Capítulo 17: Gráficos en R

La visualización de gráficas es una de las herramientas más poderosas y usadas de R. A continuación utilizaremos la función `plot()` que es una de las herramientas principales para hacerlo.

Preparar los datos

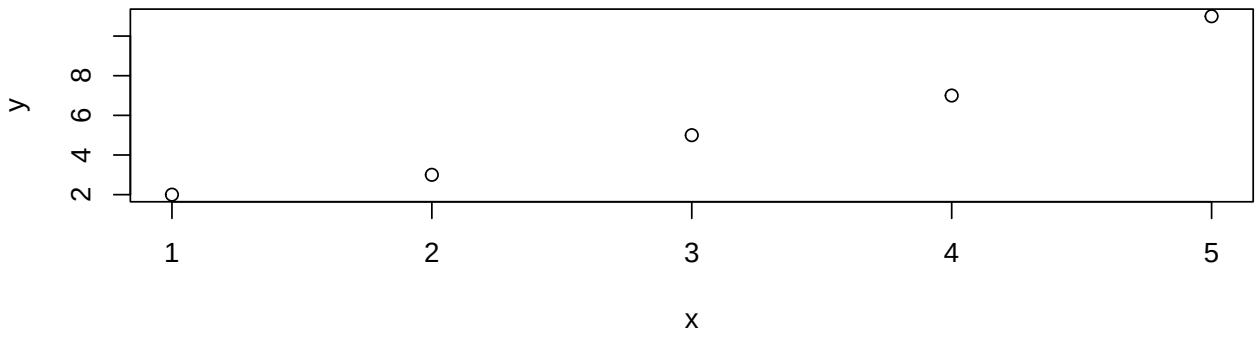
Uno de los primeros pasos para la graficación en R es tener qué graficar. Para ello se debe preparar primero los datos, estos pueden ser vectores, arreglos, data frames, etc. Ejemplo:

```r
# Ejemplo de datos
x <- c(1, 2, 3, 4, 5)
y <- c(2, 3, 5, 7, 11)
```

Obtener el gráfico

La función `plot()` es la forma más sencilla de crear un gráfico en R . Aquí hay algunos ejemplos básicos:

```r
#Gráfico de dispersión
plot(x, y)
```

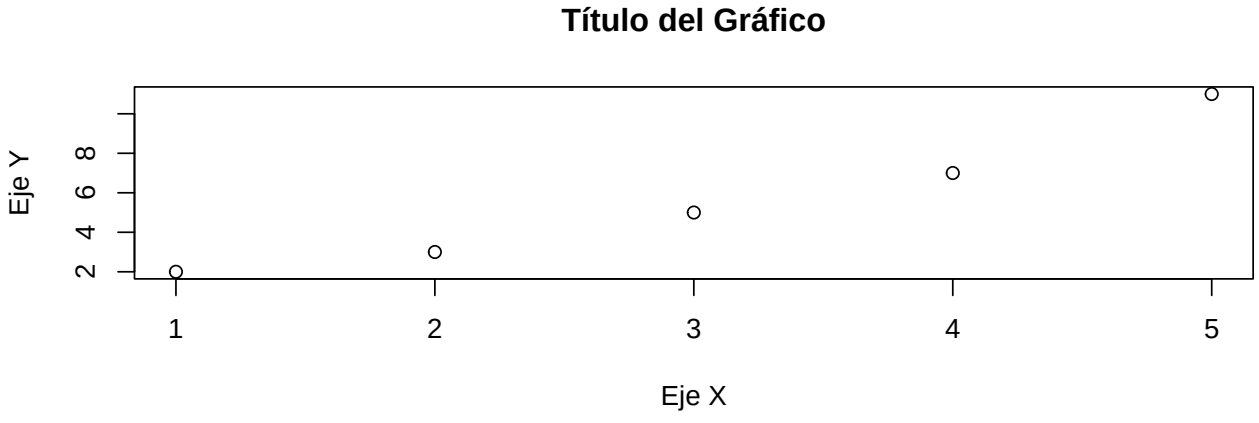


Este comando creará un gráfico de dispersión con x en el eje horizontal e y en el eje vertical.

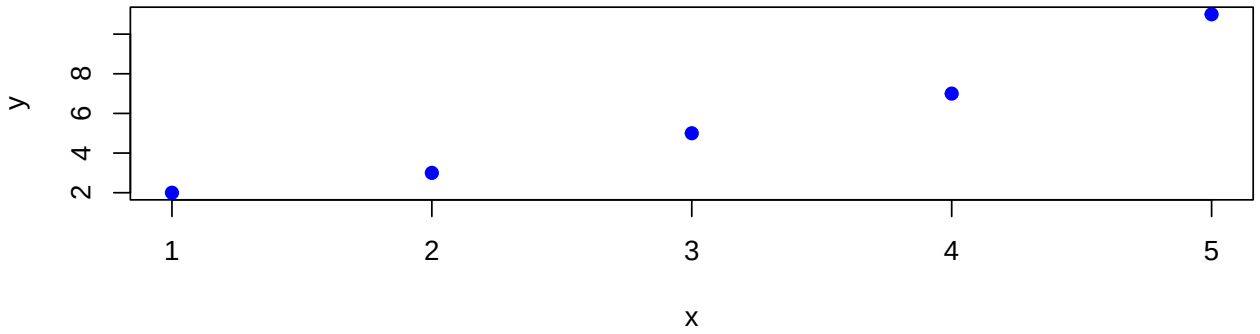
Personalizar el gráfico

Se va a poder personalizar el gráfico agregando títulos, etiquetas y cambiando el tipo de puntos o líneas.

```r
#Títulos y etiquetas
plot(x, y,
     main = "Título del Gráfico",  # Título del gráfico
     xlab = "Eje X",               # Etiqueta del eje X
     ylab = "Eje Y")               # Etiqueta del eje Y
```



```r
#Cambiar tipo de puntos y color
plot(x, y,
     pch = 19,                    # Tipo de punto (19 es un círculo sólido)
     col = "blue")                # Color de los puntos
```



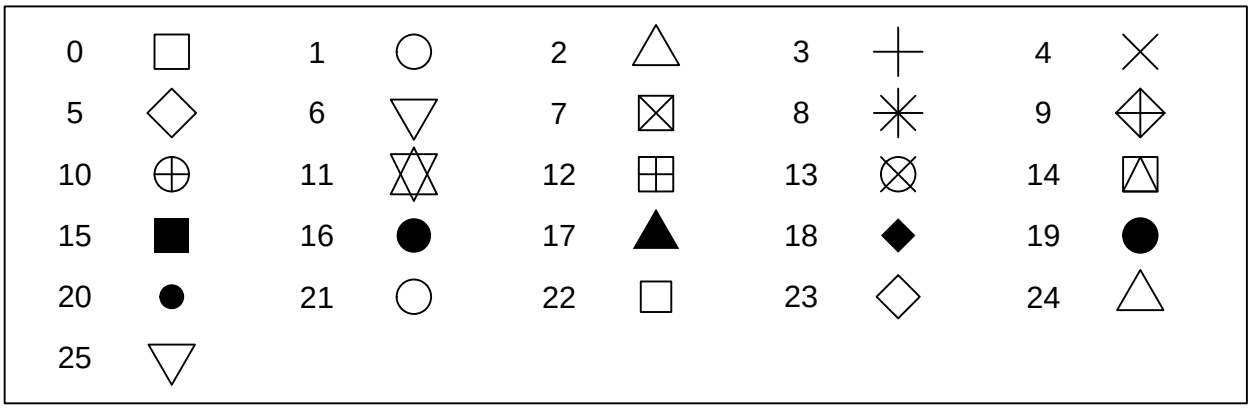
Nótese como pch tiene el número 19, sin embargo, se puede usar 25 diferentes pch que vienen por default en este lenguaje. Véase la siguiente gráfica:

Tipos de pch

```r
grid <- expand.grid(1:5, 6:1)
plot(grid, pch = 0:30, cex = 2.5,yaxt = "n", xaxt = "n",ann = FALSE,
     xlim = c(0.5, 5.25),
     ylim = c(0.5, 6.5))
```

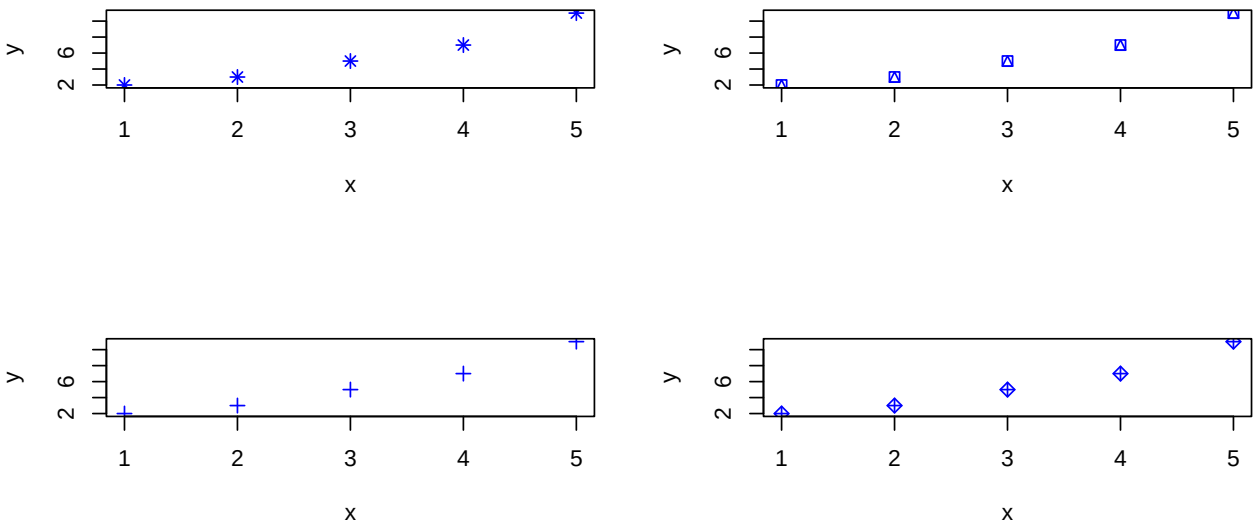
```
## Warning in plot.xy(xy, type, ...): unimplemented pch value '26'

## Warning in plot.xy(xy, type, ...): unimplemented pch value '27'

## Warning in plot.xy(xy, type, ...): unimplemented pch value '28'

## Warning in plot.xy(xy, type, ...): unimplemented pch value '29'
```

```r
grid2 <- expand.grid(seq(0.6, 4.6, 1), 6:1)
text(grid2$Var1[1:26], grid2$Var2[1:26], 0:25)
```



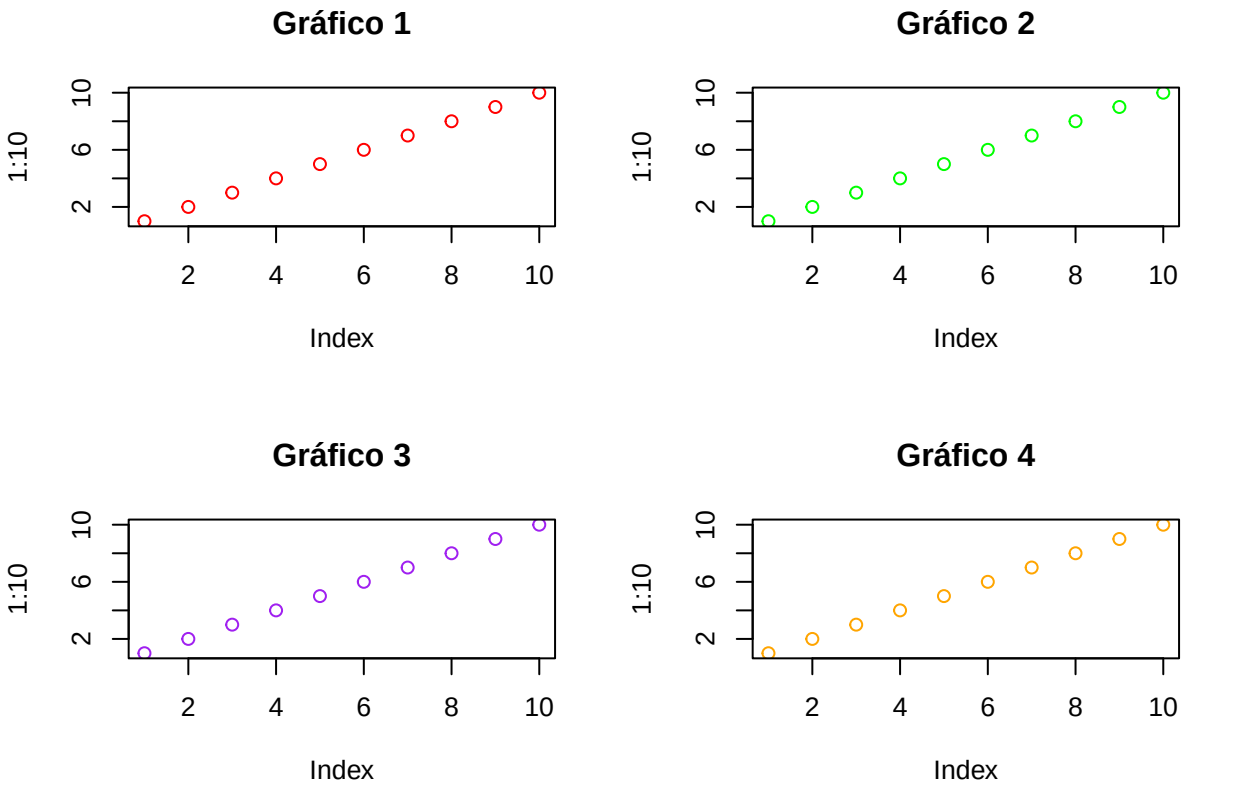
Ahora repitamos dos veces el código anterior a este gráfico, pero con un pch distinto:

```r
par(mfrow=c(2, 2))
plot(x, y, pch = 8, col = "blue")
plot(x, y, pch = 14, col = "blue")
plot(x, y, pch = 3, col = "blue")
plot(x, y, pch = 9, col = "blue")
```



Nótese como aparecen diferentes diseños según el gráfico anterior. También se usó la función `par()`, con esta se va a poder dividir la ventana gráfica en varias partes para mostrar múltiples gráficos al mismo tiempo usando el parámetro mfrow o mfcol. Ejemplo:

```r
# Dividir la ventana gráfica en 2 filas y 2 columnas
par(mfrow = c(2, 2))
plot(1:10, main = "Gráfico 1", col = "red")
plot(1:10, main = "Gráfico 2", col = "green")
plot(1:10, main = "Gráfico 3", col = "purple")
plot(1:10, main = "Gráfico 4", col = "orange")
```



Colores

También algo útil son los colores. En R hay 657 colores predefinidos que puedes utilizar directamente en gráficos o visualizaciones. Estos colores tienen nombres que pueden ser usados como cadenas de texto (por ejemplo, “red”, “blue”, “lightblue”, “darkgreen”, etc.). Para consultar eso se puede usar la función `color()`:

```
length(colors()) #Total de colores que hay.
```

```
## [1] 657
```

```
head(colors(), 10)
```

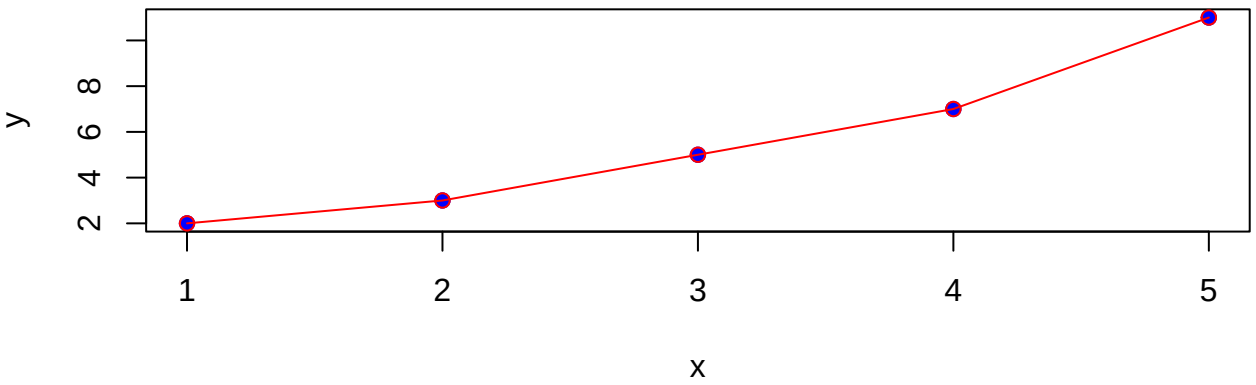
```
##  [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"
##  [5] "antiquewhite2"  "antiquewhite3"  "antiquewhite4"  "aquamarine"
##  [9] "aquamarine1"    "aquamarine2"
```

Aquí nada más se muestra los primeros 10 pero se pueden ver todos los colores si se usa la función sola.

Más elementos

Puedes agregar más elementos al gráfico existente usando funciones como `lines()`, `points()`, `abline()`, etc.

```
## Añadir una línea
# Crear el gráfico base
plot(x, y, pch = 19, col = "blue")

# Añadir una línea
lines(x, y, type = "o", col = "red")  # "o" indica que se dibujen líneas y puntos
```



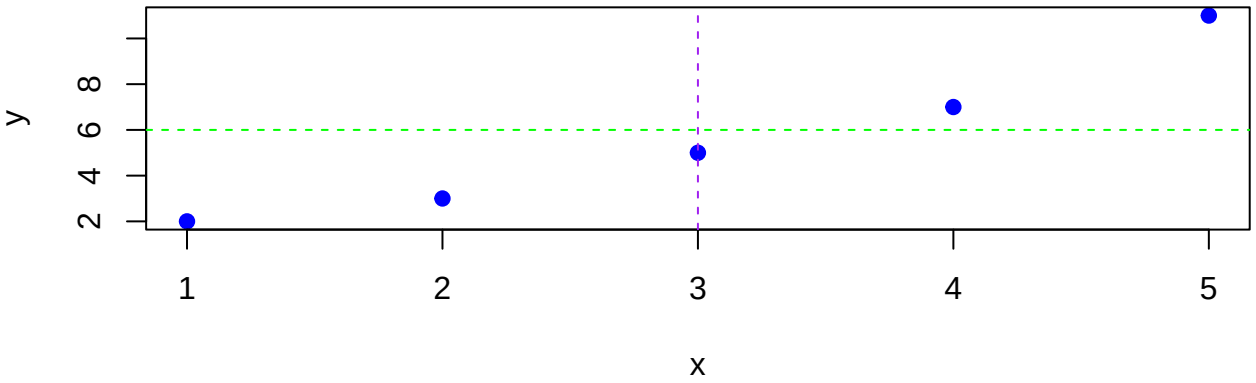
Note como la parte de lines tiene un parámetro `type`, este parámetro ofrece diferentes opciones:

Type

Añadir una línea de referencia

```
# Crear el gráfico base
plot(x, y, pch = 19, col = "blue")

# Añadir una línea horizontal en y = 6
abline(h = 6, col = "green", lty = 2)  # lty = 2 es una línea discontinua

# Añadir una línea vertical en x = 3
abline(v = 3, col = "purple", lty = 2)
```

"p"	Puntos: Solo dibuja los puntos del gráfico (por defecto).
"l"	Líneas: Conecta los puntos con líneas.
"b"	Puntos y líneas: Dibuja los puntos y los conecta con líneas.
"c"	Líneas discontinuas: Similar a "b", pero no dibuja los puntos, solo las líneas.
"o"	Puntos y líneas superpuestos: Dibuja puntos y líneas, pero los puntos están sobre las líneas.
"h"	Líneas verticales: Dibuja líneas verticales desde los puntos hasta el eje X (tipo histograma).
"s"	Escalones: Crea líneas en forma de escalones (horizontal primero, luego vertical).
"S"	Escalones inversos: Similar a "s", pero comienza con la línea vertical y luego horizontal.
"n"	Ninguno: No dibuja nada en el gráfico (útil para agregar elementos personalizados después).



Nótese el parámetro `lty`, se utiliza para definir el tipo de línea en gráficos. Permite personalizar cómo se dibujan las líneas en términos de estilo, como líneas continuas, punteadas, o combinaciones de ambas.

lty

1. Números del 1 al 6:

   Cada número representa un estilo de línea predeterminado:

   1: Línea sólida (por defecto).
   2: Línea punteada larga (– –).
   3: Línea punteada corta (· · ·).
   4: Línea discontinua y punteada (- ·).
   5: Línea discontinua y punteada larga (- · · ·).
   6: Línea discontinua y punteada corta (- ..).

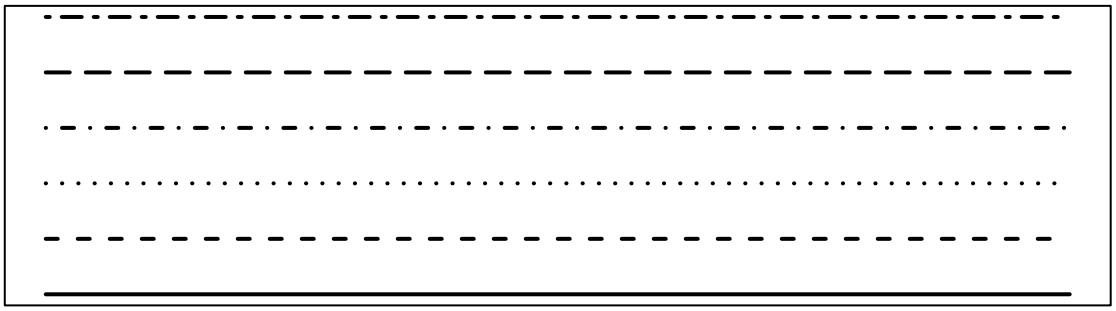
2. Cadenas de texto:

   Puedes usar una secuencia personalizada de guiones y espacios mediante una cadena de números. Esto define el patrón de la línea:

   "4 2": Dibuja un guion de longitud 4 seguido de un espacio de longitud 2.
   "1 1": Dibuja una línea punteada fina.

```r
plot(1:6, type = "n", main = "Estilos de línea en R", xlab = "", ylab = "", xaxt = "n", yaxt = "n")
for (i in 1:6) {
  lines(c(1, 6), c(i, i), lty = i, lwd = 2)
  text(6.5, i, labels = paste("lty =", i), pos = 4)
}
```

Estilos de línea en R



Guardar el gráfico

Puedes guardar el gráfico en diferentes formatos (PNG, PDF, etc.) usando funciones como `png()`, `pdf()`, `jpeg()`, etc.

```
# Abrir el dispositivo gráfico
png("mi_grafico.png")

# Crear el gráfico
plot(x, y,
     pch = 19,
     col = "blue",
     main = "Gráfico de Ejemplo",
     xlab = "Eje X",
     ylab = "Eje Y")

# Cerrar el dispositivo gráfico
dev.off()
```

```
## pdf
##   2
```

Ejemplo completo:

```
# Datos
x <- c(1, 2, 3, 4, 5)
y <- c(2, 3, 5, 7, 11)

# Abrir el dispositivo gráfico
png("mi_grafico_completo.png")

# Crear el gráfico
plot(x, y, main = "Gráfico Completo", xlab = "Eje X", ylab = "Eje Y", pch = 19, col = "blue")

# Añadir una línea de puntos y líneas
lines(x, y, type = "o", col = "red")

# Añadir líneas de referencia
abline(h = 6, col = "green", lty = 2)
abline(v = 3, col = "purple", lty = 2)

# Cerrar el dispositivo gráfico
dev.off()
```

```
## pdf
##   2
```

Gráficos específicos

Este tipo de gráfica que acabamos de ver no es la única forma de graficar, R ofrece gráficos más especificos dependiendo lo que se ocupe y los tipos de datos que se tengan:

Gráficas básicas:

1. plot(). Gráfico general para dispersión, líneas, etc.
2. barplot(). Gráficos de barras.
3. hist(). Diagramas de caja y brazos.
4. pie(). Histogramas.
5. lines(). Gráficos de pastel (cuidado: no se recomiendan mucho en análisis serios).
6. points(). Añadir líneas a un gráfico existente.
7. matplot(). Gráficos de líneas o puntos para múltiples columnas.
8. stripchart(). Gráficos de dispersión simples.
9. dotchart(). Gráficos de puntos.

Visualización de distribuciones:

1. density(). Estimación de densidad de un conjunto de datos.
2. curve(). Graficar funciones matemáticas.
3. qqnorm(). Gráficos Q-Q para verificar la normalidad de los datos.
4. qqline(). Línea de referencia en gráficos Q-Q.
5. rug(). Añade marcas a lo largo del eje para mostrar datos individuales.

Gráficos de series de tiempo:

1. ts.plot(). Gráficos de series temporales.
2. plot.ts(). Gráficos para objetos de series temporales (ts).

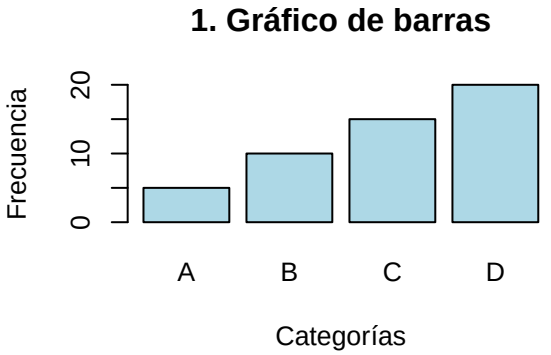
Gráficas tridimensionales:

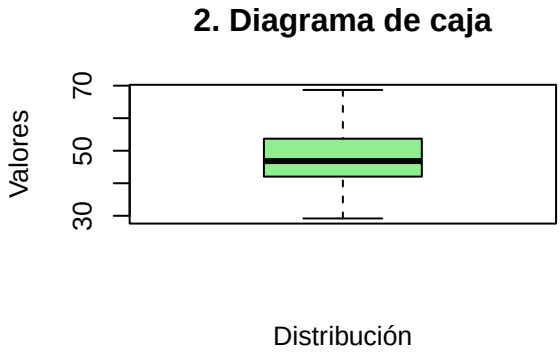
1. persp(). Superficies tridimensionales.
2. contour(). Gráficos de contornos.
3. image(). Mapas de calor.

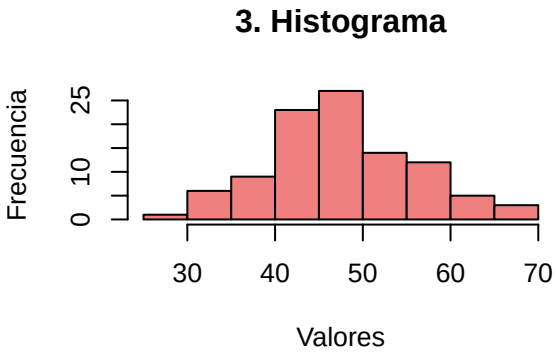
Cada uno de estos gráficos cuenta con sus parámetros, por lo que para propósito de este curso solo se van a explicar algunos que se usen en los siguientes ejemplos:

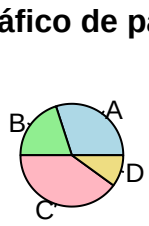
Ejemplos:

```r
# Establecer los gráficos en una cuadrícula 2x2
par(mfrow = c(2,2)) # 2 filas y 2 columnas
# 1. Gráfico de barras (barplot)
counts <- c(5, 10, 15, 20)
barplot( counts, main = "1. Gráfico de barras", xlab = "Categorías",
         ylab = "Frecuencia", names.arg = c("A", "B", "C", "D"),
         col = "lightblue")
# 2. Diagrama de caja (boxplot)
data <- rnorm(100, mean = 50, sd = 10)
boxplot( data, main = "2. Diagrama de caja", xlab = "Distribución",
         ylab = "Valores", col = "lightgreen")

# 3. Histograma (hist)
hist( data, main = "3. Histograma", xlab = "Valores", ylab = "Frecuencia",
      col = "lightcoral", border = "black" )

# 4. Gráfico circular (pie)
values <- c(30, 20, 40, 10)
pie( values, main = "4. Gráfico de pastel", labels = c("A", "B", "C", "D"),
  col = c("lightblue", "lightgreen", "lightpink", "lightgoldenrod") )
```





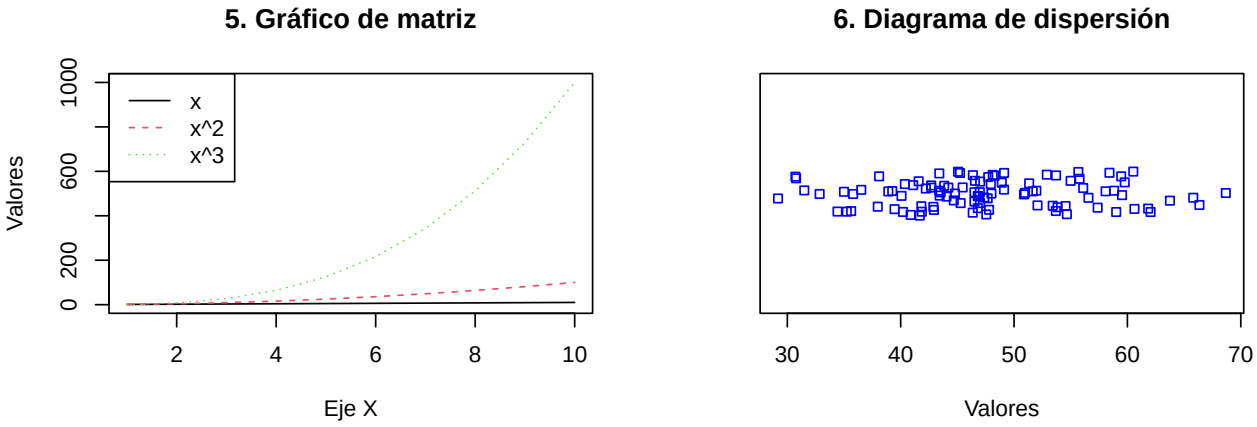


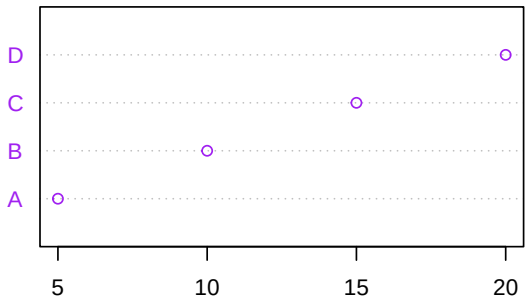


Parámetros a tomar en cuenta:

- names.arg es un vector de nombres que se trazarán debajo 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.
- col se usa para definir los colores de las barras o de la caja según sea el caso.
- border es el color del borde alrededor de las barras.Por defecto se utiliza el color de primer plano estándar.
- labels es el nombre de las "slices".

```r
# Establecer los gráficos en una cuadrícula 2x2
par(mfrow = c(2,2)) # 2 filas y 2 columnas

# 5. Gráfico de matriz (matplot)
x <- 1:10
y <- cbind(x, x^2, x^3)
matplot(x, y,type = "l",main = "5. Gráfico de matriz",xlab = "Eje X",
        ylab = "Valores", col = 1:3, lty = 1:3)
legend("topleft", legend = c("x", "x^2", "x^3"), col = 1:3, lty = 1:3)

# 6. Diagrama de dispersión simple (stripchart)
stripchart( data, main = "6. Diagrama de dispersión", xlab = "Valores",
            col = "blue", method = "jitter")


# 7. Gráfico de puntos (dotchart)
dotchart(counts, main = "7. Gráfico de puntos", labels = c("A", "B", "C", "D"),
         col = "purple")

# Ajustar espaciado y restablecer layout
par(mfrow = c(1, 1)) # Volver a un gráfico por ventana
```





Parámetros a tomar en cuenta:

- type similar al caso de plot().
- lty similar al caso de plot().
- legend() se utiliza para agregar una leyenda a un gráfico. La leyenda proporciona una referencia visual para identificar las características de los elementos del gráfico, como líneas, puntos, colores, etc.

- method se utiliza para especificar cómo se distribuyen los puntos en el gráfico. Este parámetro cuenta con 3 diferentes opciones:
  - "stack": Los puntos se apilan si tienen el mismo valor, es decir, se colocan uno sobre otro.
  - "jitter": Se dispersan los puntos de forma aleatoria alrededor de la posición original para evitar que se amontonen.
  - "overplot": Los puntos se dibujan en las mismas posiciones si tienen el mismo valor, sin ningún ajuste.

Capítulo 18: Tidyverse

Figura 1: Principales paquetes de Tidyverse

Fuente: Tidyverse para Data Análisis. (2021)

Tidyverse es una colección de paquetes disponibles en R y orientados a la manipulación, importación, exploración y visualización de datos. El uso de Tidyverse permite facilitar el trabajo estadístico y la generación de trabajos reproducibles. Está compuesto de los siguientes paquetes:

- readr. Para importar datos desde archivos (CSV, TSV, etc.).
- dplyr. Para manipulación de datos, como seleccionar, filtrar, agrupar y resumir datos.
- ggplot2. Para crear gráficos.
- tibble. Una versión mejorada de los data frames.
- tidyr. Para limpiar y organizar datos en formatos ordenados (tidy data).
- purr. Para trabajar con programación funcional (listas, mapeo de funciones).
- stringr. Para manipulación de cadenas de texto.
- forcats. Para manipular factores.

Para instalar y cargar el `Tidyverse`, puedes usar:

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Para propósitos de este curso nos centraremos en 3 librerías del `Tidyverse`:

dplyr

El paquete dplyr entrega una serie de funciones muy útiles para la manipulación de data frames, permitiendo simplificar las acciones sobre este tipo de datos.

Ejemplos para dplyr:

1. Selección de columnas

```
data <- data.frame( Nombre = c("Ana", "Carlos", "Juan", "Lucía"),
                    Edad = c(23, 30, 25, 28),
                    Ciudad = c("Madrid", "Barcelona", "Sevilla", "Valencia"))

# Seleccionar solo las columnas "Nombre" y "Edad"
resultado <- data %>% select(Nombre, Edad)
resultado
```

```
##   Nombre Edad
## 1    Ana   23
## 2 Carlos   30
## 3   Juan   25
## 4  Lucía   28
```

A priori se nota la diferencia a la hora de seleccionar columnas, pues con esta librería se uso `%>%` y el comando `select()`en lugar de las maneras que hemos visto antes:

```
data[ , c(1,2)]
```

```
##   Nombre Edad
## 1    Ana   23
## 2 Carlos   30
## 3   Juan   25
## 4  Lucía   28
```

Depende de cada uno como le resulte más fácil.

2. Filtrar filas

```
adultos <- data %>% filter(Edad > 25)
print(adultos)
```

```
##   Nombre Edad    Ciudad
## 1 Carlos   30 Barcelona
## 2  Lucía   28  Valencia
```

Nótese el uso de `filter()`.

3. Crear una nueva columna Añadir una columna con la edad en meses:

```
data <- data %>% mutate(EdadMeses = Edad * 12)
print(data)
```

```
##   Nombre Edad    Ciudad EdadMeses
## 1    Ana   23    Madrid       276
## 2 Carlos   30 Barcelona       360
## 3   Juan   25   Sevilla       300
## 4  Lucía   28  Valencia       336
```

Nótese el comando `mutate()`

4. Resumir y agrupar datos

```
# Crear un data frame con grupos
ventas <- data.frame(
  Vendedor = c("Ana", "Carlos", "Ana", "Carlos", "Juan"),
  Ventas = c(200, 150, 300, 250, 100)
)

# Resumir ventas totales por vendedor
resumen <- ventas %>%
  group_by(Vendedor) %>%
  summarise(VentasTotales = sum(Ventas))
print(resumen)
```

```
## # A tibble: 3 x 2
##   Vendedor VentasTotales
##   <chr>            <dbl>
## 1 Ana                500
## 2 Carlos             400
## 3 Juan               100
```

Nótese los comandos de `summarise()` y `group_by()`.

5. Ordenar datos

```
# Ordenar por la columna Ventas de manera descendente
ordenado <- ventas %>% arrange(desc(Ventas))
print(ordenado)
```

```
##   Vendedor Ventas
## 1      Ana    300
## 2   Carlos    250
## 3      Ana    200
## 4   Carlos    150
## 5     Juan    100
```

Nótese el uso de `arrange()`.

Otro ejemplo con `mtcars`:

```
mtcars_clean <- mtcars %>%
  select(mpg, cyl, hp, wt) %>%  # Seleccionar columnas específicas
  filter(cyl == 6) %>%          # Filtrar solo autos con 6 cilindros
  arrange(desc(mpg))            # Ordenar por millas por galón (mpg), de mayor a menor

print(mtcars_clean)
```

```
##                 mpg cyl  hp    wt
## Hornet 4 Drive 21.4   6 110 3.215
## Mazda RX4      21.0   6 110 2.620
## Mazda RX4 Wag  21.0   6 110 2.875
## Ferrari Dino   19.7   6 175 2.770
## Merc 280       19.2   6 123 3.440
## Valiant        18.1   6 105 3.460
## Merc 280C      17.8   6 123 3.440
```

Como vemos el uso de dplyr es principalmente para este tipo de estructura de datos, es cuestión de cada uno elegir si usar o no la librería dependiendo de lo que se quiera hacer. Para ver más usos de esta librería se recomienda consultar el siguiente link: https://rpubs.com/paraneda/tidyverse.

ggplot2

Esta librería, creada por Hadley Wickham en 2005, cuenta con un amplio uso en la visualización de datos. A continuación se presentarán ejemplo del tipo de gráficos que se pueden hacer con esta librería.

Gráfico de Barras

```r
# Datos de ejemplo
categories <- c("Cat1", "Cat2", "Cat3", "Cat4")
values <- c(23, 17, 35, 50)
df <- tibble(category = categories, value = values)

# Crear gráfico de barras
ggplot(df, aes(x = category, y = value)) +
  geom_bar(stat = "identity", fill = "dodgerblue", color = "darkblue") +
  labs(title = "Gráfico de Barras", x = "Categorías", y = "Valores")
```

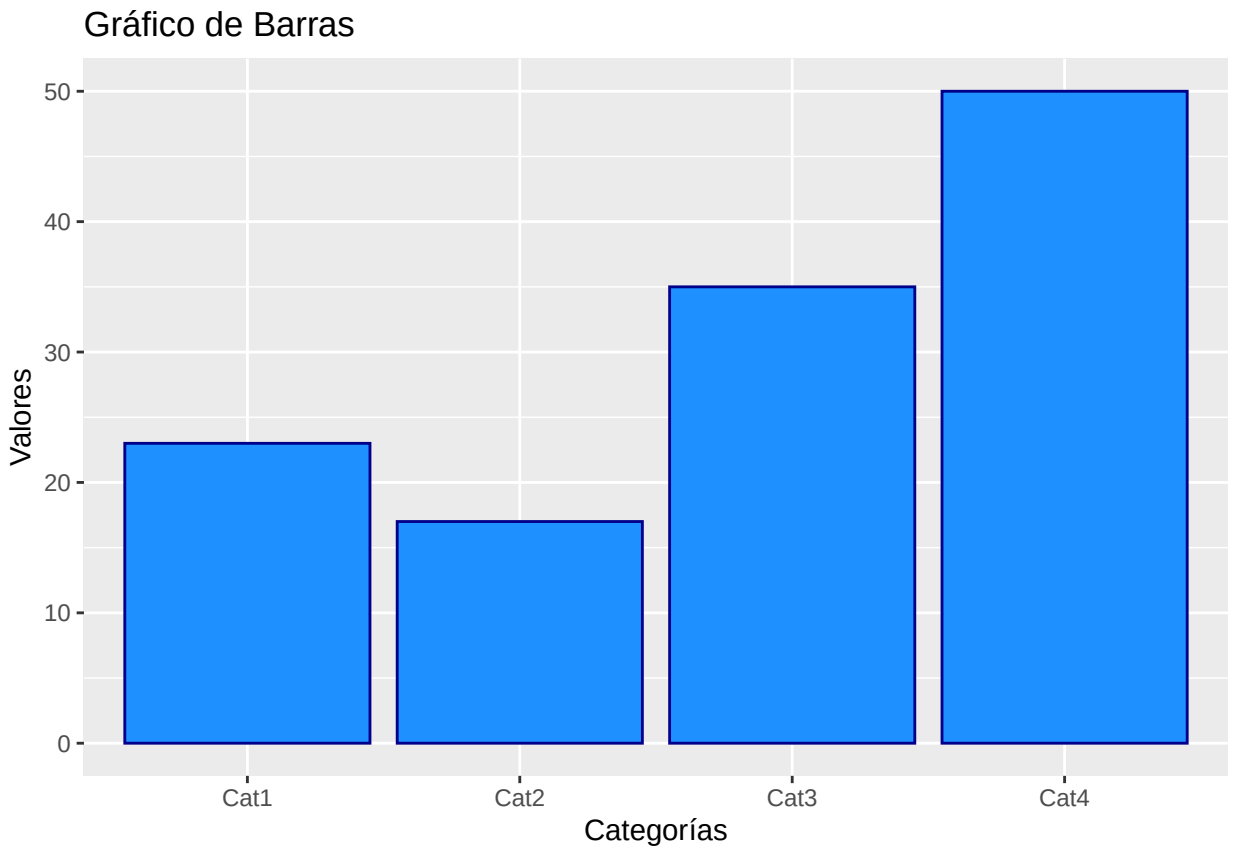
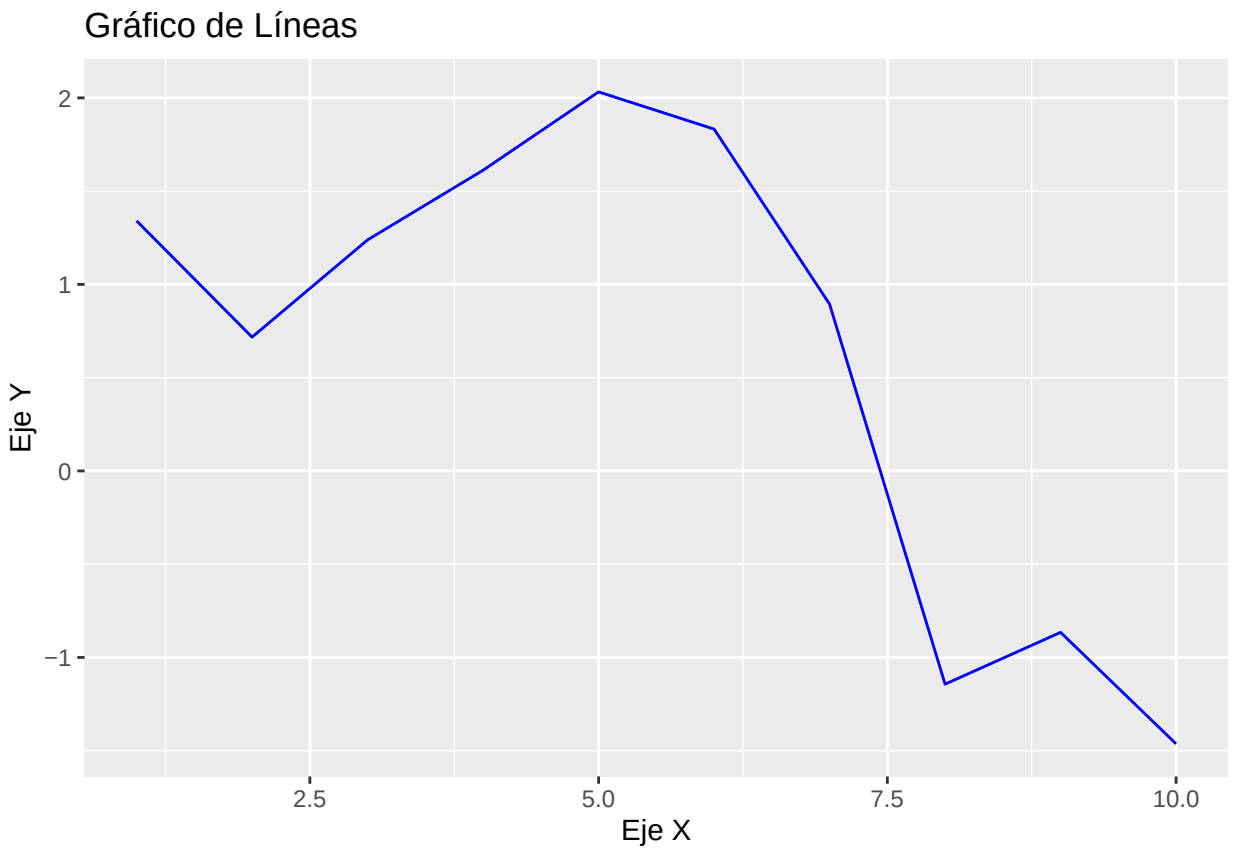


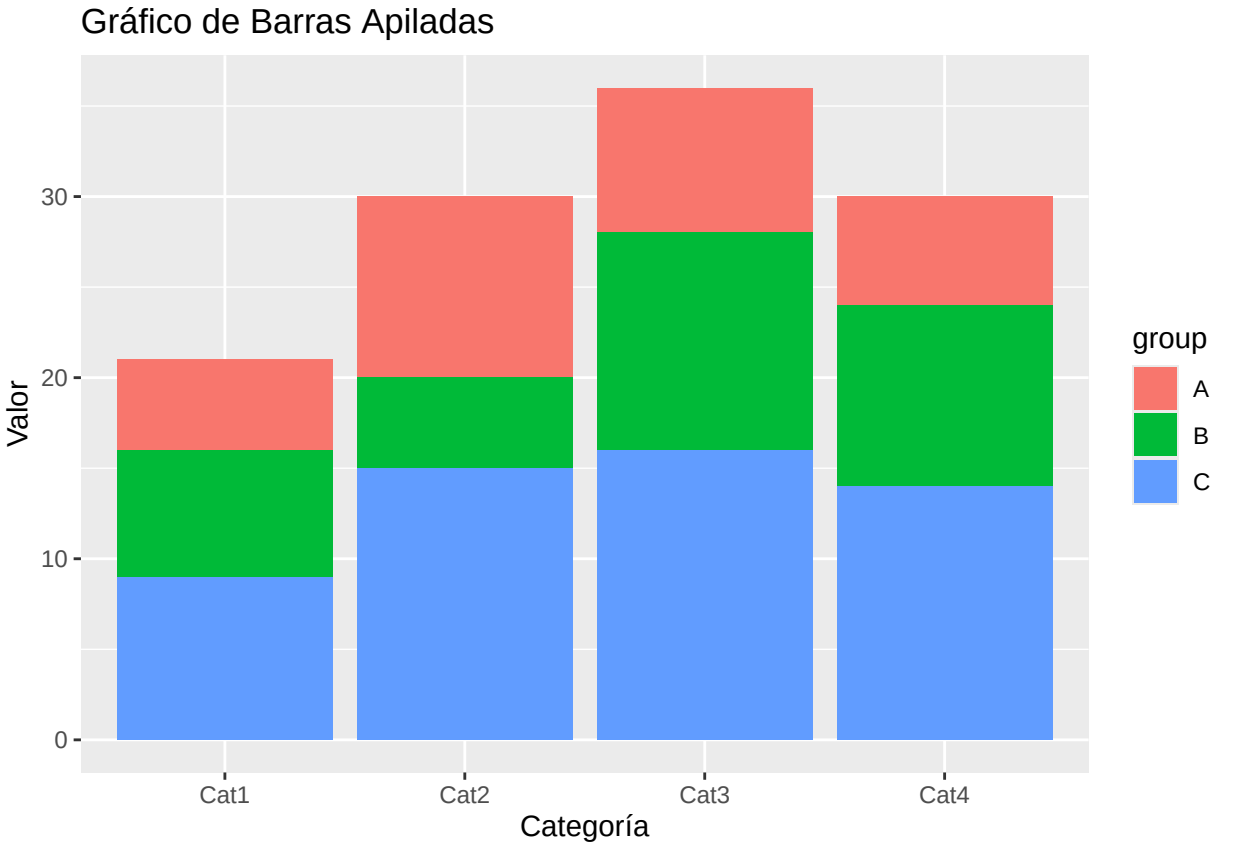
Gráfico de Línea

```r
# Datos de ejemplo
df <- tibble(x = 1:10, y = cumsum(rnorm(10)))

# Crear gráfico de líneas
ggplot(df, aes(x = x, y = y)) + geom_line(color = "blue") +
  labs(title = "Gráfico de Líneas", x = "Eje X", y = "Eje Y")
```



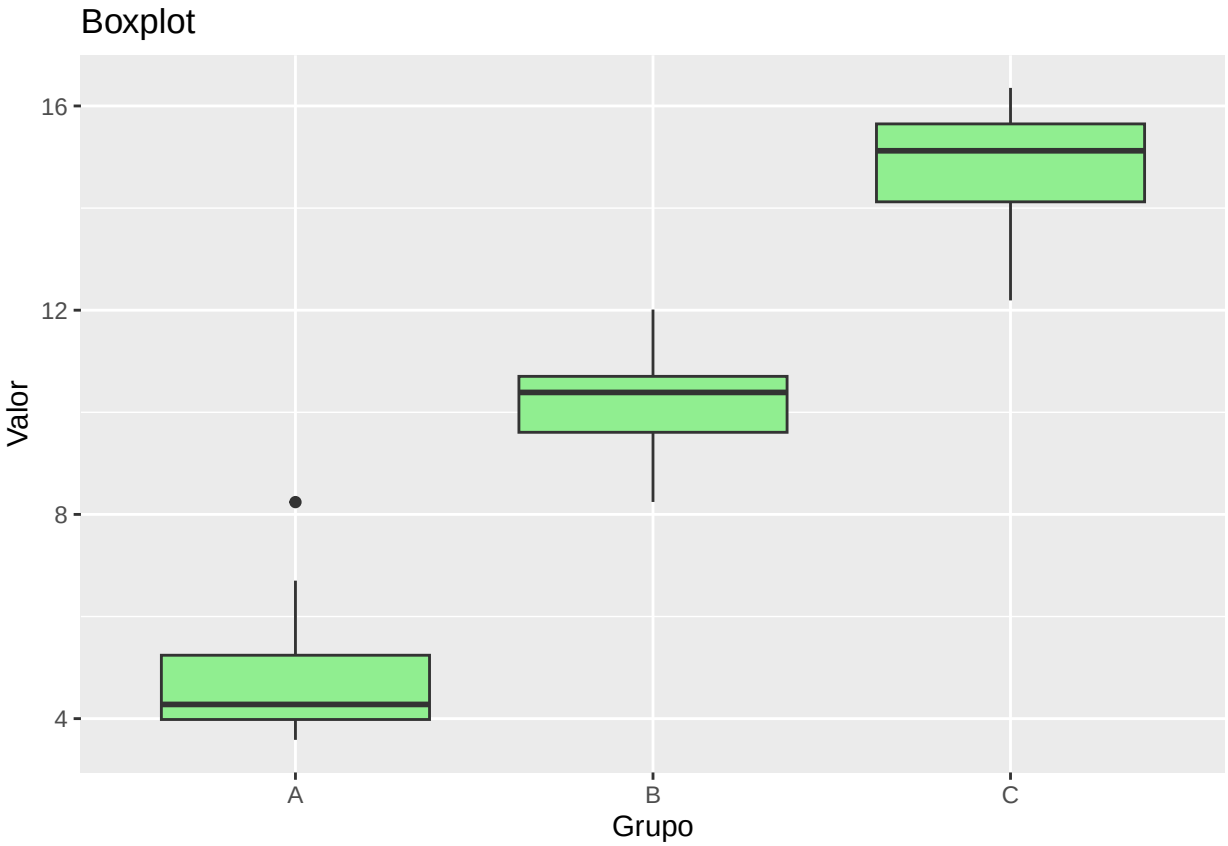
Barras ampliadas

```
# Datos de ejemplo
df <- tibble(group = rep(c("A", "B", "C"), times = 4),
             category = rep(c("Cat1", "Cat2", "Cat3", "Cat4"), each = 3),
             value = c(5, 7, 9, 10, 5, 15, 8, 12, 16, 6, 10, 14))

# Crear gráfico de barras apiladas
ggplot(df, aes(x = category, y = value, fill = group)) +
  geom_bar(stat = "identity") +
  labs(title = "Gráfico de Barras Apiladas", x = "Categoría", y = "Valor")
```



Boxplot

```r
# Datos de ejemplo
df <- tibble(group = rep(c("A", "B", "C"), each = 20),
             value = c(rnorm(20, mean = 5), rnorm(20, mean = 10), rnorm(20, mean = 15)))

# Crear boxplot
ggplot(df, aes(x = group, y = value)) +
  geom_boxplot(fill = "lightgreen") +
  labs(title = "Boxplot",
       x = "Grupo",
       y = "Valor")
```



En principio, vemos que la gran diferencia radica en como se van añadiendo elementos a los gráficos, también en el fondo y las funciones que utiliza se suele ver distinto. En un principio se podría decir que usar plot es más para tareas rápidas y gráficos simples. Mientras que usar ggplot2 es para gráficos más complejos, personalizables y escalables.

Para más información véase los siguientes links:

- https://r-coder.com/graficos-r/
- https://ggplot2.tidyverse.org/

purrr

La librería purrr es ideal para programación funcional, especialmente cuando necesitas trabajar con listas o aplicar funciones repetitivamente. A continuación unos ejemplos de su uso:

1. Aplicar una función a cada elemento de una lista (map())

```
library(purrr)

# Crear una lista de números
numeros <- list(1, 2, 3, 4)

numeros
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
```

```
# Calcular el cuadrado de cada número
cuadrados <- map(numeros, ~ .x^2)
cuadrados
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 16
```

2. Aplicar una función y devolver un vector simplificado (map_dbl())

```
# Crear una lista de números
numeros <- list(1, 2, 3, 4)

# Calcular el doble y devolver un vector numérico
dobles <- map_dbl(numeros, ~ .x * 2)
dobles
```

```
## [1] 2 4 6 8
```

3. Filtrar listas (keep() y discard())

```r
# Crear una lista de números
numeros <- list(1, 2, 3, 4, 5, 6)

# Mantener solo los números pares
pares <- keep(numeros, ~ .x %% 2 == 0)
pares
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 6
```

```r
# Descartar los números pares
impares <- discard(numeros, ~ .x %% 2 == 0)
impares
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 5
```

4. Crear combinaciones de listas (cross2() y map2())

```r
# Crear dos listas
letras <- list("A", "B", "C")
numeros <- list(1, 2, 3)

# Crear todas las combinaciones entre letras y números
combinaciones <- cross2(letras, numeros)
```

```
## Warning: `cross2()` was deprecated in purrr 1.0.0.
## i Please use `tidyr::expand_grid()` instead.
## i See <https://github.com/tidyverse/purrr/issues/768>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
combinaciones
```

```
## [[1]]
## [[1]][[1]]
## [1] "A"
##
## [[1]][[2]]
## [1] 1
##
##
## [[2]]
```

```
## [[2]][[1]]
## [1] "B"
##
## [[2]][[2]]
## [1] 1
##
##
## [[3]]
## [[3]][[1]]
## [1] "C"
##
## [[3]][[2]]
## [1] 1
##
##
## [[4]]
## [[4]][[1]]
## [1] "A"
##
## [[4]][[2]]
## [1] 2
##
##
## [[5]]
## [[5]][[1]]
## [1] "B"
##
## [[5]][[2]]
## [1] 2
##
##
## [[6]]
## [[6]][[1]]
## [1] "C"
##
## [[6]][[2]]
## [1] 2
##
##
## [[7]]
## [[7]][[1]]
## [1] "A"
##
## [[7]][[2]]
## [1] 3
##
##
## [[8]]
## [[8]][[1]]
## [1] "B"
##
## [[8]][[2]]
## [1] 3
##
##
## [[9]]
## [[9]][[1]]
## [1] "C"
```

```
##
## [[9]][[2]]
## [1] 3
```

```
# Aplicar una función a las combinaciones
resultado <- map2(letras, numeros, ~ paste(.x, .y, sep = "-"))
resultado
```

```
## [[1]]
## [1] "A-1"
##
## [[2]]
## [1] "B-2"
##
## [[3]]
## [1] "C-3"
```

5. Iterar condicionalmente (map_if())

```
# Lista con números positivos y negativos
numeros <- list(-1, 2, -3, 4)

# Convertir en positivo solo los números negativos
positivos <- map_if(numeros, ~ .x < 0, ~ abs(.x))
positivos
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
```

Capítulo 19: Bibliografía del curso.

Casanova, S. & Barrios, E. (2021) Introducción_a_R. Recuperado de:https://gente.itam.mx/ebarrios/cursoR/Agosto2022/Introduccion_a_R.pdf

Barrios, E.(2010). R: Un lenguaje para análisis de datos y graficación. Recuperado de:https://gente.itam.mx/ebarrios/docs/porqueR.pdf

Hernández, F. & Usuga, O.(2024) Manual de R. Recuperado de: https://fhernanb.github.io/Manual-de-R/

Martín-González, F. (2024). Gráficos en R. Recuperado de https://www4.ujaen.es/~fmartin/R/graficos.html

R Core Team. (s.f.). readr: Read rectangular text data. Comprehensive R Archive Network (CRAN). Recuperado de : https://cran.r-project.org/web/packages/readr/index.html

Araneda. (2021) Tidyverse para Data Análisis. Recuperado de: https://rpubs.com/paraneda/tidyverse.

Wickham e.t. al. (s.f.) ggplot2. Recuperado de: https://ggplot2.tidyverse.org/index.html

R Coder. (2024) Gráficos en R. Recuperado de: https://r-coder.com/graficos-r/