento cambiando la posición a `identity` (y usar colores con trasparencia) o a `dodge` como en los ejemplos siguientes.

- **Posición `identity`**

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

- **Posición `dodge`**

  La otra opción es usar `position = "dodge"`, que agrega un espacio entre cada barra de forma que puedas ver ambos histogramas.

Ejemplo: Comportamiento con histogramas que se solapan.

```
# Histograma por grupo position = "identity"
ggplot(Cars2020, aes(x = Length, colour = Size, fill = Size)) +
  geom_histogram(binwidth = 10, alpha = 0.5, position = "identity") +
  labs(title = 'position = "identity"', x = "Longitud del vehículo", y
= "Frecuencia") +
  theme(plot.title     =     element_text(color="#615E5E",     size=11,
face="bold.italic", hjust = 0.5))

# Histograma por grupo position = "dodge"
ggplot(Cars2020, aes(x = Length, colour = Size, fill = Size)) +
  geom_histogram(binwidth = 10, alpha = 0.5, position = "dodge") +
  xlab("Longitud del vehículo") +
  ylab("Frecuencia") +
  ggtitle(label = 'position = "dodge"') +
```

```
  theme(plot.title = element_text(color="#615E5E", size=11,
face="bold.italic", hjust = 0.5))
```

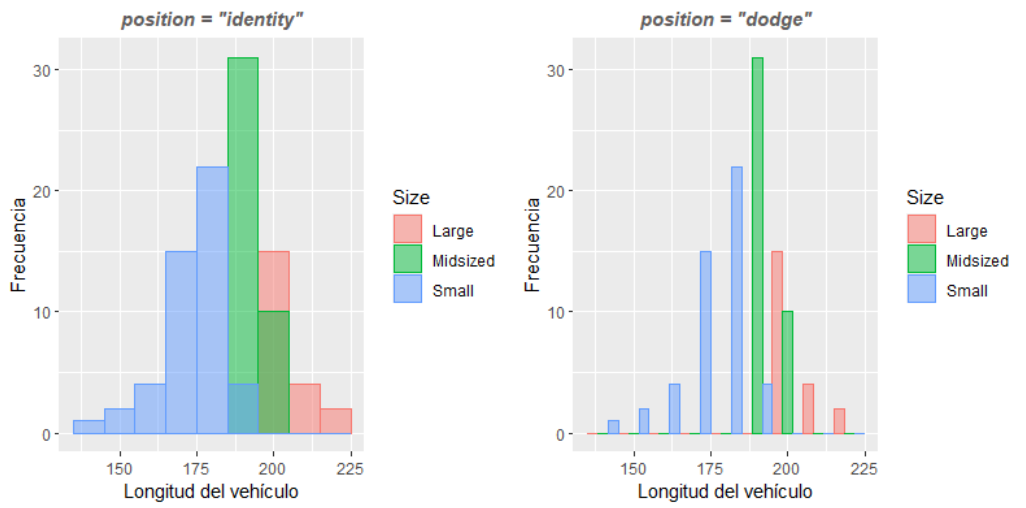


Figura 64: Especificando `position` para histogramas que se solapan.

También podemos disponer el gráfico en forma de una matriz de gráficos por fila y columna por una variable seleccionada. Para esto podemos utilizar las funciones `facet_grid()` o `facet_wrap()`.

```
ggplot(Cars2020, aes(x = Length, fill = Size)) +
  geom_histogram(binwidth = 10, colour = "white") +
  xlab("Longitud del vehículo") +
  ylab("Frecuencia") +
  facet_wrap(~ Size, nrow = 1, labeller=labeller(Size = c(Large =
"Grande", Midsized = "Mediano", Small = "Pequeño"))) +
  theme(legend.position="none",
        strip.text.x = element_text(size=8, angle=0))
```

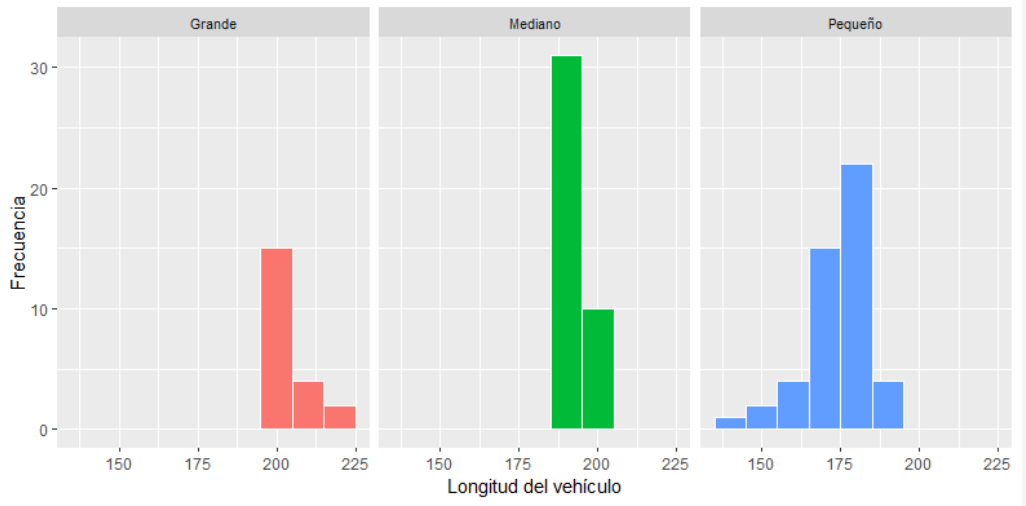


Figura 65: Histogramas por grupo dispuestos en cuadrícula.

En el [este enlace](), encontrarás más detalles sobre cómo utilizar estas funciones.

Como podemos observar en los últimos ejemplos, para agregar los nombres de los ejes y un título al gráfico, podemos hacerlo de dos maneras: utilizando la función `labs(title = 'position = "identity"', x = "Etiqueta eje X", y = "Etiqueta eje Y")` o bien especificando `ggtitle(label = 'position = "dodge"')`, `xlab("Etiqueta eje X")` y `ylab("Etiqueta eje Y")`. Luego, con la función `theme()` podemos personalizar estos títulos y/o etiquetas.

`geom_bar()` o diagrama de barras

El diagrama de barras, como ya sabemos, puede utilizarse para representar variables categóricas (atributos u ordinales) y variables cuantitativas discretas.

La función principal para crear gráficos de barras en `{ggplot2}` es `geom_bar()`. Por **defecto**, esta función **cuenta el número de ocurrencias para cada categoría** de la variable.

Sin embargo, si el conjunto de datos ya contiene el recuento para cada grupo (es decir, tenemos una tabla de frecuencias y no los datos crudos, debemos pasar `stat = "identity"` dentro de `geom_bar`.

Ejemplo: Gráfico de barras básico.

```
# gráfico de barras con datos crudos
ggplot(Cars2020, aes(x = Drive)) +
  geom_bar()

# gráfico de barras con tabla de frecuencias
df_aux = data.frame(
  traccion = c("Delantera", "Trasera", "Total"),
  frecuencias = c(25, 5, 80))

ggplot(df_aux, aes(x = traccion, y = frecuencias)) +
  geom_bar(stat = "identity")
```

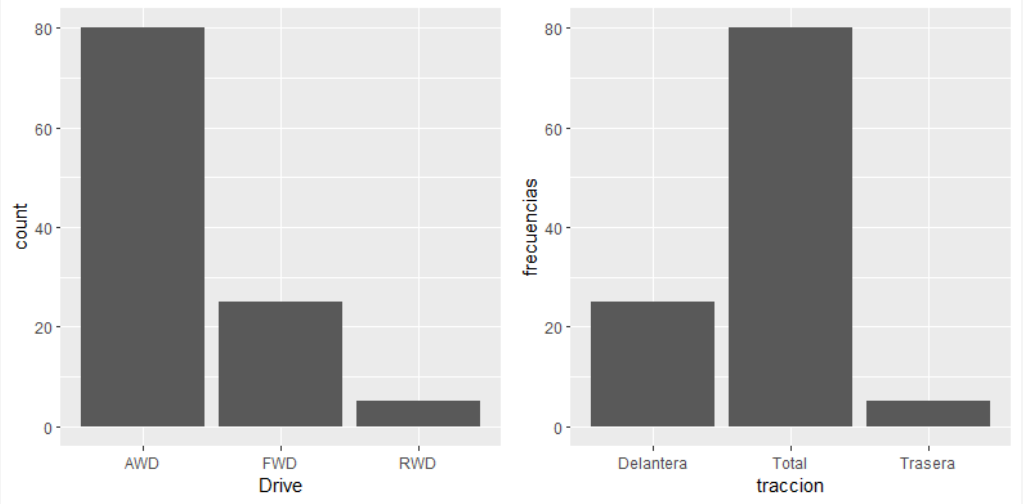


Figura 66: Gráfico de barras básico.

Orden de las barras

El orden por defecto de las barras depende de los niveles de la variable categórica que se transforma internamente en factor. En nuestro ejemplo puedes comprobar que el orden de las barras corresponde con `levels(Cars2022$Drive)`, en el primer caso, y `levels(as.factor(df_aux$traccion))`, en el segundo. Sin embargo, puede que queramos reordenar las barras. Esto podemos hacerlo de diferentes formas: cambiando los límites con `scale_x_discrete`, modificando el orden de los niveles con `factor` o incluso utilizando la función `reorder`.

Ejemplo: Reordenar las barras con `scale_x_discrete`

```
# orden de las barras
ggplot(Cars2020, aes(x = Drive)) +
  geom_bar() +
  scale_x_discrete(
    limits = c("FWD", "RWD", "AWD"),
    labels = c(
      "FWD" = "Delantera",
      "RWD" = "Trasera",
      "AWD" = "Total"
    )
  )
```

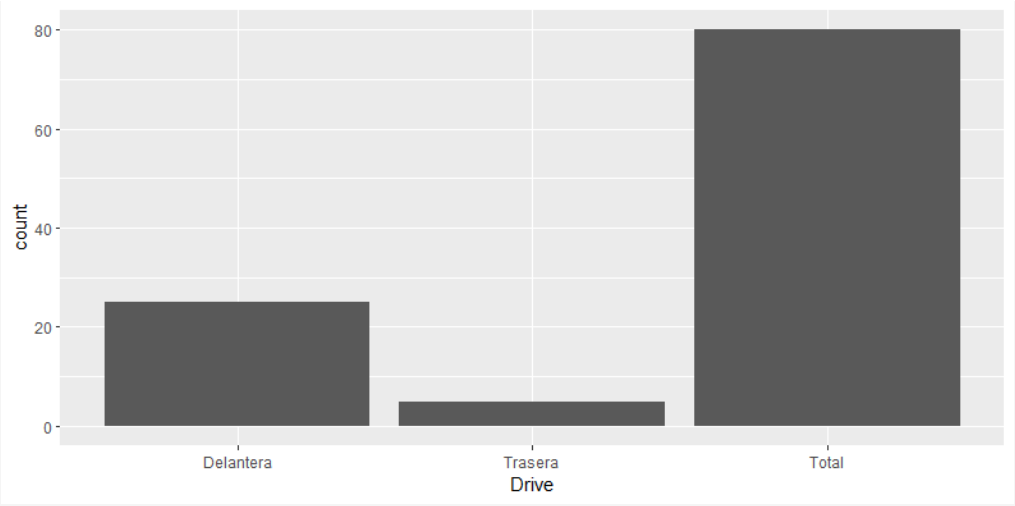


Figura 67: Reordenar las barras.

Usamos `limits` para especificar que categorías y en qué orden se graficarán, y usamos `labels` cuando queremos cambiar la etiqueta que se mostrará en cada barra.

Agregar etiquetas a las barras

En algunas ocasiones resulta de interés agregar etiquetas para mostrar la altura de cada barra o un texto describiendo las barras. Para ello puedes utilizar la función `geom_text` (o `geom_label`) y establecer las etiquetas dentro del argumento `label` de `aes`, así como cambiar su ajuste vertical con `vjust`.

Ejemplo: Argegar etiquetas o texto sobre las barras.

```
# Etiqueta dentro de cada barra
ggplot(Cars2020, aes(x = factor(Drive, levels = c("FWD", "RWD",
"AWD")))) +
  geom_bar() +
  geom_text(aes(label = ..count..),
            stat = "count",
            vjust = 1.5,
            colour = "white",
            size = 3) +
  labs(x="Tracción del vehículo", y="Número de autos")

# Etiqueta encima de cada barra
ggplot(Cars2020, aes(x = factor(Drive, levels = c("FWD", "RWD",
"AWD")))) +
  geom_bar() +
  geom_text(aes(label = ..count..),
            stat = "count",
```

```
            vjust = -1,
            colour = "black",
            size = 3) +
  ylim(c(0, 85)) + # Incrementa los límites del eje Y si es necesario
  labs(x="Tracción del vehículo", y="Número de autos")
```

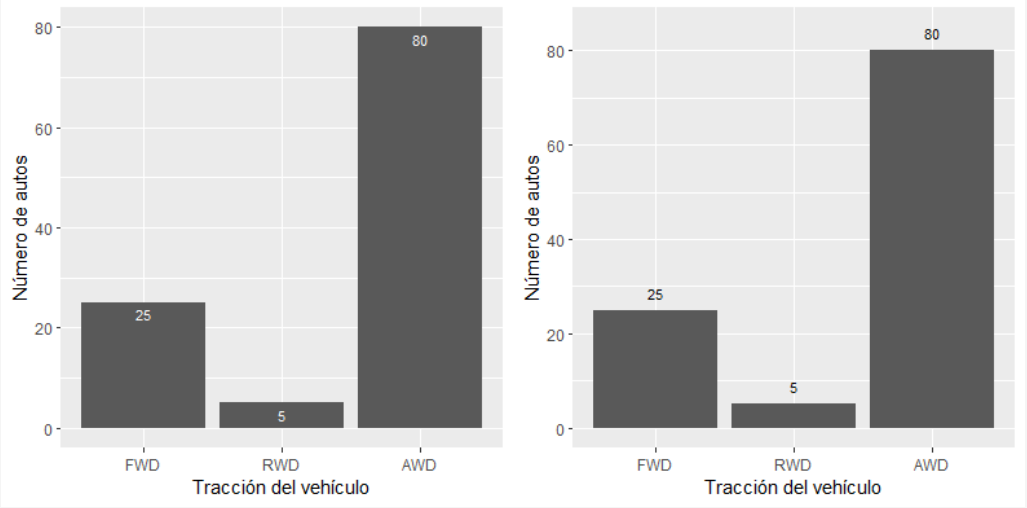


Figura 68: Agregar etiquetas a cada barra.

Como ya hemos visto por los gráficos anteriores, podemos personalizar los colores ya sea asignando un mismo color a todas las barras, o pintando de diferente color las barras según las categorías de la variable.

Cuando se colorean las barras por grupo los colores serán los de la paleta por defecto de ggplot2. Para sobrescribir los colores podemos utilizar otra paleta de colores o seleccionar los colores que queramos con la función `scale_fill_manual`, tanto con un vector ordenado de colores como con un vector con nombres, donde los nombres son las diferentes categorías.

Ejemplo: Personalizando colores de las barras.

```
ggplot(Cars2020, aes(x = Drive, fill = Drive)) +
  geom_bar() +
  scale_x_discrete(
    limits = c("FWD", "RWD", "AWD"),
    labels = c(
      "FWD" = "Delantera",
      "RWD" = "Trasera",
      "AWD" = "Total"
    )
  ) +
  scale_fill_manual(values = c("#0098CD", "#DB8100", "#CD0098"))
```

```
# O usar un vector con nombres donde los nombres son las categorías
ggplot(Cars2020, aes(x = Drive, fill = Drive)) +
  geom_bar() +
  scale_x_discrete(
    limits = c("FWD", "RWD", "AWD"),
    labels = c(
      "FWD" = "Delantera",
      "RWD" = "Trasera",
      "AWD" = "Total"
    )
  ) +
  scale_fill_manual(values = c("FWD" = "#DB8100", "RWD" = "#CD0098",
"AWD" = "#0098CD"))
```

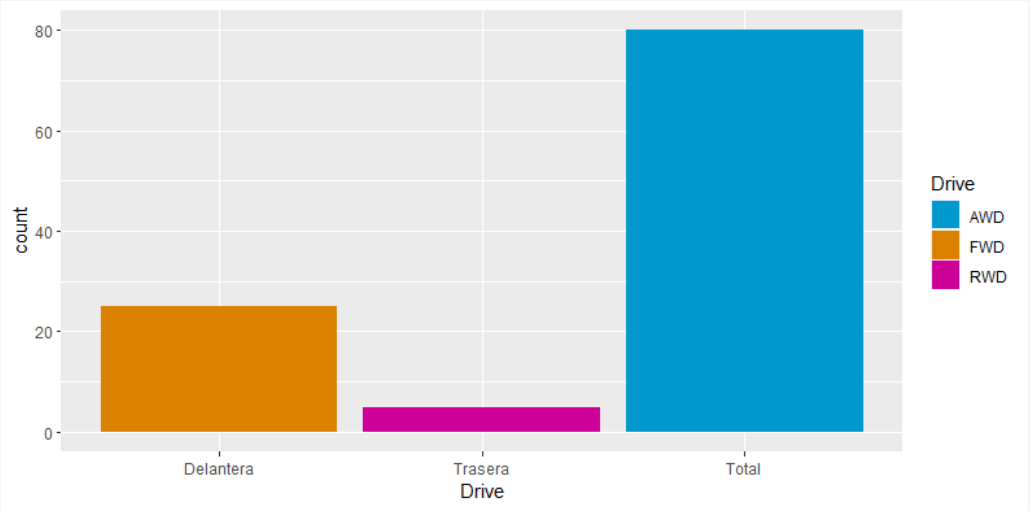


Figura 69: Cambiar los colores de las barras.

Temas en `{ggplot2}`

Para finalizar esta introducción a gráficos con `{ggplot2}`, veamos cómo podemos hacer para cambiar detalles de la apariencia del gráfico que no tienen que ver con los datos, como el tamaño, fuentes y color de los títulos, pero también de los puntos, las líneas, el fondo del gráfico, la apariencia de las *grid-lines*, el lugar para las leyendas, etc. Para esto contamos con las "funciones de tema"; todas ellas comienzan con `theme_()`.

En general con las funciones `theme_()` podemos cambiar/ajustar cualquier elemento del gráfico, con la excepción de la propia representación de los datos (ya sabemos que esto se hacen con las funciones `geom_()`). Estos elementos afectan a la apariencia y detalles del gráfico, pero no a la relación entre variables que se muestra realmente en el gráfico.

{ggplot2} incorpora un conjunto de "temas" que podemos utilizar para cambiar la apariencia del gráfico a nuestro gusto. En este enlace, encontrarás todos los temas que trae incorporado {ggplot2}. El tema que usa por defecto ggplot2 es theme_gray(). Veamos algunos otros *temas* en acción:

Ejemplo: Los temas de {ggplot2}

```
p <- ggplot(Cars2020, aes(x = Drive, fill = Drive)) +
  geom_bar(alpha = 0.5) +
  scale_x_discrete(
    limits = c("FWD", "RWD", "AWD"),
    labels = c(
      "FWD" = "Delantera",
      "RWD" = "Trasera",
      "AWD" = "Total"
    )
  ) +
  scale_fill_manual(values = c("FWD" = "#DB8100", "RWD" = "#CD0098",
"AWD" = "#0098CD"))

uno <- p + ggtitle("theme_gray()") + theme_gray() #- tema por defecto
dos <- p + ggtitle("theme_light()") + theme_light()
tres <- p + theme_dark()
cuatro <- p + ggtitle("theme_classic()") + theme_classic()
cinco <- p + ggtitle("theme_minimal()") + theme_minimal()
seis <- p + ggtitle("theme_void()") + theme_void()

ggpubr::ggarrange(uno, dos, tres, cuatro, cinco, seis)
```

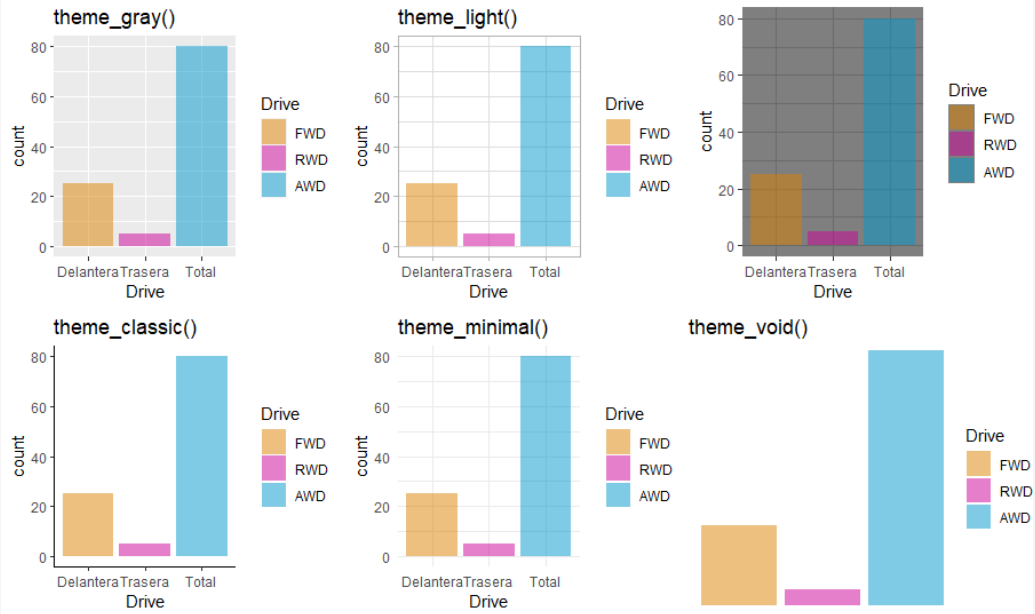


Figura 70: Cambiar apariencia con theme_().

El paquete de R {ggthemes} incorpora una amplia lista de temas adicionales, algunos de ellos tratan de replicar el estilo de corporaciones famosas como *The Economist* o Stata. Algunos de los temas ahí proporcionados, también vienen con sus escalas de color correspondientes. Veamos algunos (la lista completa de temas disponibles en el paquete la puedes encontrar aquí y en este enlace encontrás ejemplos de cada tema y el código R correspondiente):

Ejemplo: Temas del paquete {ggthemes}

```
library(ggthemes)
siete <- p + theme_economist() + scale_fill_economist()
ocho <- p + theme_fivethirtyeight() + scale_fill_fivethirtyeight()
nueve <- p + theme_stata() + scale_fill_stata(scheme = "s2color")
diez <- p + theme_solarized() + scale_fill_solarized()

ggpubr::ggarrange(siete, ocho, nueve, diez)
```

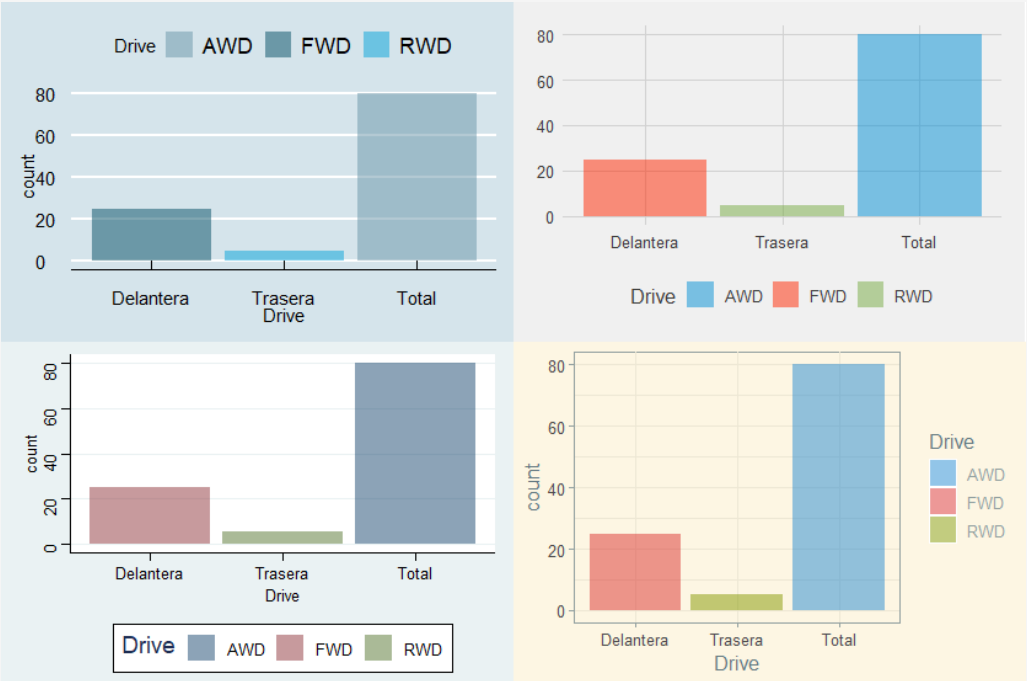


Figura 71: Temas del paquete {ggthemes}.

Además de {ggthemes}, hay otros paquetes que extienden las posibilidades, por ejemplo {hrbrthemes}, {bbplot}, entre otros. En este enlace, y en este otro, puedes encontrar una lista no exhaustiva, pero cubre la mayoría de los paquetes que brindan extensiones al paquete {ggplot2}.

6.8. Referencias bibliográficas

Añadir leyenda en R. (s.f.) R-coder. Recuperado el 30 de agosto de 2022 de: https://r-coder.com/leyenda-r/

Les différents types de points dans R: Comment utiliser pch? [Los diferentes tipos de puntos en R: ¿cómo utilizar pch?] (s.f.). STHDA: Statistical Tools For High-Throughput Data Analysis. Recuperado el 27 de agosto de 2022 de http://www.sthda.com/french/wiki/les-differents-types-de-points-dans-r-comment-utiliser-pch

Wickham, H. and Grolemund, G. (2019). R para Ciencia de Datos. https://es.r4ds.hadley.nz/

Wilkinson, L. (2005). *The Grammar of Graphics*. Springer New York, NY. https://doi.org/10.1007/0-387-28695-0

6.9. Cuaderno de ejercicios

Ejercicio 1

Para este ejercicio vamos a utilizar el conjunto de datos `paises`, contenido en el paquete {datos} (que ya trabajamos antes).

A partir de los datos y utilizando las funcionalidades gráficas de R base, debes crear un gráfico de dispersión con los siguientes parámetros:

1. la variable `anio` en el eje X y `esperanza_de_vida` en el eje Y.
2. los puntos coloreados según la variable `continente`.
3. los puntos más pequeños (tamaño de 0.7).
4. incluye un suavizado de los datos.

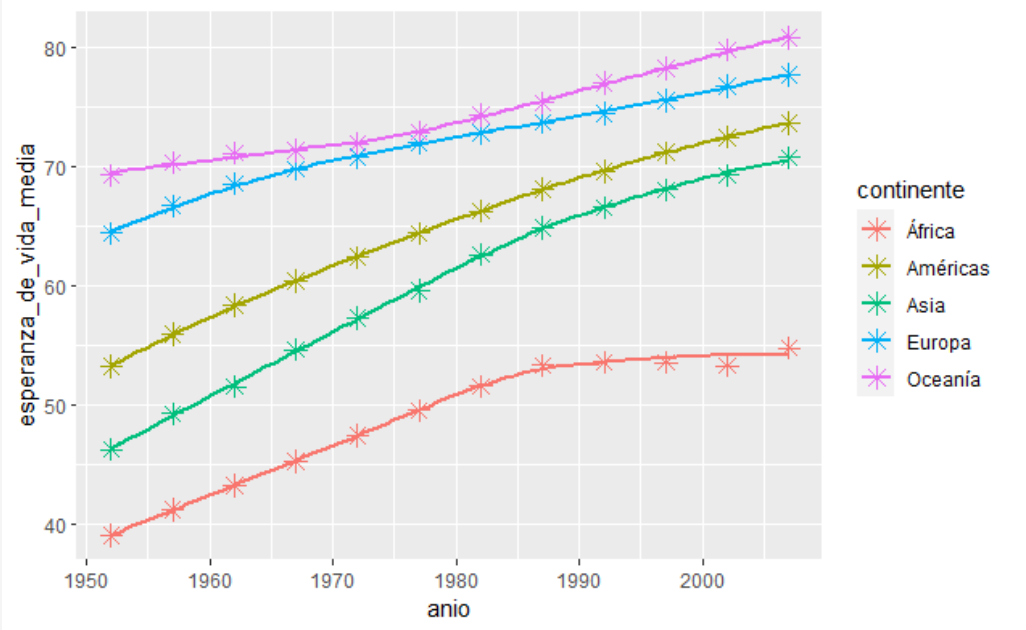
Ejercicio 2

Repite el ejercicio anterior, pero usando ahora el paquete `{ggplot2}`.

Ejercicio 3

Completa el siguiente código para obtener el gráfico que se muestra más abajo.

```
paises %>%
  group_by(_____________, anio) %>%
  summarize(esperanza_de_vida_media = mean(esperanza_de_vida)) %>%
  ggplot(aes(anio, _________________)) +
  geom_point(aes(color = continente), size = 3, shape = ____) +
  geom_smooth(aes(color = continente), ____= FALSE)
```



Ejercicio 4

Usa la base de datos `paises` para:

1. Crear un gráfico de la evolución de la esperanza **media** y **máxima** de vida en el mundo.
2. Colorea las líneas de color diferente

3. Establece diferentes tipos para cada línea.

Ejercicio 5

Para este ejercicio vamos a utilizar la base de datos `mpg` del paquete `{ggplot2}`. A partir de los datos:

1. Construir un gráfico de caja y bigotes para ver cómo cambia `hwy` frente a `class`.
2. Añade los puntos correspondientes a las observaciones y coloréalos por tipo `drv`.
3. Añade un punto para mostrar dónde está la media de `hmy` en una de las cajas.

Ejercicio 6

Usando el archivo de datos de `diamantes` del paquete `{datos}`, realiza histogramas del precio (`precio`) de los diamantes con las siguientes indicaciones:

1. Las barras deben ser de color violeta (el relleno).
2. El borde de las barras debe ser gris.
3. Los diamantes de diferente color deben aparecer en histogramas diferentes. Evalúa como se ve el gráfico con las diferentes opciones: `position: "identity", "stack", "dodge"`.
4. Cambia la intensidad del color
5. Cambia los nombres de los ejes al español.

En el siguiente vídeo podrás acceder a la resolución de los ejercicios 1 y 2 del cuaderno de ejercicios:


Accede al vídeo