Introducción - Software R

Seminario Permanente de Formación en Inteligencia Artificial Aplicada a la Defensa

Francisco Charte Ojeda - fcharte.com Universidad de Jaén - DaSCI

21 abril 2020

Contents

Introducción a R

¿Qué es R?

R es una herramienta y al mismo tiempo un lenguaje:

- **Herramienta** - Programa software en el que introduciremos las órdenes para analizar los datos y generar los gráficos
- **Lenguaje** - El conjunto de órdenes que entiende el software forman el lenguaje R

¿Qué es RStudio?

Una herramienta más cómoda a la hora de trabajar con el lenguaje R. Respecto al entorno de trabajo de R, RStudio nos ofrece:

- Visualización de los gráficos integrada en el mismo entorno
- Acceso cómodo a la ayuda electrónica sobre R
- Un editor que nos permite guardar nuestro trabajo

En lugar de descargar e instalar RStudio, podemos ir a RStudio Cloud, iniciar sesión con nuestra cuenta Google o crear una cuenta a medida y usar RStudio desde el navegador.

¿Qué es un paquete en R?

R es una herramienta extensible. Las funciones que facilita el núcleo de R se pueden ampliar mediante **paquetes**. Los paquetes se instalan una única vez en el equipo, la primera vez que se necesitan, y luego se cargan cada vez que van a ser usados.

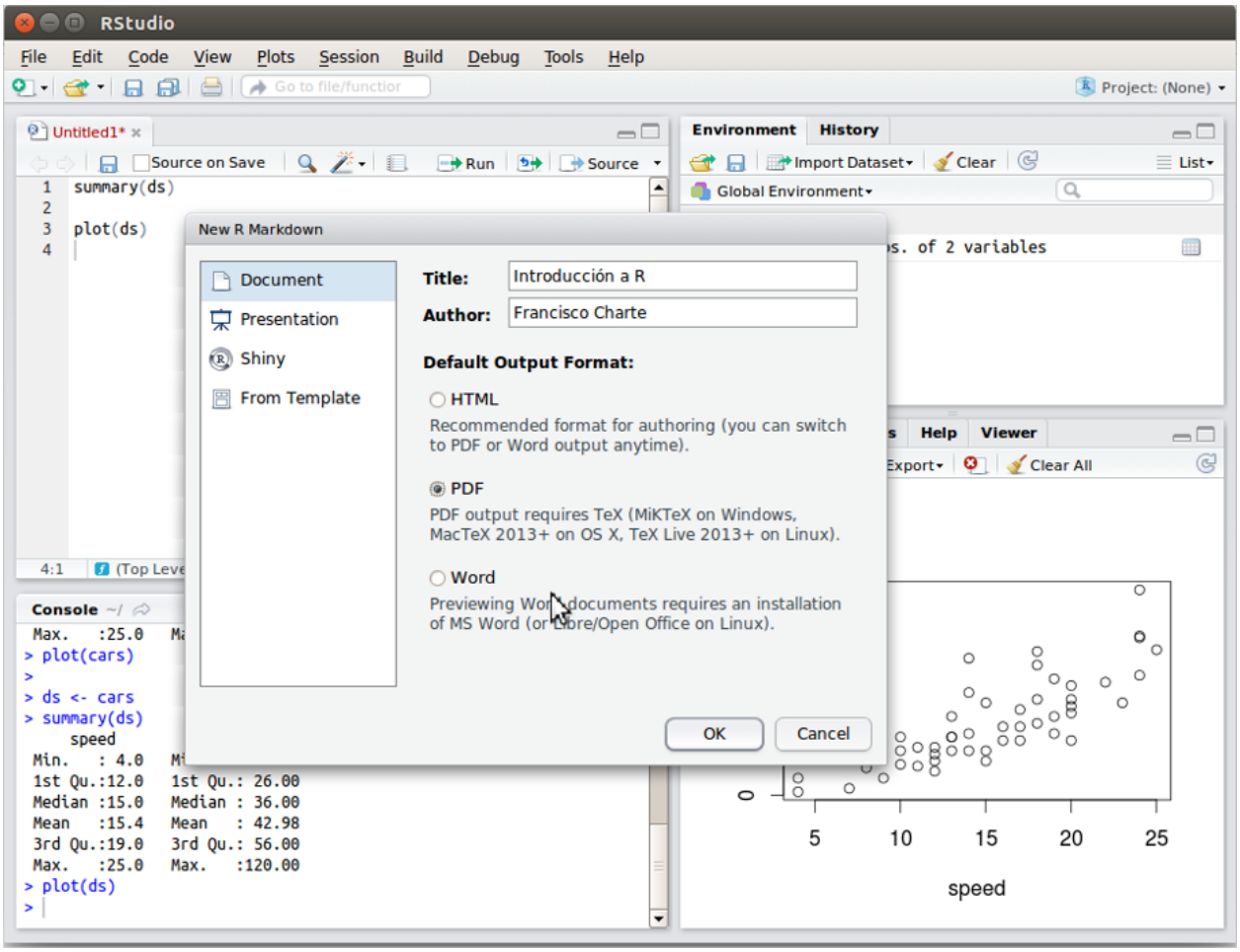


Figure 1: El entorno de RStudio

Para instalar un paquete usaremos la opción `Install Packages...` del menú `Tools` de RStudio, indicando el nombre del paquete que se necesita:

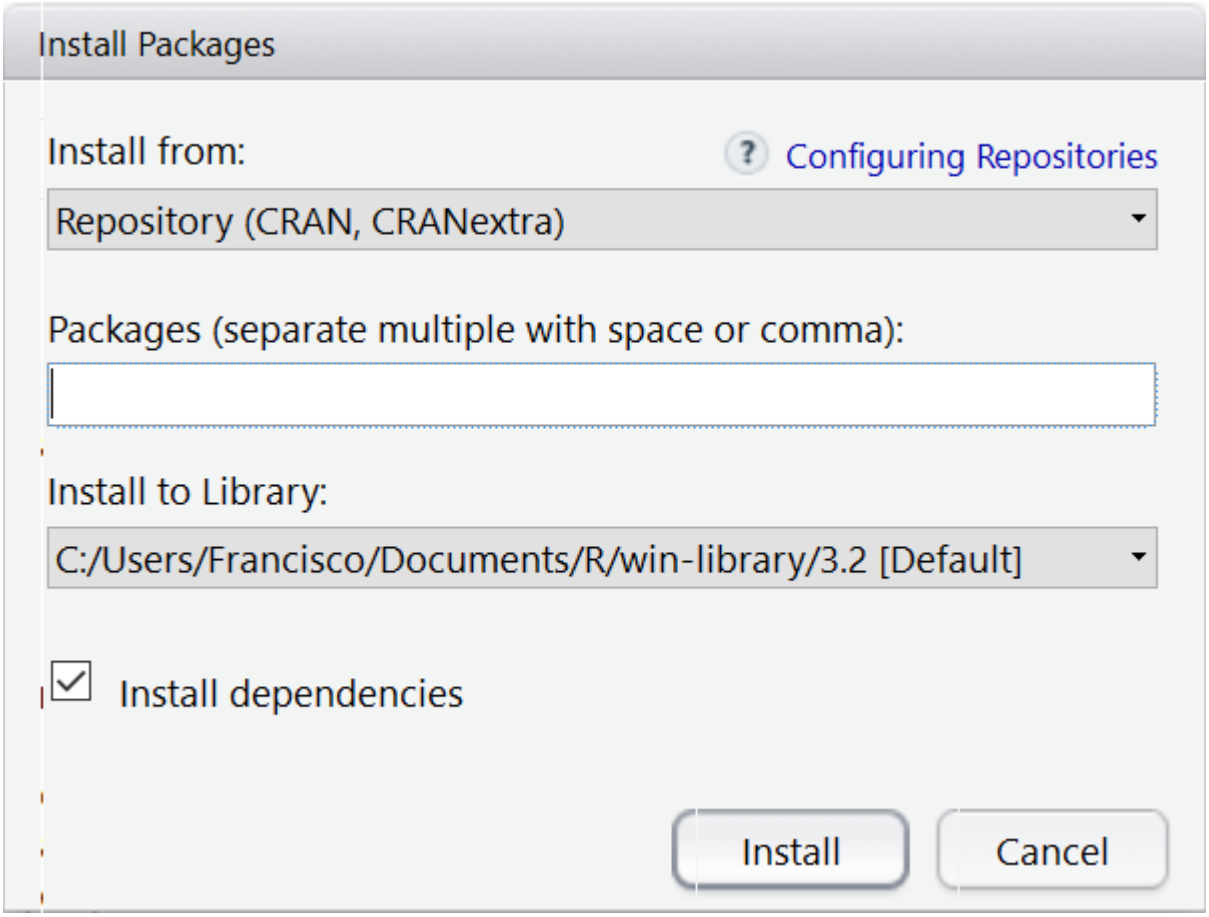


Figure 2: Opción de instalación de paquetes

Alternativamente, se puede instalar un paquete desde la línea de comandos de R con el comando `install.packages()`, facilitando entre los paréntesis y entrecomillado el nombre del paquete.

Una vez que el paquete está instalado en el sistema, cada vez que vaya a usarse hay que cargarlo desde el disco al entorno de R. Para ello se usa el comando `library()`. En el siguiente ejemplo se carga el paquete para gráficos `ggplot2`, lo cual permite utilizar los comandos que hay dicho paquete:

```
library(ggplot2)
```

Realización de tareas habituales

Algunas de las tareas que necesitaremos llevar a cabo habitualmente en el entorno de RStudio:

- **Cambiar la carpeta**: en el panel `Files` (parte inferior derecha del entorno) aparece en la parte superior la ruta de la carpeta actual. Podemos usar los botones para cambiarla. También se puede usar el comando `setwd()` en la consola de R para cambiar a la carpeta que se necesite. Por ejemplo:

```
setwd("D:/FCharte/Clases/2019-2020/MADOC")  # Cambiar a la carpeta de trabajo
```

- **Consultar la ayuda**: en el panel 'Help' se encuentran los elementos para acceder al índice y buscar cualquier tema en la documentación electrónica integrada. También podemos acceder a la misma desde

la consola de R, con el comando `help()`.

- **Cargar datos de un archivo de texto**: para trabajar con un conjunto de datos el primer paso es cargarlo en memoria. Asumiendo que los datos están en un archivo de texto, recurriremos a una de las siguientes alternativas:
  - **Formato CSV estándar**: los datos están separados por comas y se usa el punto decimal `read.csv("misdatos.csv")`

```
cultivos <- read.csv("datos/cultivos.csv")
cultivos
```

- **Formato CSV alternativo**: los datos están separados por punto y coma y se usa la coma decimal `read.csv2("misdatos.csv")`
- **Separación con tabuladores**: si los datos están separados con tabuladores, como ocurre al pegar desde Excel al portapapeles `read.delim("misdatos.txt")`
- **Cargar datos desde una hoja Excel**: en caso de que nuestros datos estén en una hoja Excel tenemos distintas alternativas para importarlos en R y trabajar con ellos:
  - **Mediante el portapapeles**: si tenemos los datos abiertos en Excel, esta es la vía más cómoda. El procedimiento consta de los pasos siguientes:
    1. Seleccionamos en Excel el rango de celdillas que contienen los datos que nos interesan
    2. Copiamos los datos al portapapeles
    3. Cambiamos al entorno de R e introducimos en la consola el siguiente comando: `misdatos <- read.delim("clipboard")`
  - **Mediante el paquete xlsx**: si los datos están almacenados en un libro Excel que no tenemos abierto, podemos recurrir al siguiente procedimiento:
    1. Cargamos el paquete `xlsx` con el comando `library("xlsx")`. Si este comando falla es porque no tenemos instalado dicho paquete, en cuyo caso tendríamos que instalarlo primero.
    2. Usamos el comando `read.xlsx()` de dicho paquete para cargar los datos: `misdatos <- read.xlsx("libroexcel.xlsx", sheetName = "nombreHoja")`

```
library(xlsx)
```

```
## Warning: package 'xlsx' was built under R version 3.6.3
```

```
poblacion <- read.xlsx("datos/DatosPoblacion.xlsx", "HistoricoMundial", encoding = "UTF-8")
poblacion
```

```
##     Año Población.mundial
## 1  1000               410
## 2  1650               545
## 3  1750               791
## 4  1800               981
## 5  1850              1262
## 6  1900              1650
## 7  1950              2516
## 8  1955              2751
## 9  1960              3018
## 10 1965              3335
## 11 1970              3697
## 12 1975              4077
## 13 1980              4446
## 14 1985              4854
## 15 1990              5259
## 16 1995              5759
```

```
## 17 2000              6228
```

- **Obtener una lista de conjuntos de datos disponibles**: R incorpora multitud de conjuntos de datos de ejemplo con los que podemos trabajar. Para obtener una lista de ellos usaremos el comando `data()`. El nombre que aparece en la columna de la izquierda es el nombre del conjunto de datos, el que utilizaremos para trabajar con su contenido. En los ejemplos del punto anterior, al cargar datos externos, asignábamos el nombre `misdatos` a nuestro conjunto de datos.
- **Obtener información de un conjunto de datos**: basta introducir `?nombre` en la línea de comandos para obtener información sobre los cambios de un conjunto de datos:

```
?diamonds
```

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

Análisis exploratorio de datos

En esta sección vamos a trabajar con uno de los conjuntos de datos de ejemplo que incorpora R: `iris`. En él se facilita información de distintas familias de lirios (flores). Veamos cómo explorar ese conjunto de datos para obtener una visión general de su contenido.

Estructura del conjunto de datos

```
str(iris)  # Resumen de la estructura
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
ncol(iris) # Cantidad de columnas
```

```
## [1] 5
```

```
nrow(iris) # Cantidad de filas
```

```
## [1] 150
```

```
colnames(iris) # Nombres de las columnas
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
head(iris) # Ver algunas filas del inicio del conjunto de datos
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
tail(iris) # Ver algunas filas del final del conjunto de datos
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
```

```
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```
iris$Sepal.Width # Ver todos los valores de una de las columnas
```

```
##   [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5
##  [19] 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
##  [37] 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
##  [55] 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7 2.2 2.5 3.2 2.8
##  [73] 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0 3.4 3.1 2.3 3.0 2.5
##  [91] 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7 3.0 2.9 3.0 3.0 2.5 2.9
## [109] 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
## [127] 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2
## [145] 3.3 3.0 2.5 3.0 3.4 3.0
```

Obtención de estadísticos básicos

```
min(iris$Sepal.Width) # Valor mínimo de Sepal.Width
```

```
## [1] 2
```

```
max(iris$Sepal.Width) # Valor máximo de de Sepal.Width
```

```
## [1] 4.4
```

```
mean(iris$Sepal.Width) # Media de Sepal.Width
```

```
## [1] 3.057333
```

```
median(iris$Sepal.Width) # Mediana de Sepal.Width
```

```
## [1] 3
```

```
var(iris$Sepal.Width) # Varianza de Sepal.Width
```

```
## [1] 0.1899794
```

```
sd(iris$Sepal.Width) # Desviación de Sepal.Width
```

```
## [1] 0.4358663
```

Estadística descriptiva de todo el conjunto de datos

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
```

```
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Cálculo del coeficiente de correlación

```
cor(iris$Petal.Length, iris$Petal.Width) # Pearson entre Petal.Length y Petal.Width
```

```
## [1] 0.9628654
```

Tabla de contingencia de dos variables

```
table(iris$Petal.Width, iris$Species)
```

```
##
##       setosa versicolor virginica
##   0.1      5          0         0
##   0.2     29          0         0
##   0.3      7          0         0
##   0.4      7          0         0
##   0.5      1          0         0
##   0.6      1          0         0
##   1        0          7         0
##   1.1      0          3         0
##   1.2      0          5         0
##   1.3      0         13         0
##   1.4      0          7         1
##   1.5      0         10         2
##   1.6      0          3         1
##   1.7      0          1         1
##   1.8      0          1        11
##   1.9      0          0         5
##   2        0          0         6
##   2.1      0          0         6
##   2.2      0          0         3
##   2.3      0          0         8
##   2.4      0          0         3
##   2.5      0          0         3
```

La importancia de visualizar los datos

Los indicadores estadísticos resumen un gran conjunto de datos en unos pocos valores, lo cual **implica pérdida de información** que puede resultar de importancia.

El conjunto de datos `anscombe`

Este conjunto de datos son realmente cuatro subconjuntos formados por parejas `x-y`. Usemos alguno de los comandos de R que hemos conocido para explorar sus características. La función `round()` redondea el

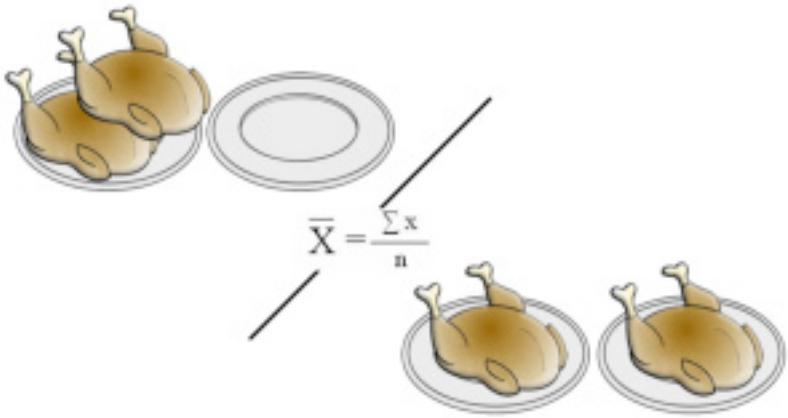


Figure 3: La métafora de los pollos

resultado: las medias, varianzas, etc., al número de decimales indicado.

```
str(anscombe)  # Estructura del conjunto de datos
```

```
## 'data.frame':    11 obs. of  8 variables:
##  $ x1: num  10 8 13 9 11 14 6 4 12 7 ...
##  $ x2: num  10 8 13 9 11 14 6 4 12 7 ...
##  $ x3: num  10 8 13 9 11 14 6 4 12 7 ...
##  $ x4: num  8 8 8 8 8 8 8 19 8 8 ...
##  $ y1: num  8.04 6.95 7.58 8.81 8.33 ...
##  $ y2: num  9.14 8.14 8.74 8.77 9.26 8.1 6.13 3.1 9.13 7.26 ...
##  $ y3: num  7.46 6.77 12.74 7.11 7.81 ...
##  $ y4: num  6.58 5.76 7.71 8.84 8.47 7.04 5.25 12.5 5.56 7.91 ...
```

```
round(c(mean(anscombe$x1),
        mean(anscombe$x2),
        mean(anscombe$x3),
        mean(anscombe$x4)),1) # Medias de las X
```

```
## [1] 9 9 9 9
```

```
round(c(mean(anscombe$y1),
        mean(anscombe$y2),
        mean(anscombe$y3),
        mean(anscombe$y4)),1) # Medias de las Y
```

```
## [1] 7.5 7.5 7.5 7.5
```

```
round(c(var(anscombe$x1),
        var(anscombe$x2),
        var(anscombe$x3),
        var(anscombe$x4)),1) # Varianzas de las X
```

```
## [1] 11 11 11 11
```

```
round(c(var(anscombe$y1),
        var(anscombe$y2),
        var(anscombe$y3),
        var(anscombe$y4)),1) # Varianzas de las Y
```

```
## [1] 4.1 4.1 4.1 4.1
```

```r
round(c(cor(anscombe$x1, anscombe$y1),   # Coeficientes de correlación
  cor(anscombe$x2, anscombe$y2),
  cor(anscombe$x3, anscombe$y3),
  cor(anscombe$x4, anscombe$y4)), 2)
```

```
## [1] 0.82 0.82 0.82 0.82
```

```r
lm(anscombe$y1 ~ anscombe$x1)   # Línea de regresión
```

```
##
## Call:
## lm(formula = anscombe$y1 ~ anscombe$x1)
##
## Coefficients:
## (Intercept)  anscombe$x1
##      3.0001       0.5001
```

```r
lm(anscombe$y2 ~ anscombe$x2)
```

```
##
## Call:
## lm(formula = anscombe$y2 ~ anscombe$x2)
##
## Coefficients:
## (Intercept)  anscombe$x2
##       3.001        0.500
```

```r
lm(anscombe$y3 ~ anscombe$x3)
```

```
##
## Call:
## lm(formula = anscombe$y3 ~ anscombe$x3)
##
## Coefficients:
## (Intercept)  anscombe$x3
##      3.0025       0.4997
```

```r
lm(anscombe$y4 ~ anscombe$x4)
```

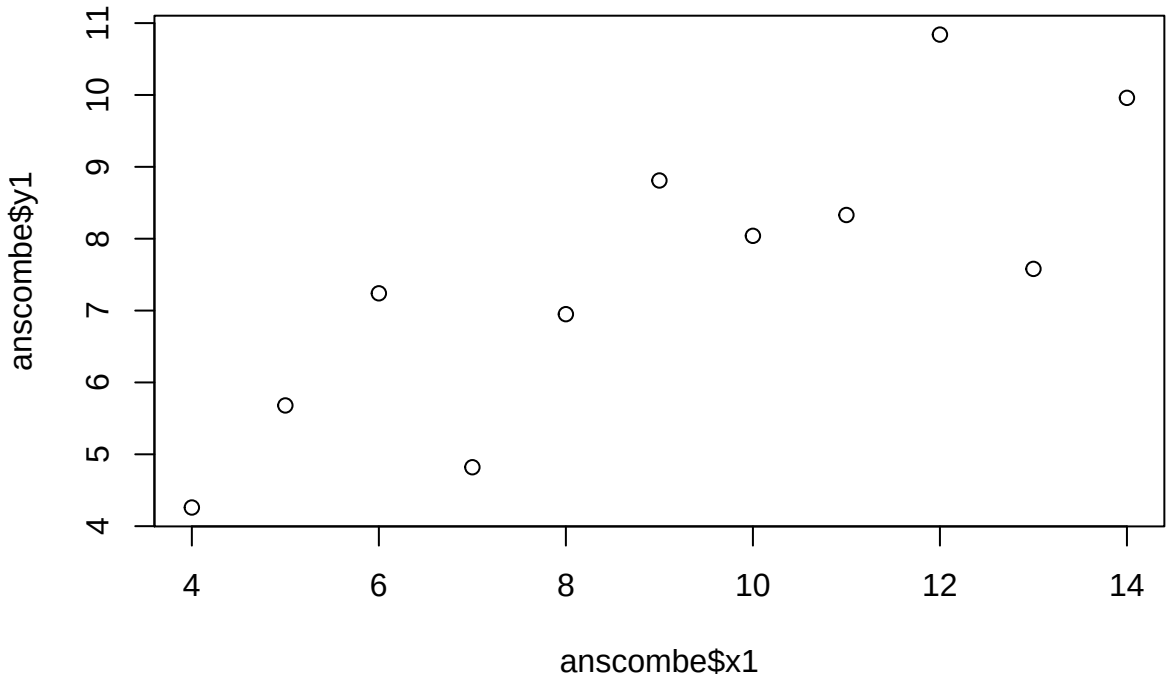
```
##
## Call:
## lm(formula = anscombe$y4 ~ anscombe$x4)
##
## Coefficients:
## (Intercept)  anscombe$x4
##      3.0017       0.4999
```

A la vista de estos resultados, podríamos concluir que los subconjuntos de datos `(xN, yN)` siguen todos la misma distribución y, por tanto, su población es básicamente idéntica. ¿SEGURO?

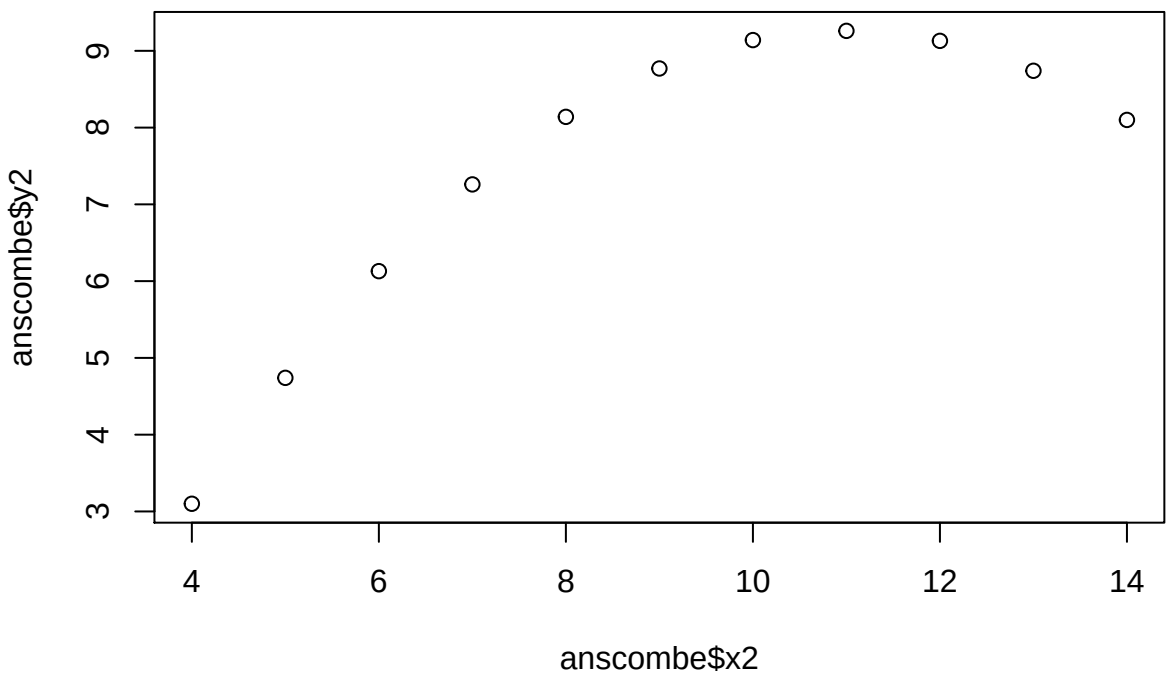
Visualización de las muestras de datos

La vía más efectiva y rápida, y en ocasiones la única, para comprobar si las muestras de datos tienen una distribución similar es la visualización.

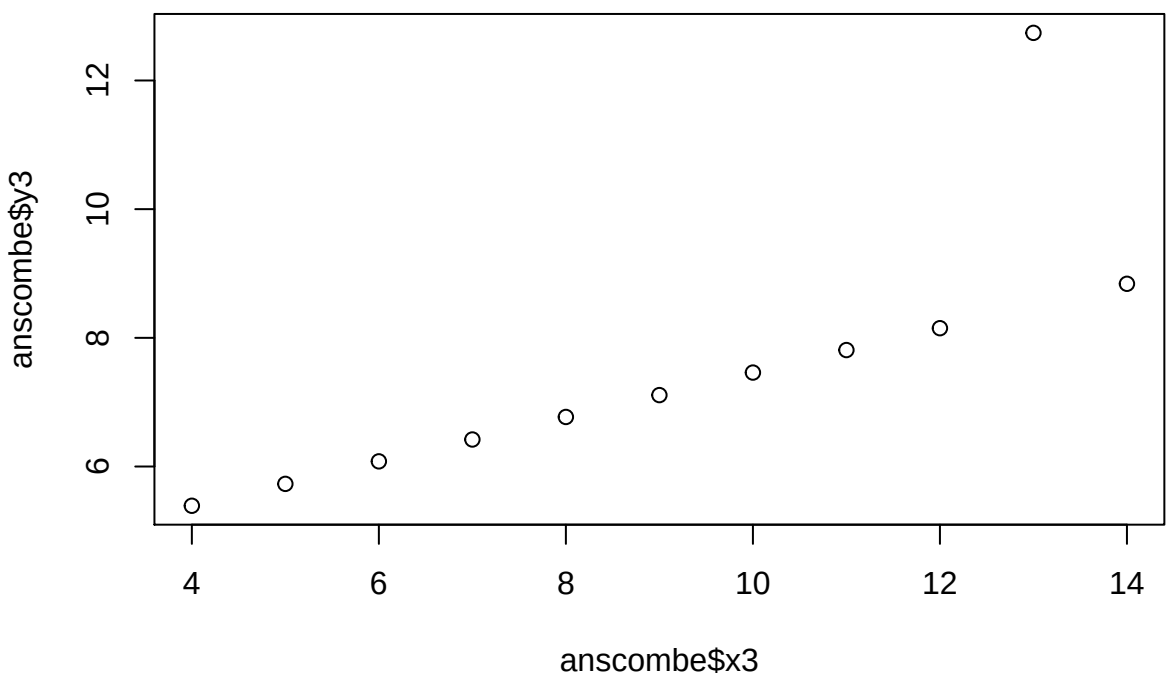
```
plot(anscombe$x1, anscombe$y1)
```



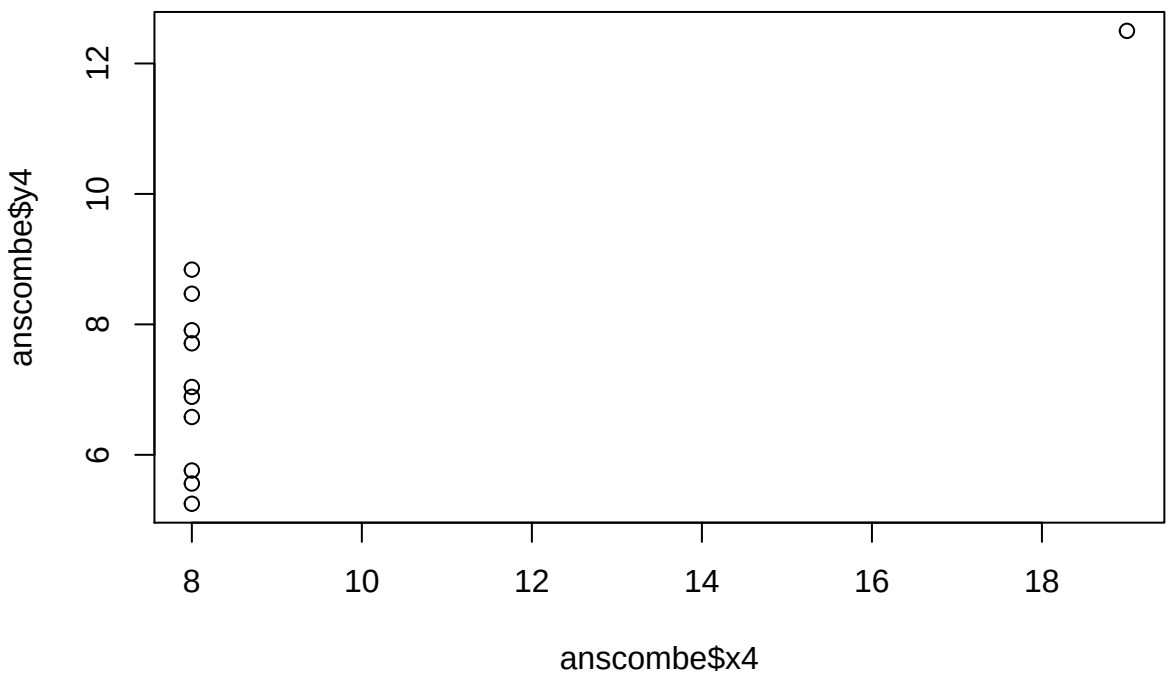
```r
plot(anscombe$x2, anscombe$y2)
```



```r
plot(anscombe$x3, anscombe$y3)
```



```r
plot(anscombe$x4, anscombe$y4)
```



Como es fácil apreciar, la distribución de estos cuatro subconjuntos de datos son totalmente distintas, de ahí la importancia de visualizar los datos y no confiar únicamente en los indicadores estadísticos.

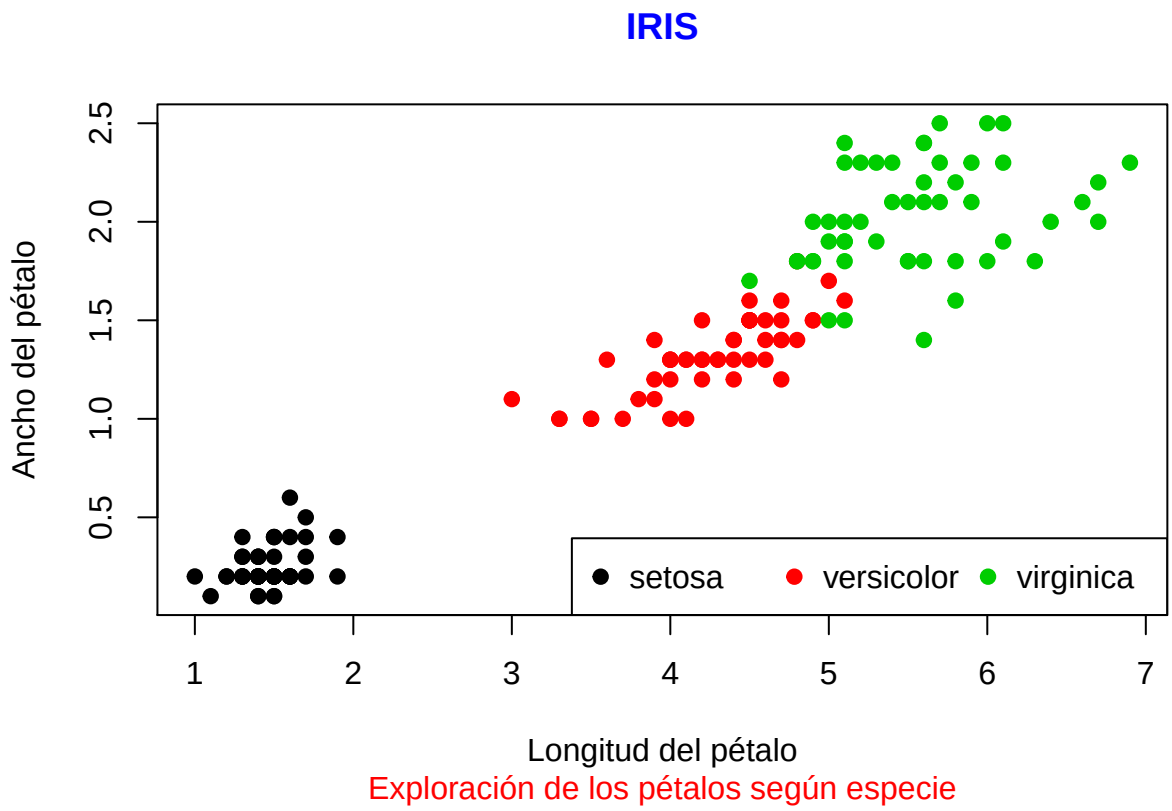
Visualización básica de datos

La visualización es una herramienta **esencial en el análisis exploratorio** de datos. No obstante existen multitud de tipos de visualización distintos y no todos sirven en todos los casos. Al que tiene un martillo todo le parecen clavos, pero es importante **elegir la herramienta adecuada** a cada trabajo. Dependiendo de los datos y de la información que se quiera analizar en los mismos, será preciso seleccionar el tipo de visualización adecuado.

Gráfica de dispersión (*scatter plot*)

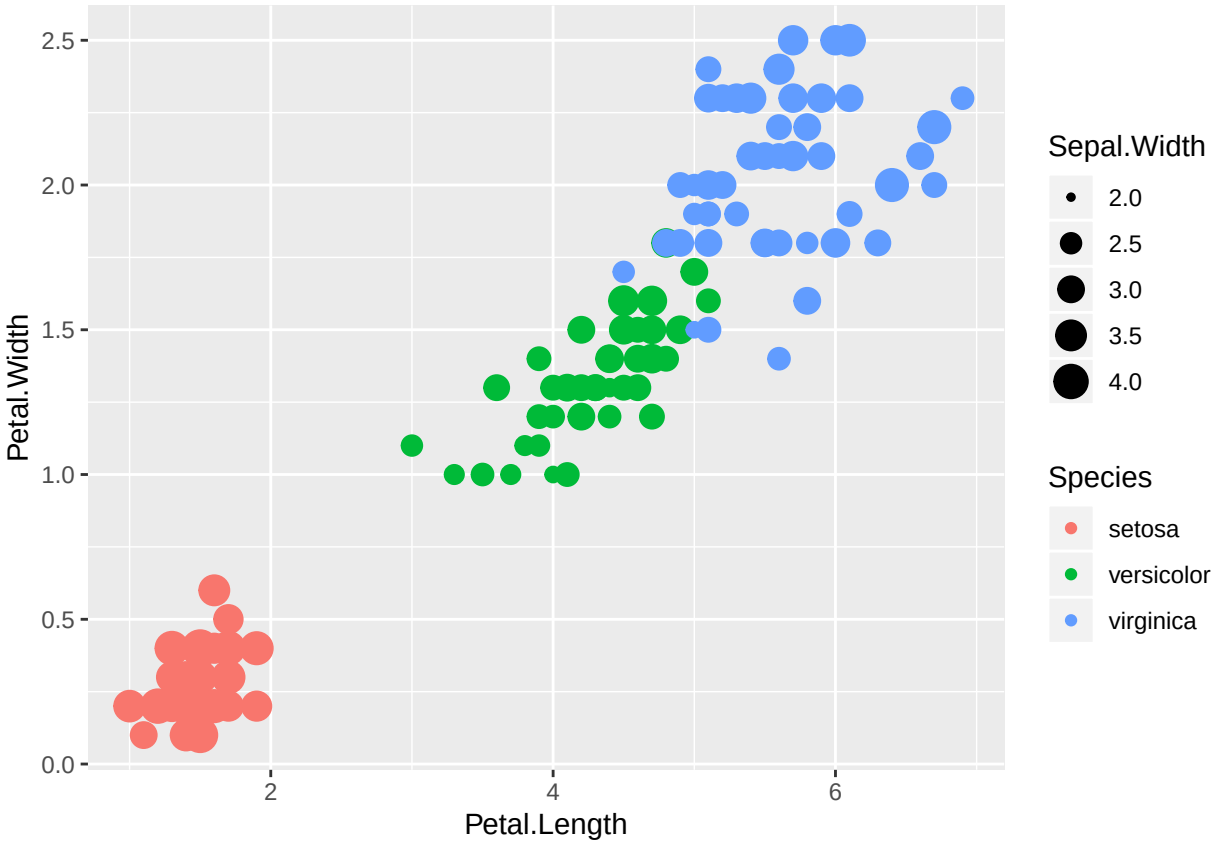
Nos permite examinar la posición de los individuos en el plano (2D) o espacio (3D).

```
plot(iris$Petal.Length, iris$Petal.Width, col=iris$Species,  pch = 19,
     xlab = 'Longitud del pétalo', ylab = 'Ancho del pétalo')

title(main = 'IRIS',
      sub = 'Exploración de los pétalos según especie',
      col.main = 'blue', col.sub = 'red')

legend("bottomright", legend = levels(iris$Species),
       col = unique(iris$Species), ncol = 3, pch = 19, bty = "y")
```



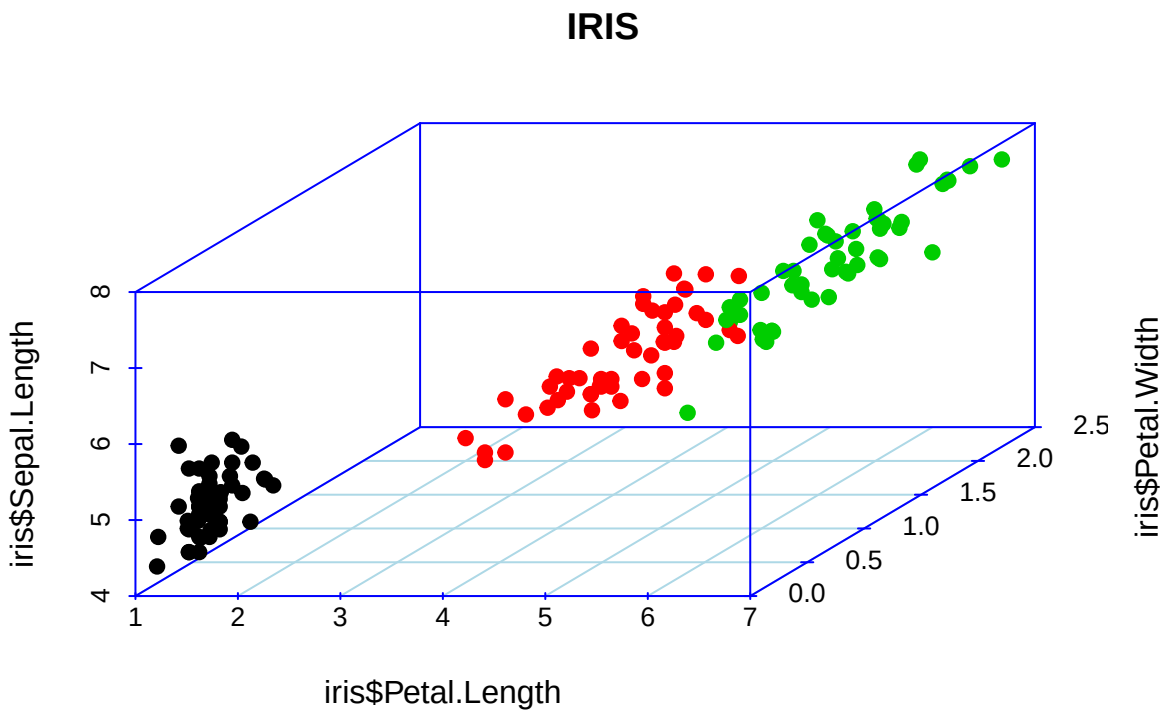
El comando `plot()` forma parte de la instalación básica de R, pero hay paquetes que ofrecen gráficos de mayor calidad. Uno de ellos es `ggplot2`.

```
library(ggplot2)

ggplot(iris, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(colour=Species, size = Sepal.Width))
```



Con el paquete `scatterplot3d` podemos usar el comando del mismo nombre, capaz de representar los puntos en 3D en lugar de un plano:

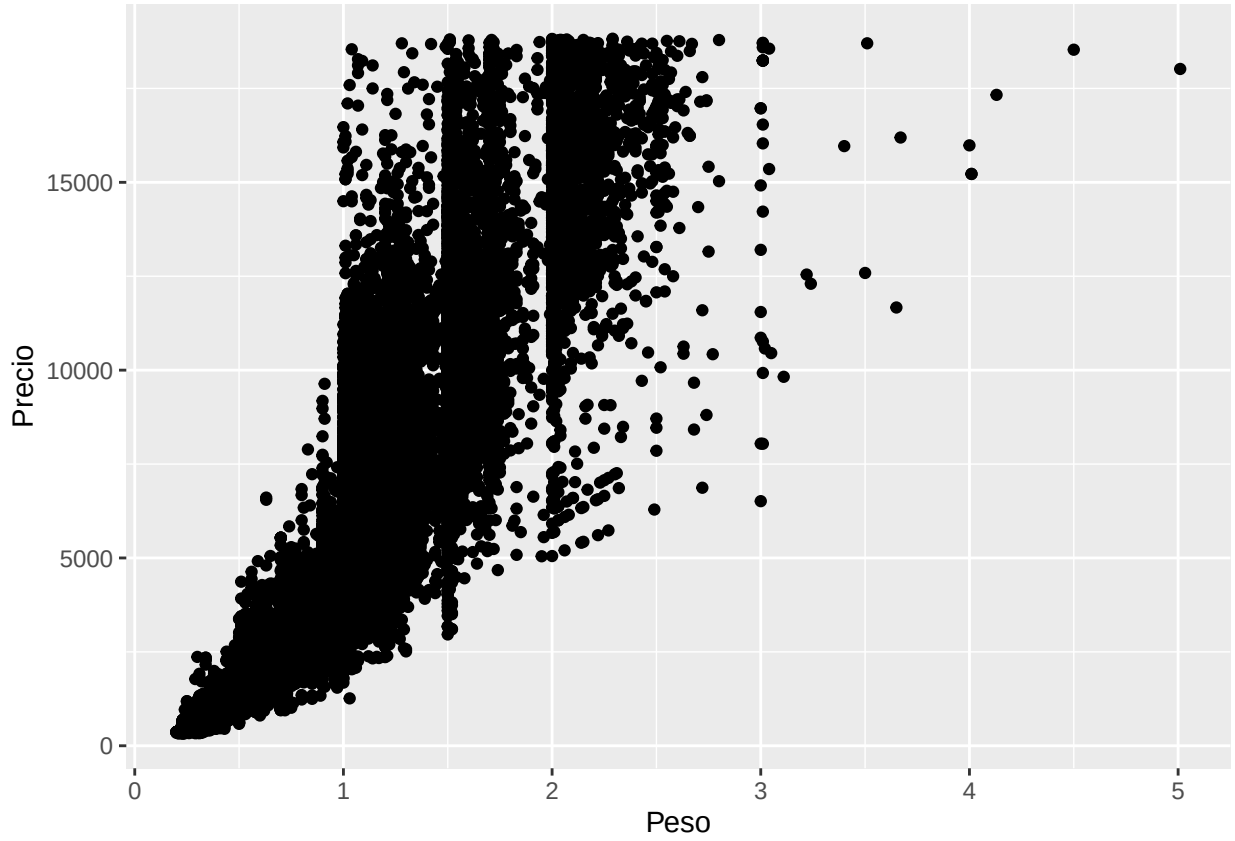
```
library(scatterplot3d)

scatterplot3d(iris$Petal.Length, iris$Petal.Width, iris$Sepal.Length,
              col.axis = "blue", color = as.numeric(iris$Species),
              col.grid = "lightblue", main = "IRIS", pch = 19)
```



Gráfica de dispersión con densidad (*hexbin*)

En ocasiones una gráfica de dispersión puede ocultar información si hay mucho solapamiento entre los puntos:

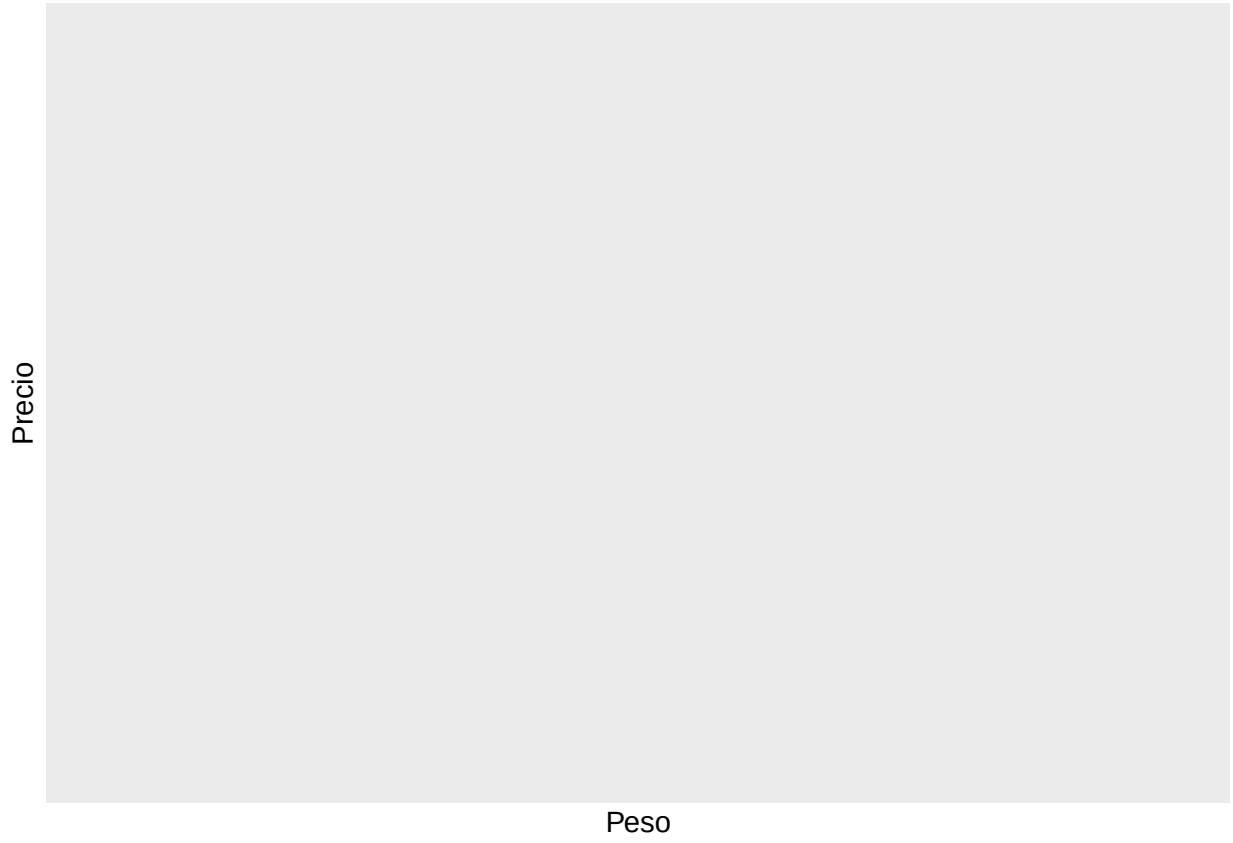
```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + labs(x = "Peso", y = "Precio")
```



En estos casos se puede recurrir a un método que consiste en dividir la gráfica en hexágonos, contar el número de puntos que hay en su interior y colorear según la densidad:

```
ggplot(diamonds, aes(x=carat, y=price)) + stat_binhex() + labs(x = "Peso", y = "Precio")
```

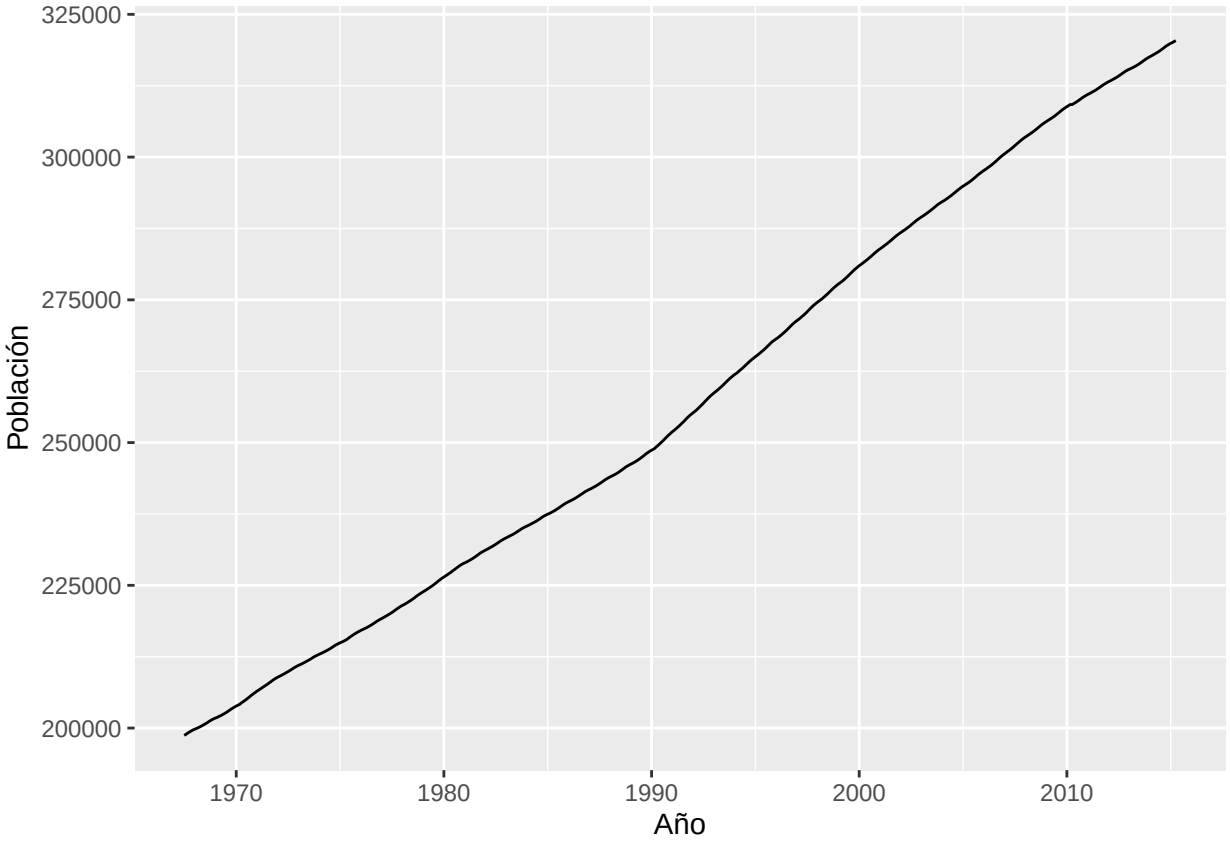
```
## Warning: Computation failed in `stat_binhex()`:
## Package `hexbin` required for `stat_binhex`.
## Please install and try again.
```



Gráficas de líneas y barras

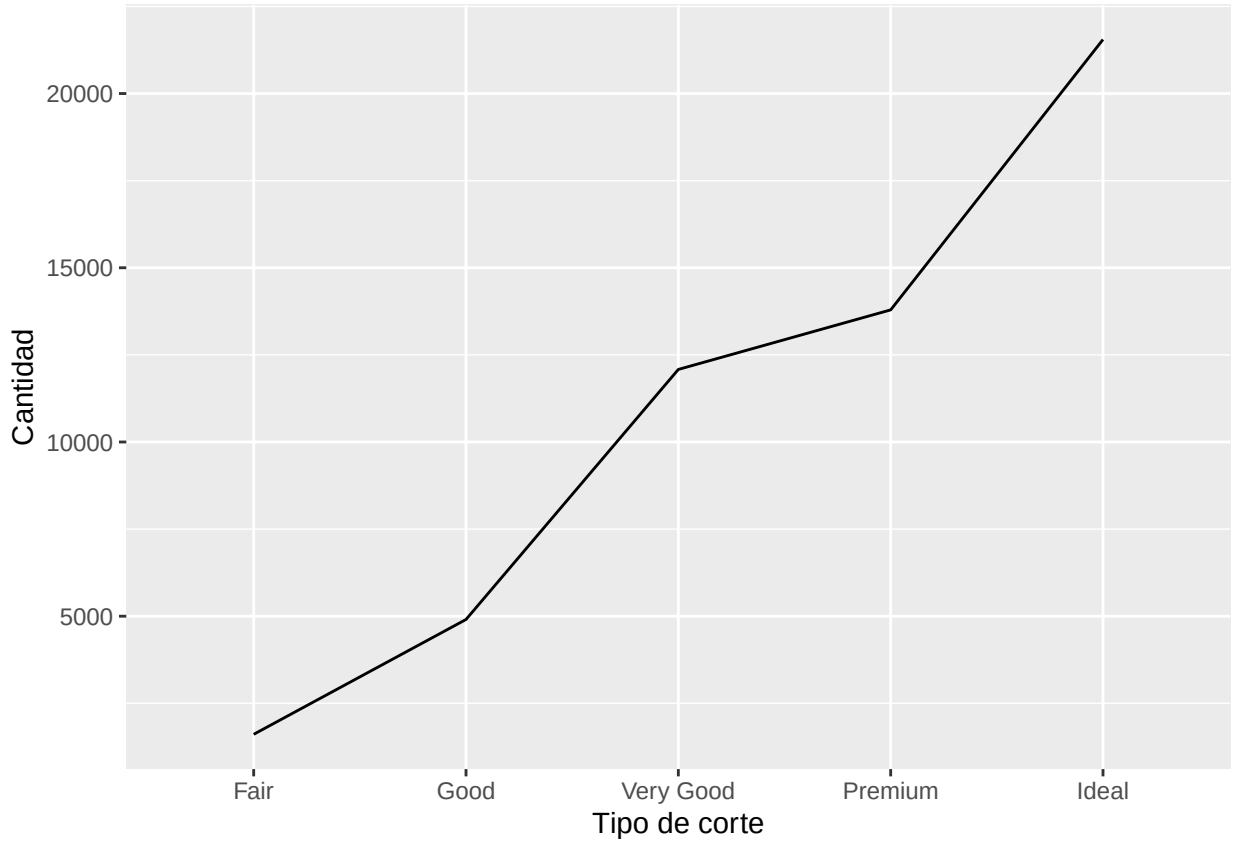
Las gráficas de líneas deben utilizarse exclusivamente para analizar la evolución de una magnitud respecto a otra. Por ejemplo:

```
ggplot(economics, aes(x=date, y=pop)) + geom_line() + labs(x = "Año", y = "Población")
```

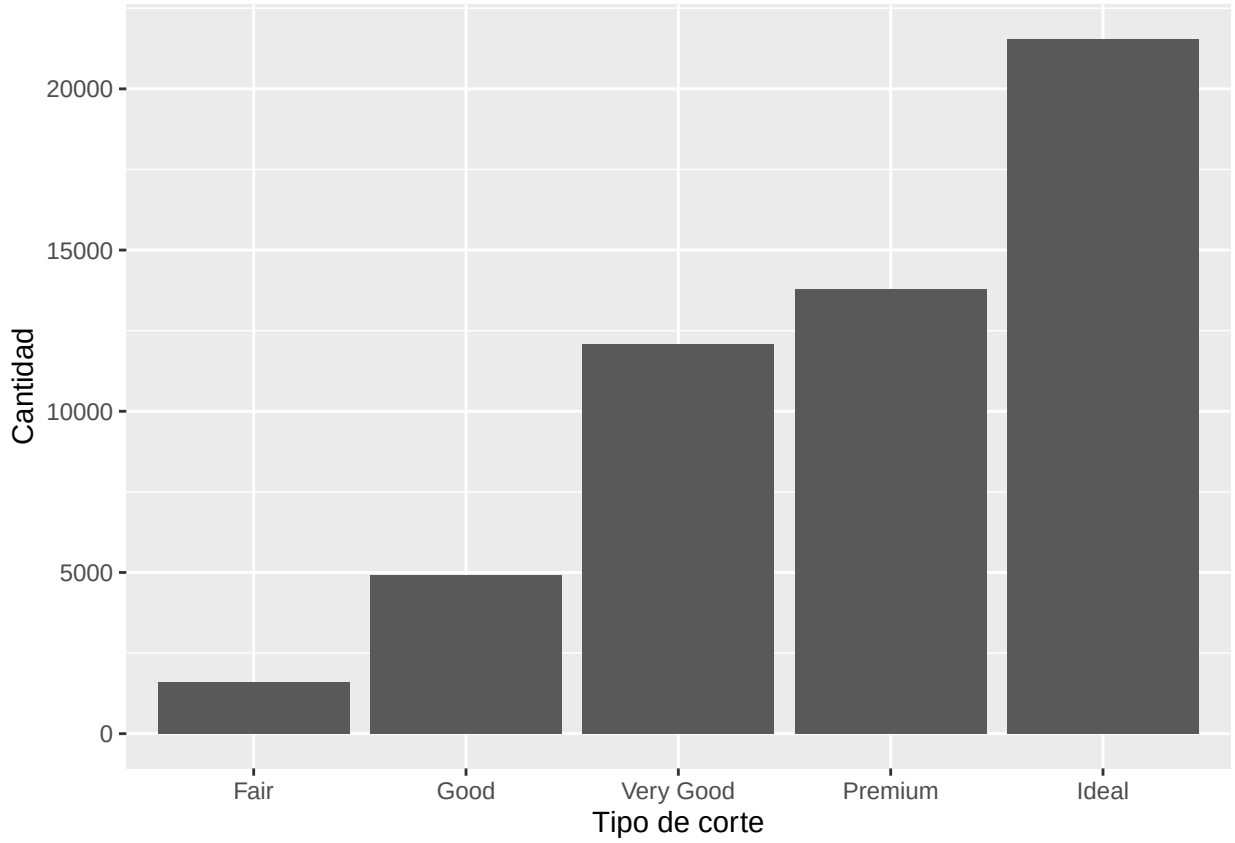


Cuando lo que interesa no es la evolución en sí, sino la relación (comparación) entre cada entrada en el eje X y la altura en el eje Y, habitualmente es mejor usar una gráfica de barras:

```
ggplot(diamonds, aes(cut)) + geom_line(aes(group="cut"), stat="count") +
  labs(x = "Tipo de corte", y = "Cantidad")
```

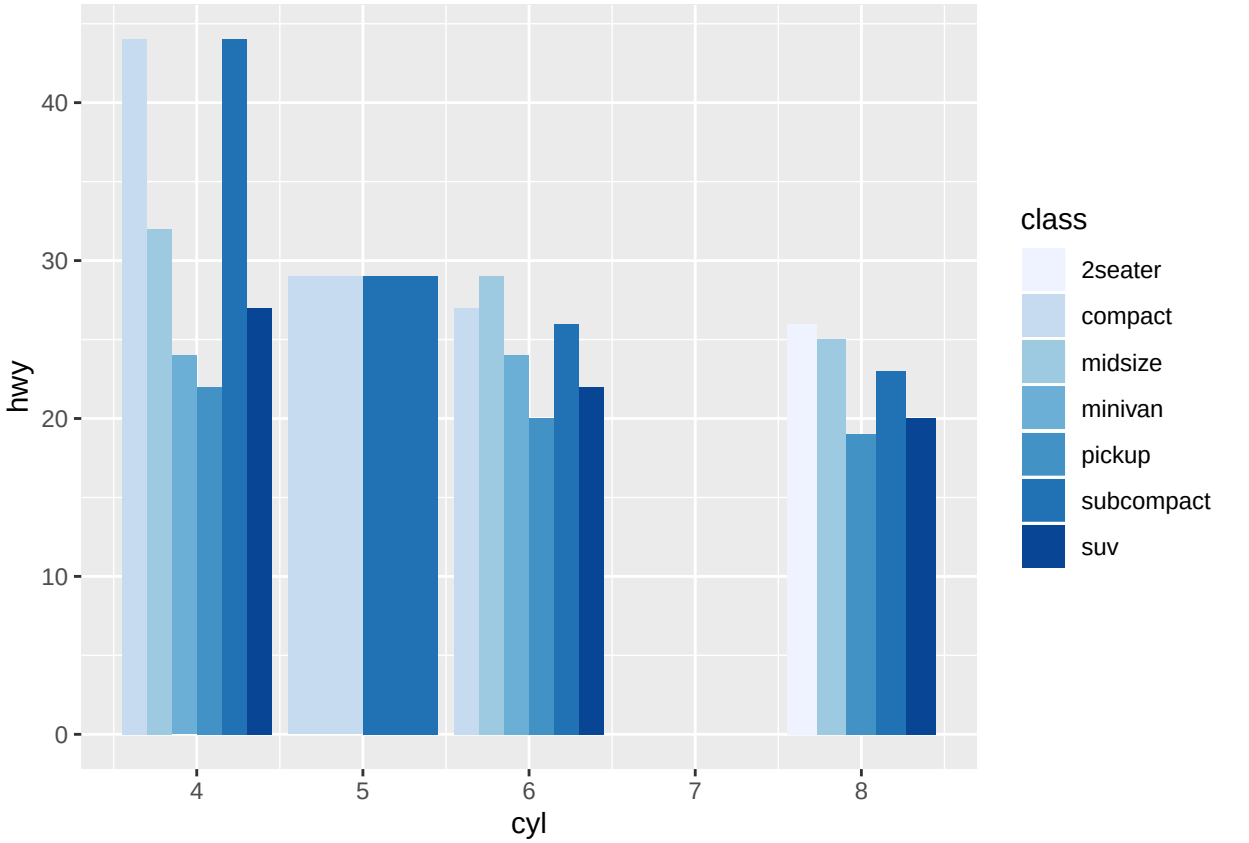


```r
ggplot(diamonds, aes(cut)) + geom_bar() + labs(x = "Tipo de corte", y = "Cantidad")
```



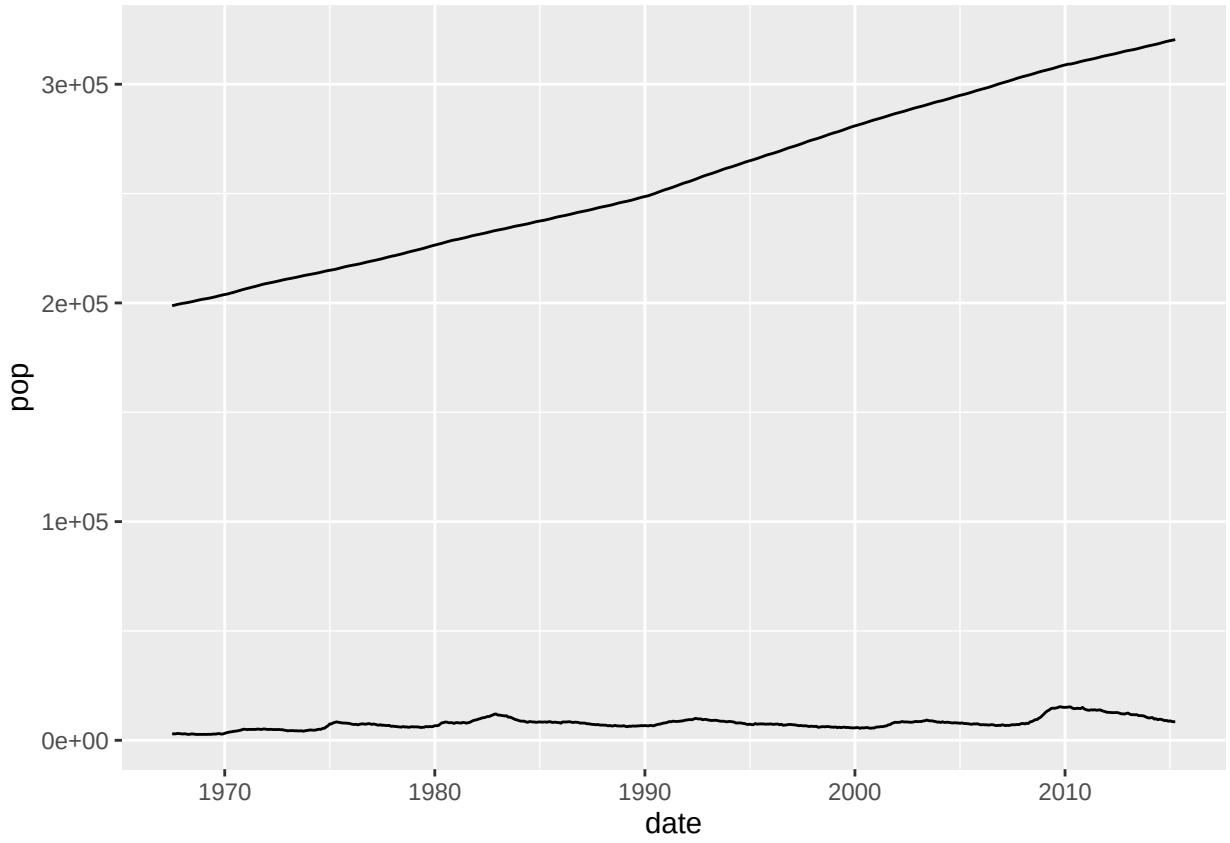
Es posible comparar más de una magnitud simultáneamente, agrupando las barras según los valores de una variable:

```
ggplot(mpg, aes(cyl, hwy, fill=class)) + scale_fill_brewer() +
  geom_bar(stat="identity", position="dodge")
```



Asimismo, podemos introducir varias líneas en una misma gráfica para representar la evolución de dos magnitudes:

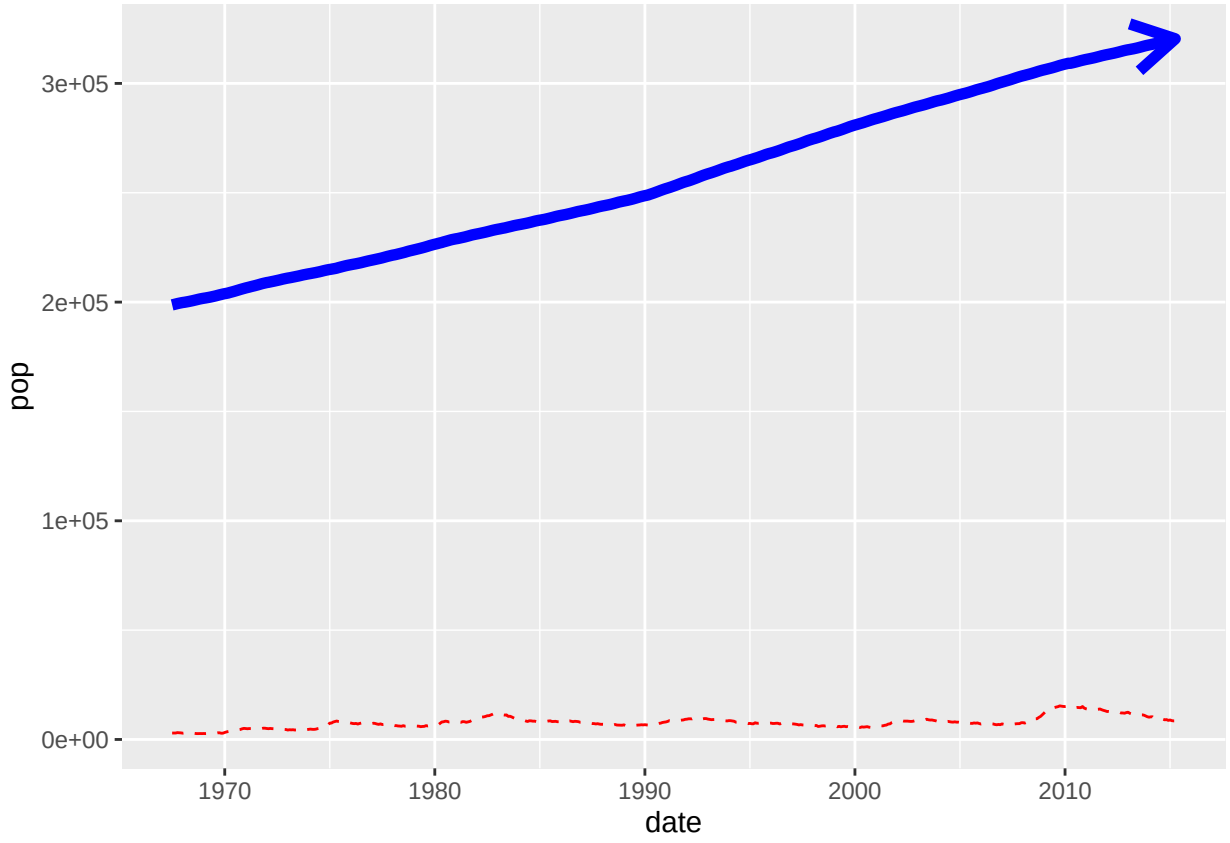
```
ggplot(economics, aes(x=date)) + geom_line(aes(y=pop)) + geom_line(aes(y=unemploy))
```



Ajustes visuales sobre gráficas de líneas y barras

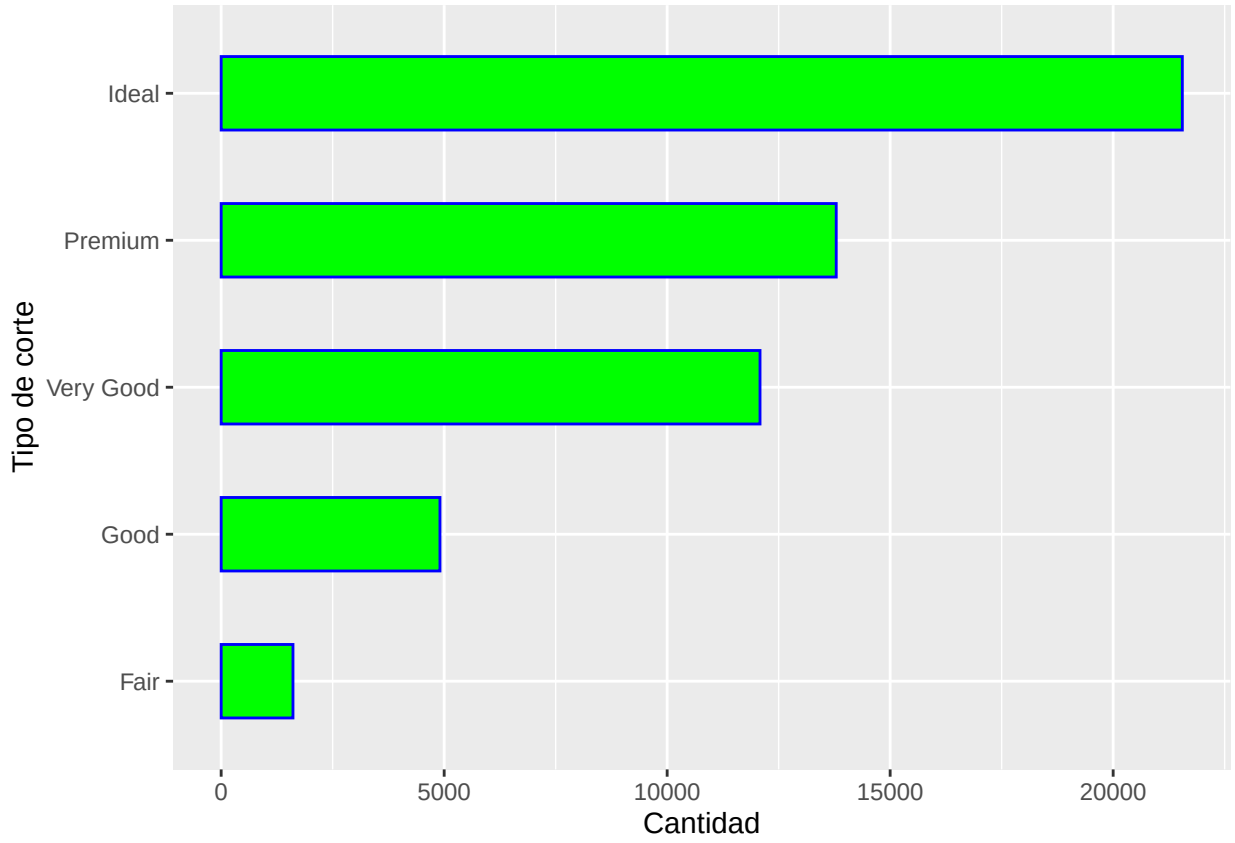
- Podemos personalizar la apariencia de una gráfica de líneas con los siguientes parámetros:
  - `colour`: establece el color de la línea. Ejemplos: `colour = "red"` `colour = class`
  - `size`: establece el grosor de la línea. Ejemplos: `size = 1` `size = cyl`
  - `linetype`: establece el tipo de trazo. Ejemplos: `linetype = 3` `linetype = "dashed"`
  - `arrow`: permite agregar flechas a las líneas: Ejemplo: `arrow = arrow()`

```
ggplot(economics, aes(x=date)) +
  geom_line(aes(y=pop), linetype="solid", size=2, color="blue", arrow = arrow()) +
  geom_line(aes(y=unemploy), linetype = "dashed", color = "red")
```



- Podemos personalizar la apariencia de una gráfica de barras con los siguientes parámetros:
  - `width`: establece el ancho relativo de las barras. Ejemplo: `width = 0.5`
  - `colour`: establece el color del borde de las barras. Ejemplo: 'colour="blue"
  - `fill`: establece el color de relleno de las barras. Ejemplo: `fill="red"`
  - `coord_flip()`: no es un parámetro, sino un objeto que se agrega para cambiar la orientación de las barras

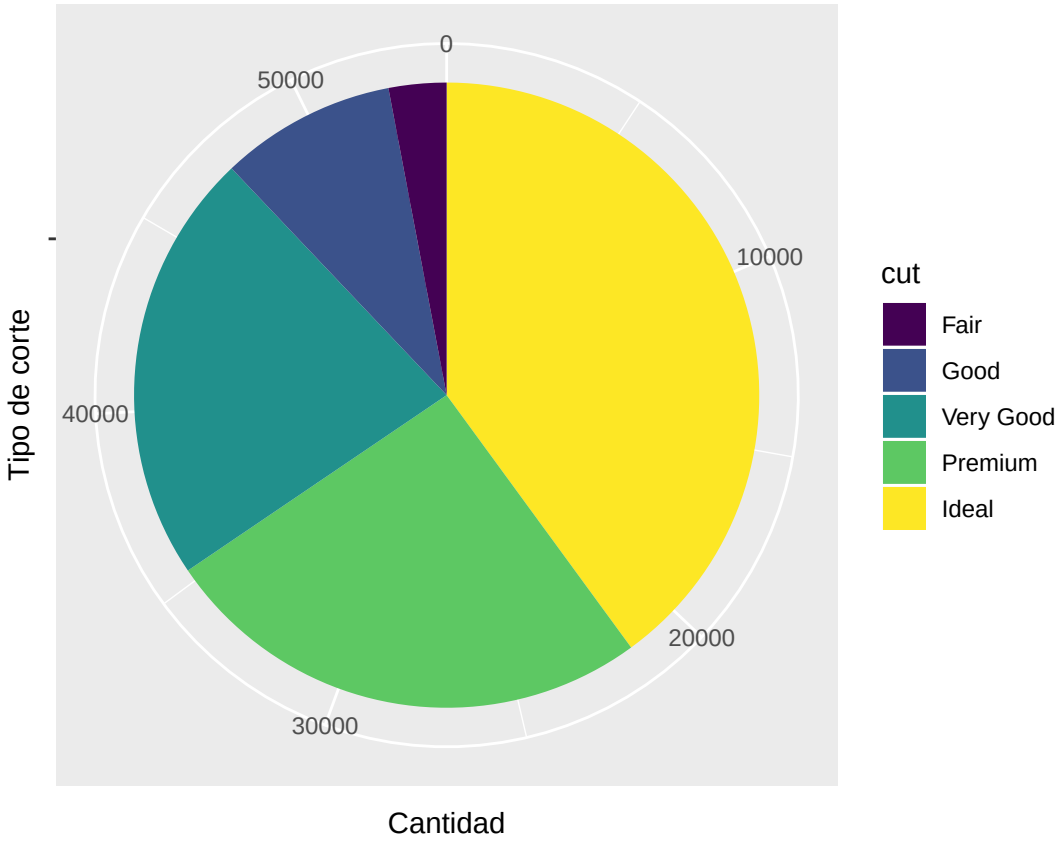
```
ggplot(diamonds, aes(cut)) +
  geom_bar(colour="blue", fill="green", width=0.5) +
  coord_flip() + labs(x = "Tipo de corte", y = "Cantidad")
```



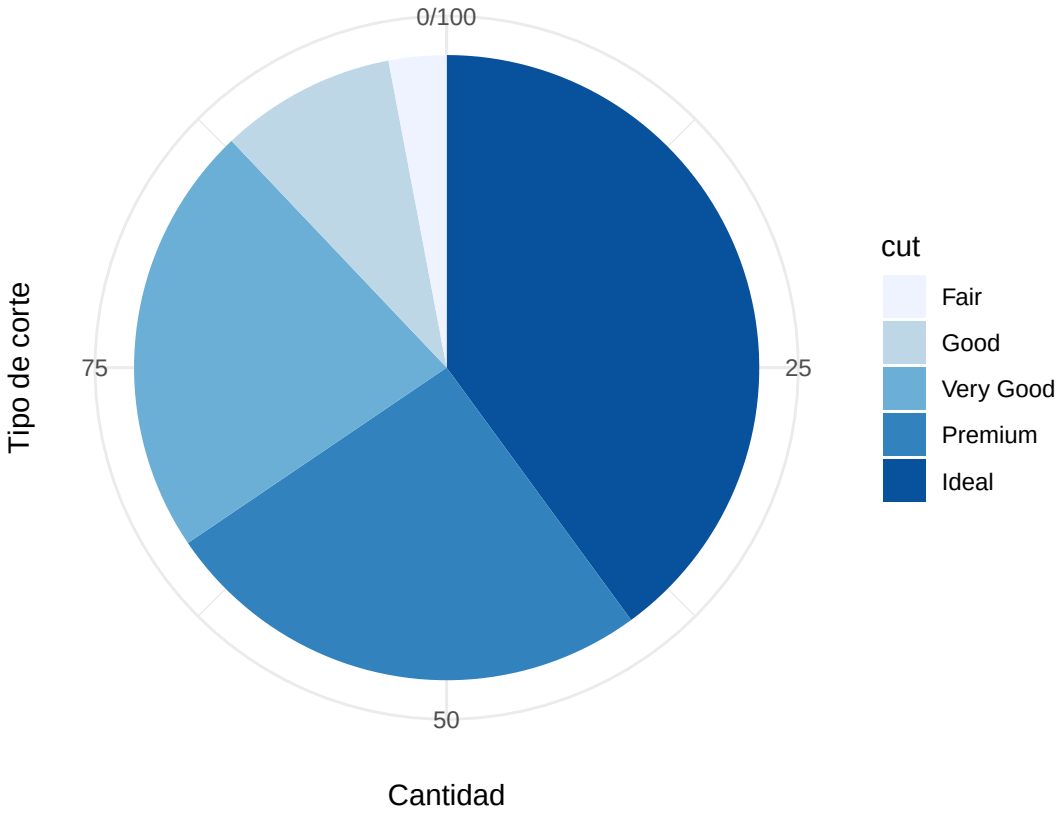
Cantidades que forman parte de un todo

Cuando las magnitudes que están comparándose en una gráfica conforman un todo, podemos optar por otro tipo de representación que denote esa idea. La primera opción, para una única serie de datos, sería una gráfica de tarta o bien una barra apilada:

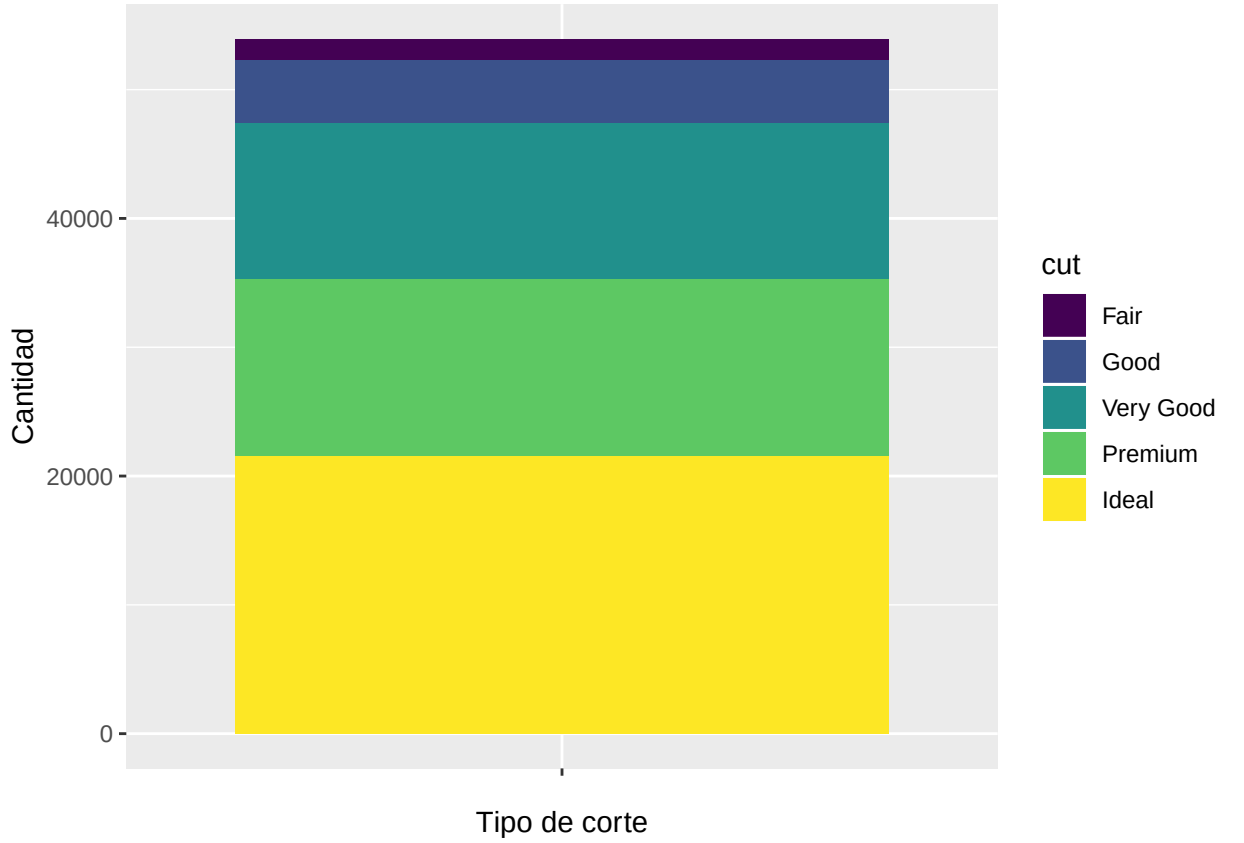
```
grafica <- ggplot(diamonds, aes("", fill=cut)) +
  labs(x="Tipo de corte", y="Cantidad")   # Gráfica con los datos básicos

grafica + geom_bar() + coord_polar("y")  # Gráfica de tarta
```



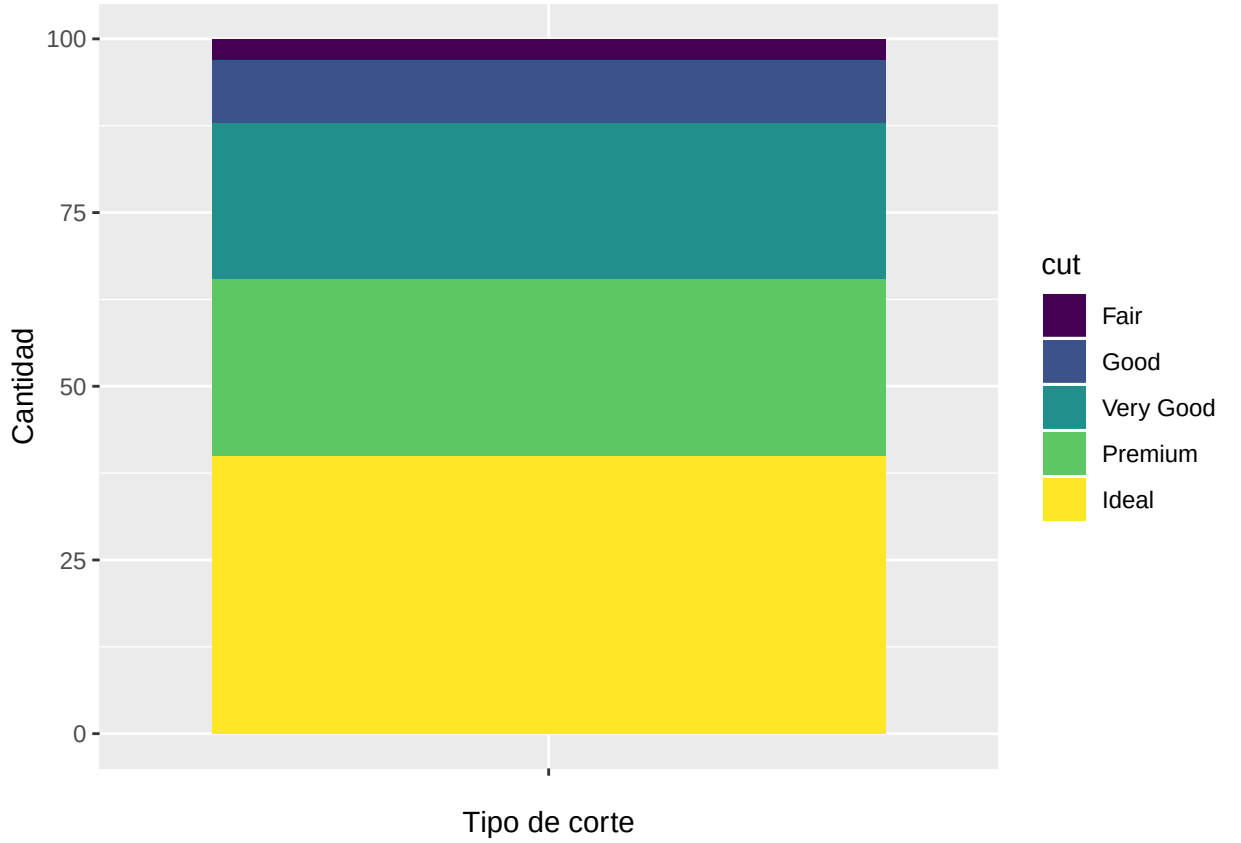
```r
grafica + coord_polar("y") +  # Gráfica de tarta con porcentaje y esquema de color
  geom_bar(aes(y = (..count..)/sum(..count..)*100)) +
  scale_fill_brewer(palette="Blues") +  theme_minimal()
```



```r
grafica + geom_bar()    # Barra apilada básica
```

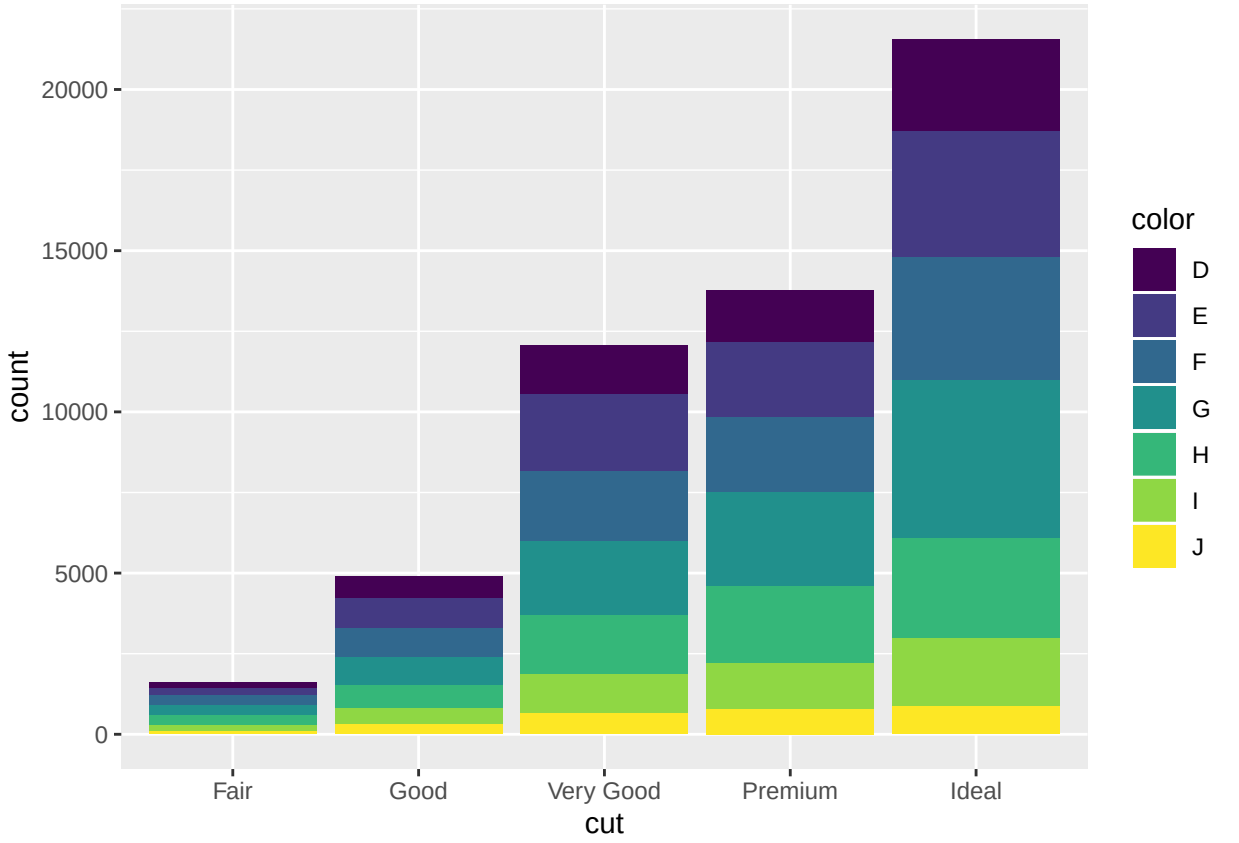


```r
grafica + geom_bar(aes(y = (..count..)/sum(..count..)*100))    # Barra con porcentaje en lugar de cuenta
```

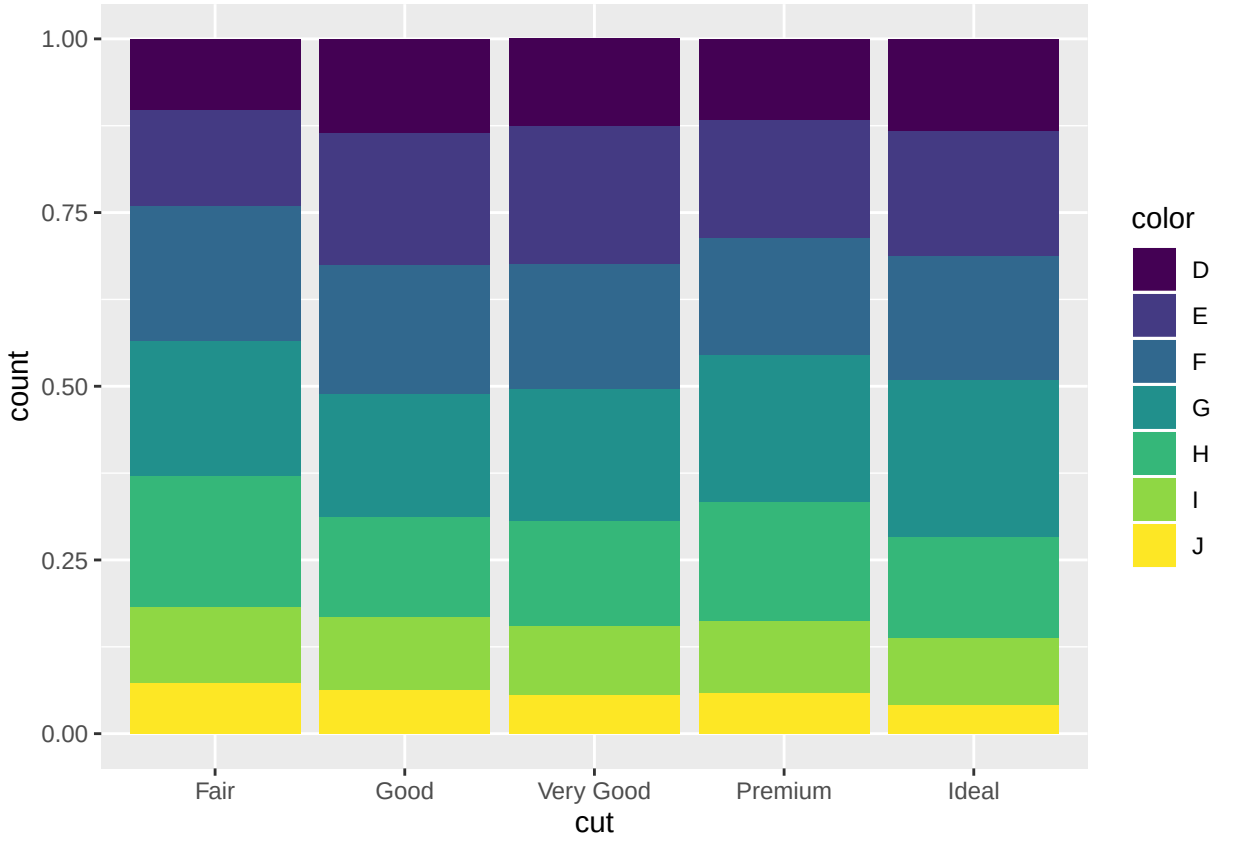


Esta técnica puede combinarse con la representación de múltiples series de datos, cada una de ellas como una barra apilada:

```
ggplot(diamonds, aes(cut)) + geom_bar(aes(fill=color))
```

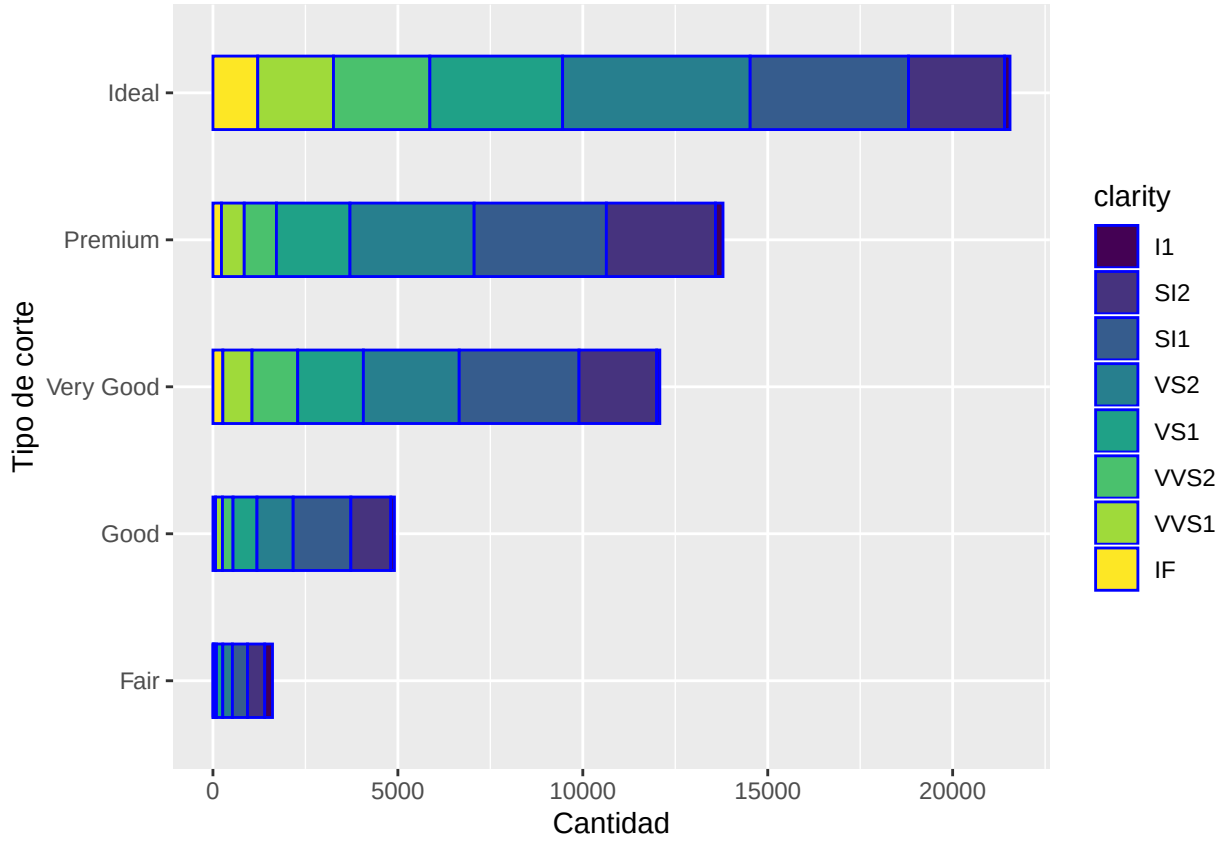


```
ggplot(diamonds, aes(cut)) + geom_bar(aes(fill=color), position = "fill")
```



La apariencia de la gráfica puede personalizarse con los atributos enumerados anteriormente, cambiando la orientación, colores, ancho de las barras, etc.

```
ggplot(diamonds, aes(cut)) +
  geom_bar(aes(fill=clarity), colour="blue", width=0.5) +
  coord_flip() + labs(x = "Tipo de corte", y = "Cantidad")
```



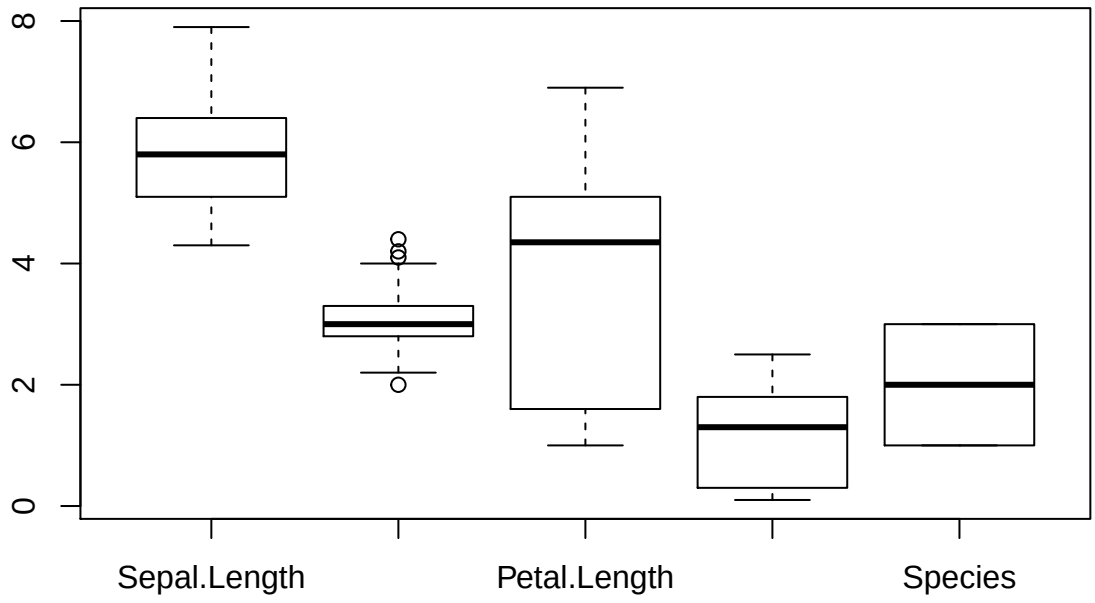
Análisis de la distribución de los datos

En ocasiones trabajaremos con una o más series de datos y lo que nos interesará no será hacer comparaciones o analizar la evolución, sino estudiar cómo se distribuyen esas muestras y **comparar las distribuciones**. Con este fin recurriremos habitualmente a dos tipos de representaciones: los diagramas de cajas y bigotes y los histogramas.

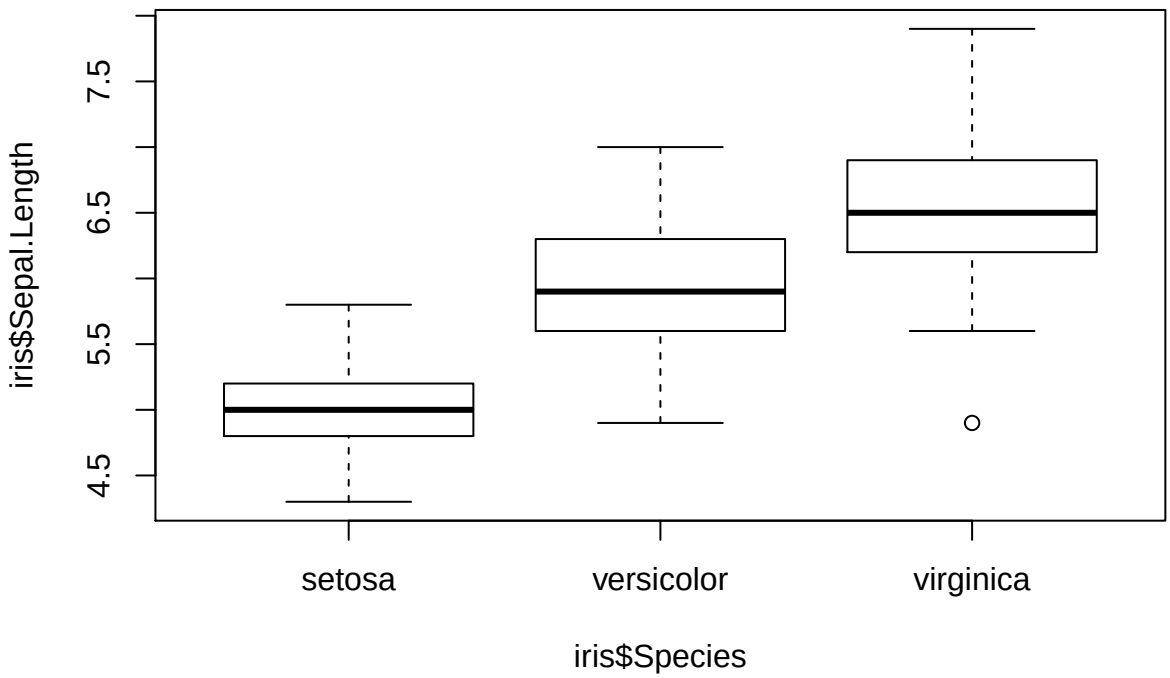
Diagrama de cajas y bigotes (*boxplot*)

Sirven para obtener los estadísticos esenciales de la distribución de un conjunto de valores. Podemos generar este tipo representación con el comando `boxplot()` o bien con `ggplot()`, como otros tipos de gráficas.

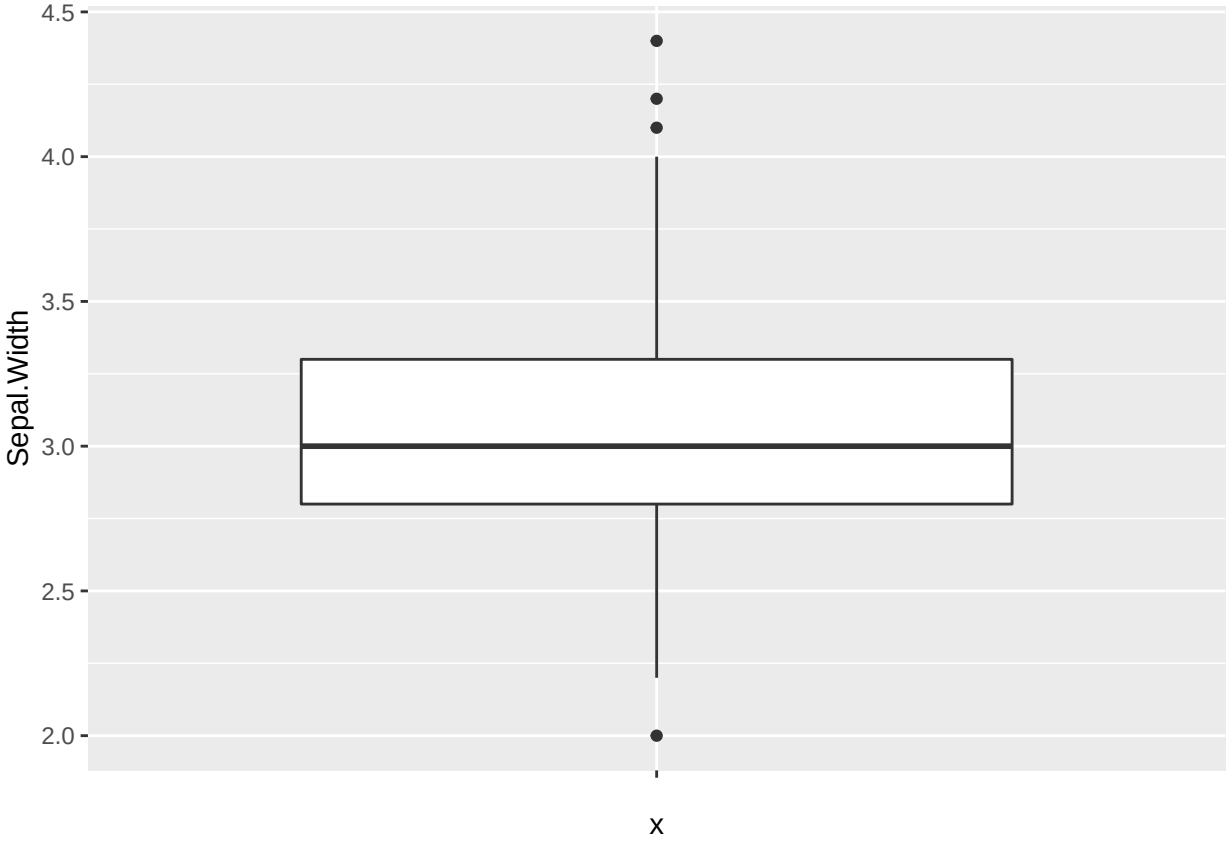
```
boxplot(iris) # Distribución de cada una de las variables
```



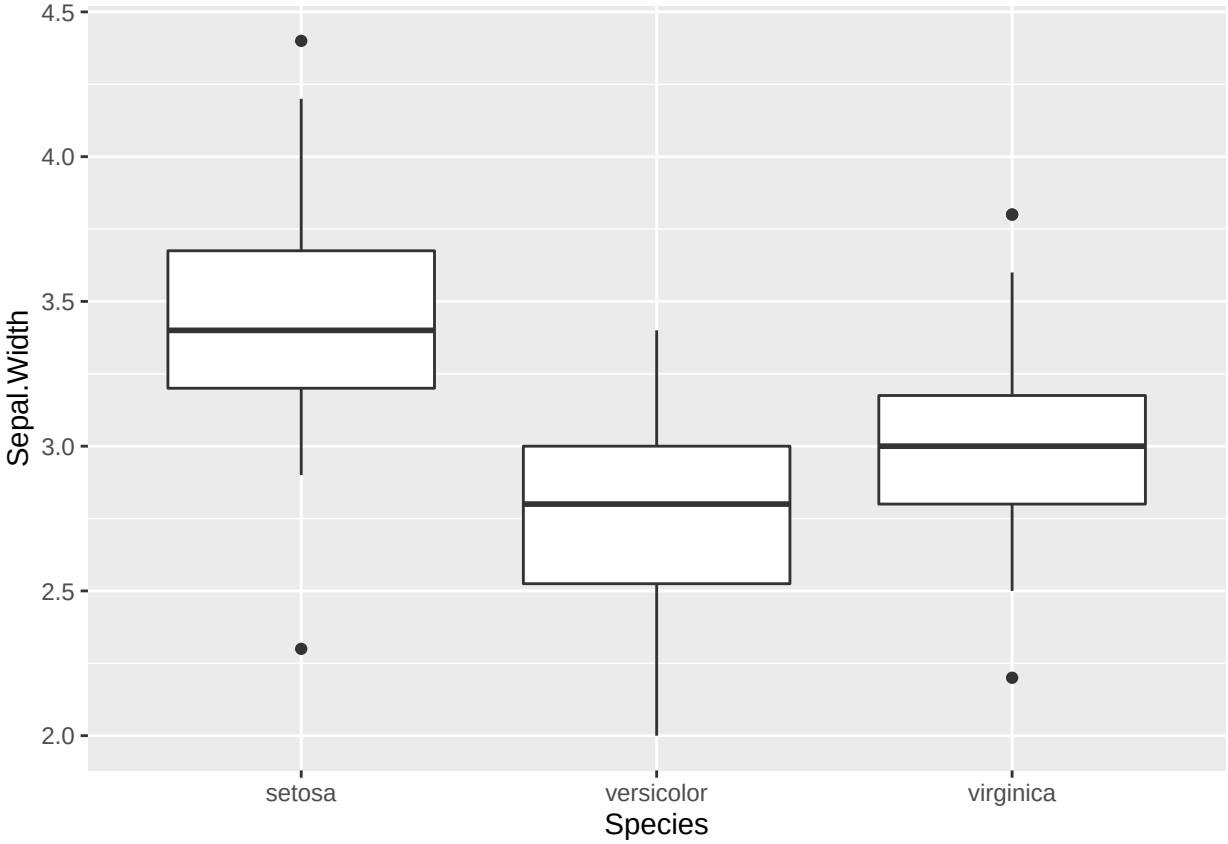
```r
boxplot(iris$Sepal.Length ~ iris$Species) # Distribución de una variable agrupada según los valores de
```



```
ggplot(iris, aes("", Sepal.Width)) + geom_boxplot()
```

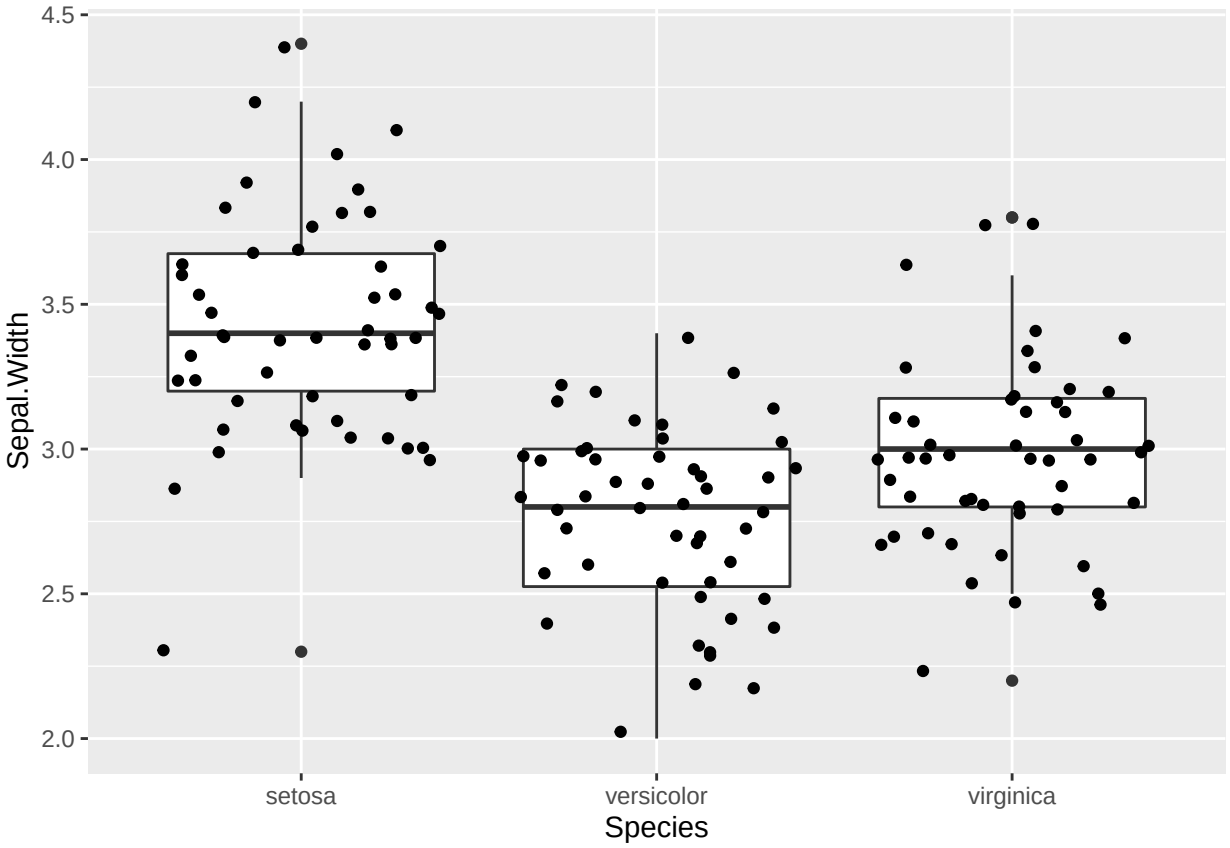


```
ggplot(iris, aes(Species, Sepal.Width)) + geom_boxplot()
```



Podemos superponer a los diagramas de cajas y bigotes la posición que ocupe cada integrante de la muestra:

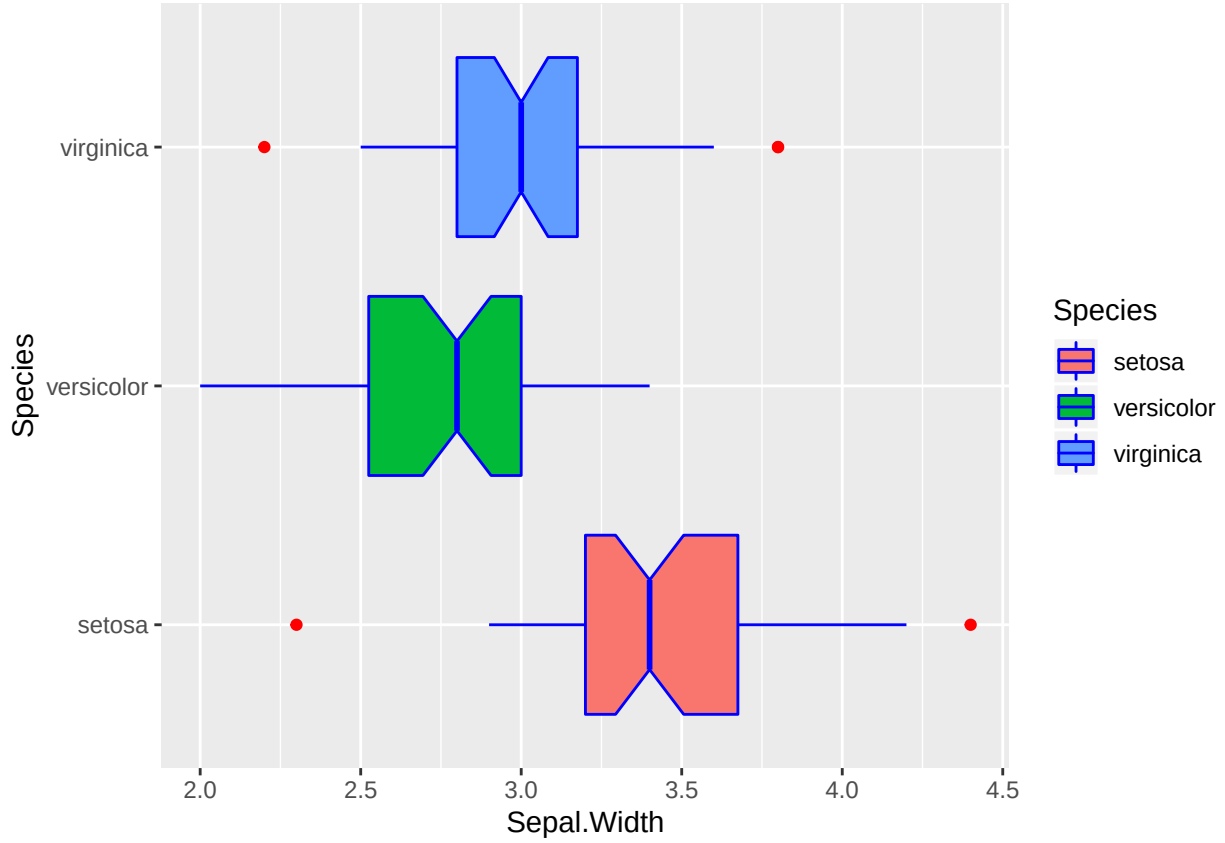
```
ggplot(iris, aes(Species, Sepal.Width)) + geom_boxplot() + geom_jitter()
```



Ajustes visuales de un diagrama de cajas y bigotes

- Podemos personalizar la apariencia de algunos de los elementos de este tipo de gráfica con los siguientes parámetros:
  - `color` y `fill`: establecen el color de borde y relleno de las cajas
  - `outlier.colour` y `outlier.size`: establecen el color y tamaño de los casos extremos (*outliers*)
  - `coord_flip()`: al igual que con las gráficas de barras, podemos girar los ejes de coordenadas
  - `notch`: por defecto es `FALSE`. Asignándole `TRUE` se dibujará el intervalo de confianza de la mediana

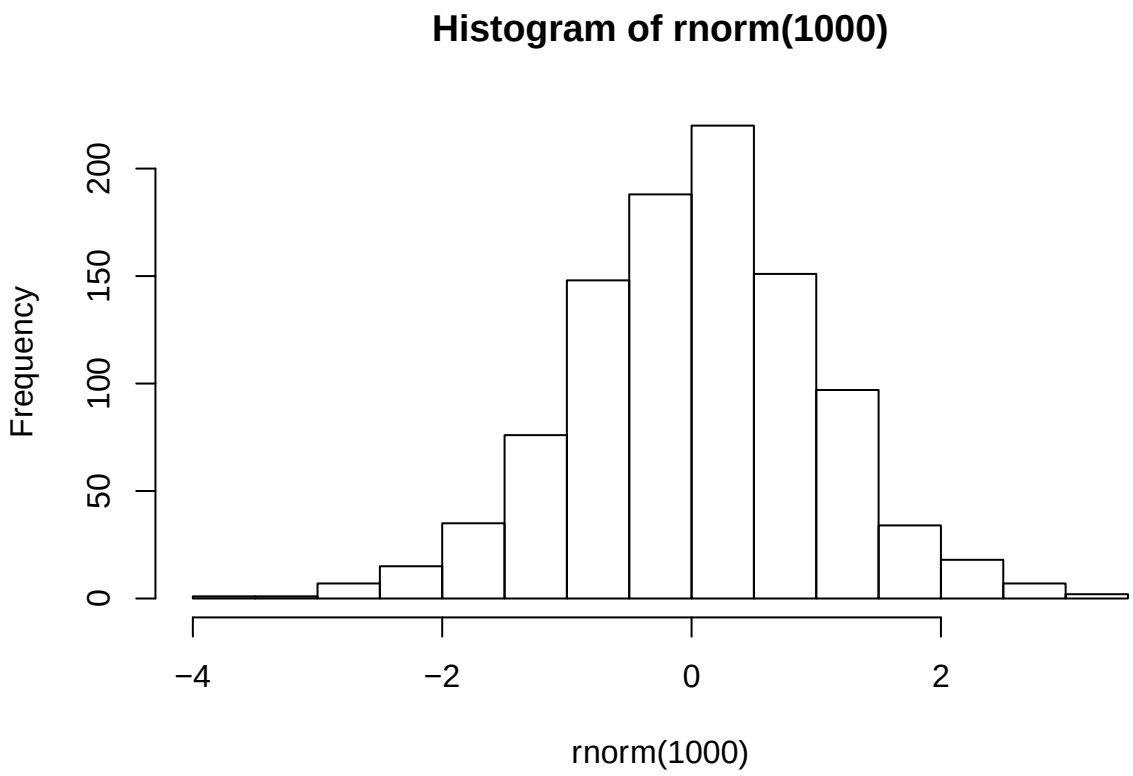
```
ggplot(iris, aes(Species, Sepal.Width)) +
  geom_boxplot(aes(fill=Species), color = "blue", outlier.colour = "red", notch = TRUE) +
  coord_flip()
```



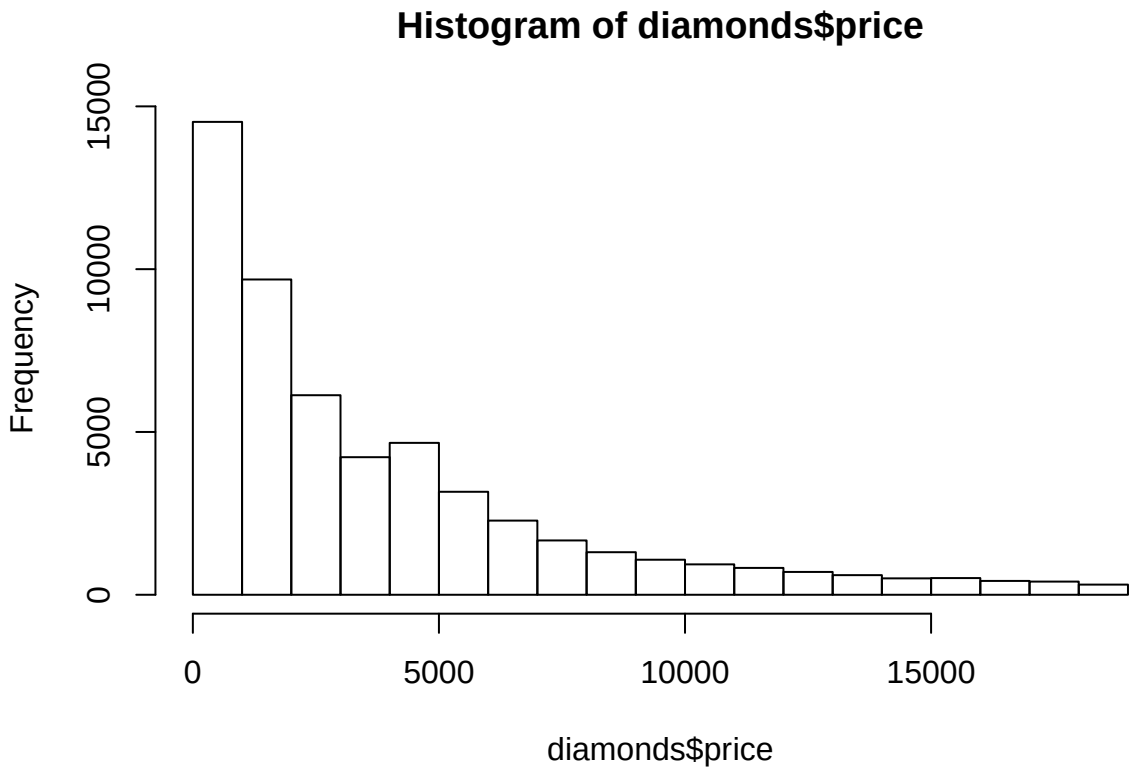
Histogramas

Se utilizan para **agrupar los valores** de una muestra de datos **en intervalos** y representar la frecuencia de cada intervalo, facilitando así el análisis de la distribución subyacente.

```r
hist(rnorm(1000))  # Histograma de una distribución normal
```

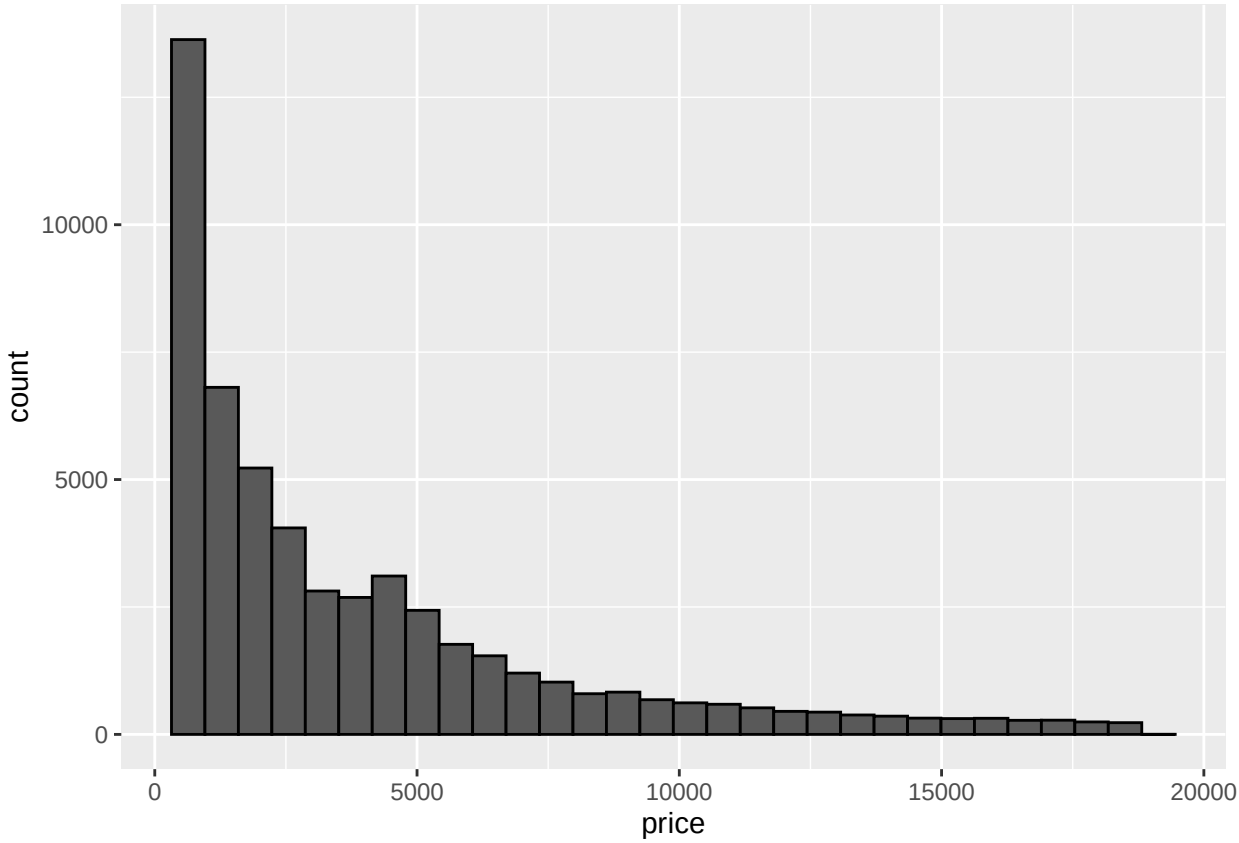


```r
hist(diamonds$price) # Histograma del precio de los diamantes
```



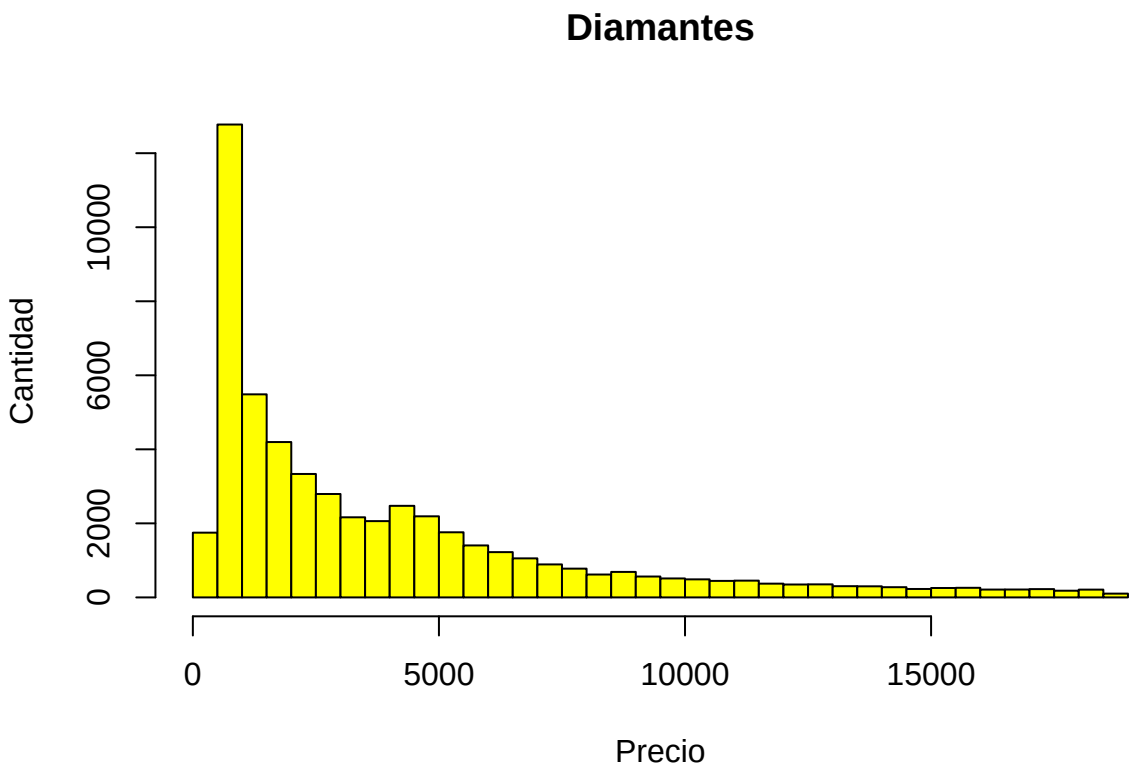
```r
ggplot(diamonds, aes(price)) + geom_histogram(color="black") # Alternativa con ggplot
```

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

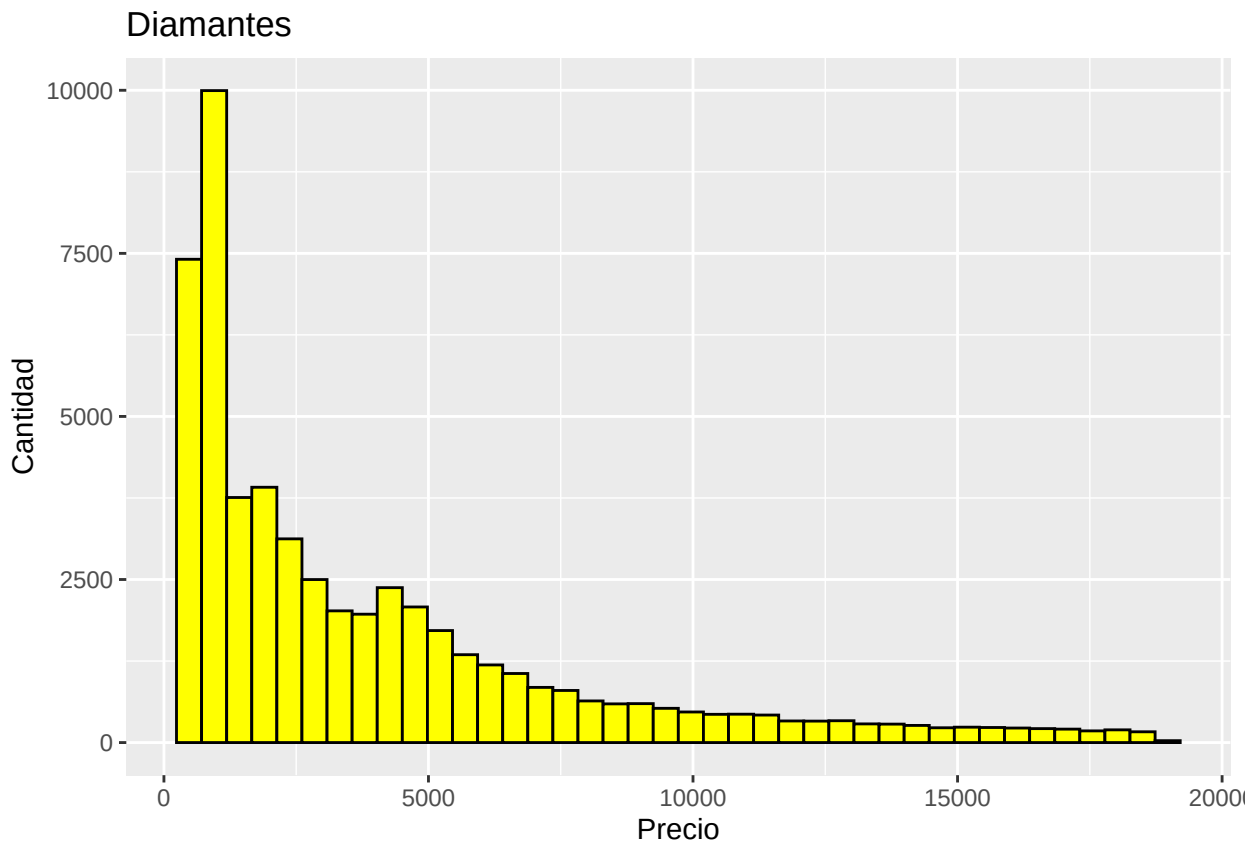


El número de divisiones (*bins*) en que se agrupan los valores es configurable. Usaremos `breaks` en el caso del comando `hist()` y `bins` con ggplot. También podemos establecer títulos de los ejes y color de las barras:

```
hist(diamonds$price, breaks=40,
     xlab="Precio", ylab = "Cantidad", main="Diamantes", col = "yellow")
```

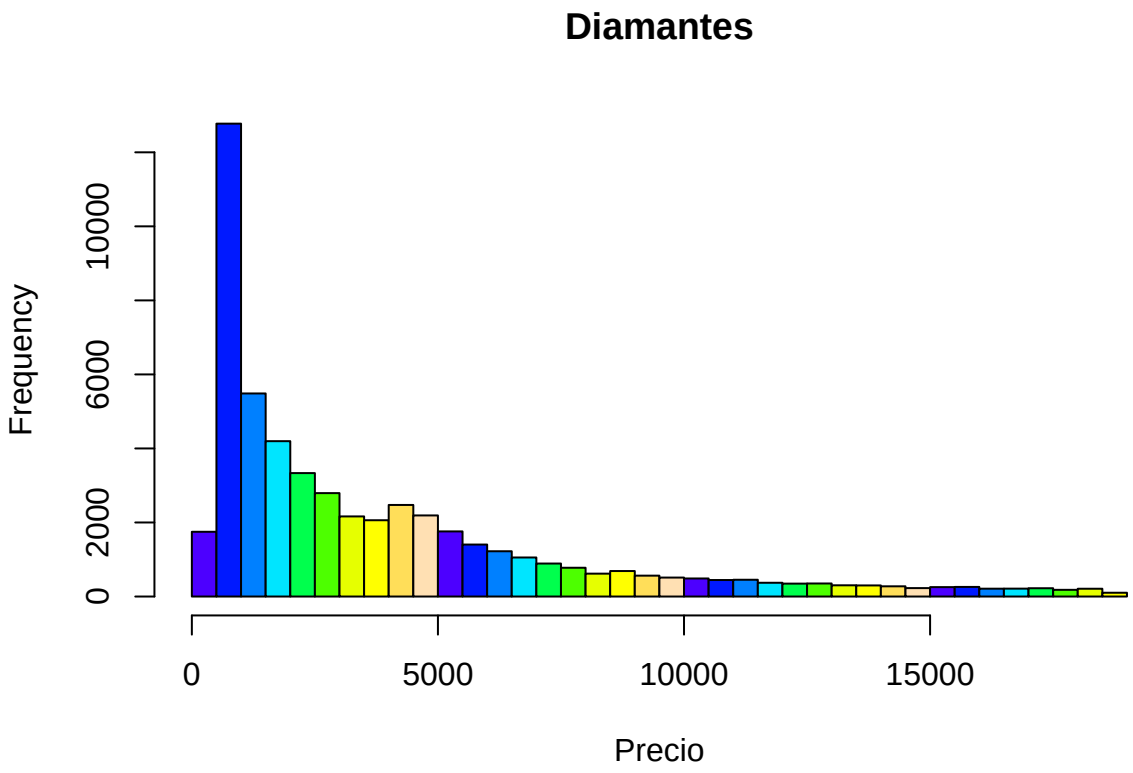


```
ggplot(diamonds, aes(price)) +
  geom_histogram(bins = 40, color="black", fill="yellow") +
  labs(title = "Diamantes", x = "Precio", y = "Cantidad")
```

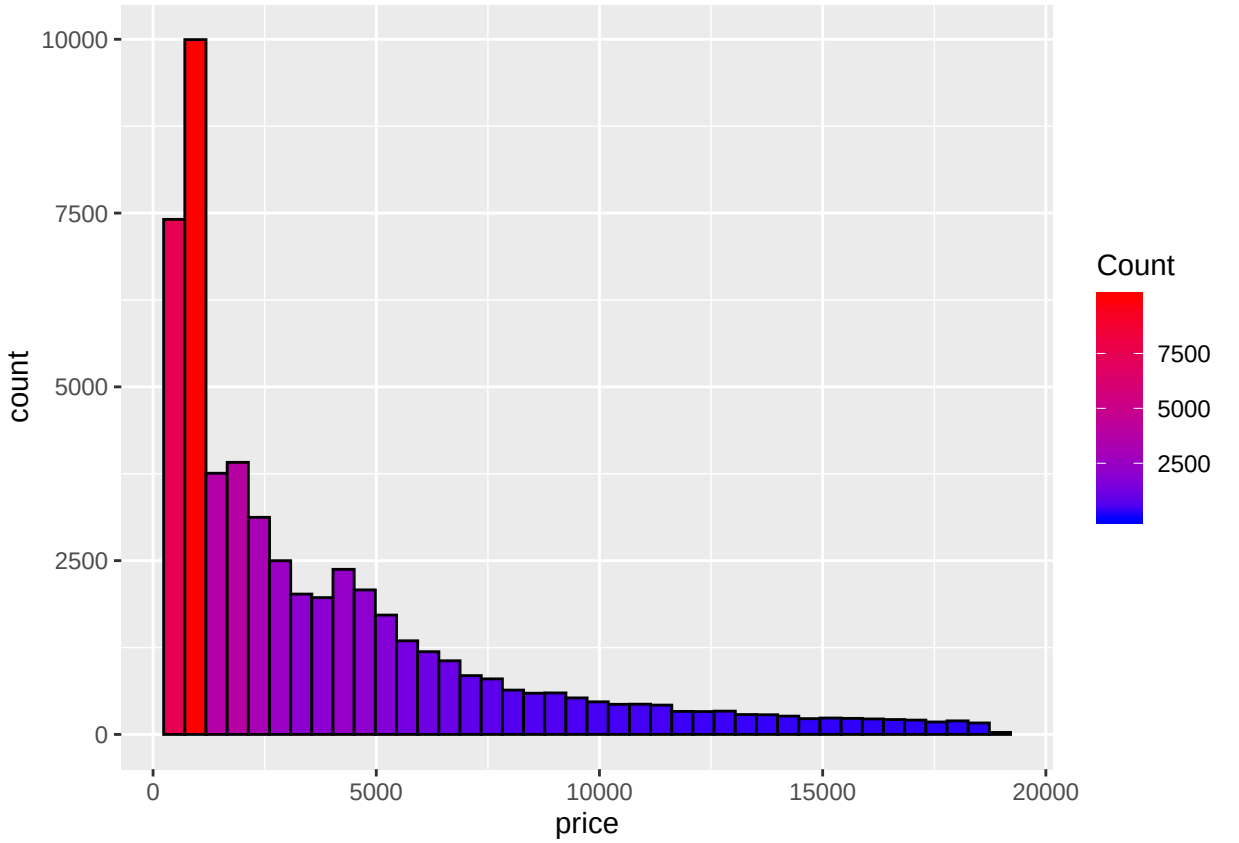


Es posible ajustar el color de las barras de forma individual, por ejemplo usando una gama de colores concreta, estableciendo el color en función del valor representado o incluso estableciendo criterios a medida:

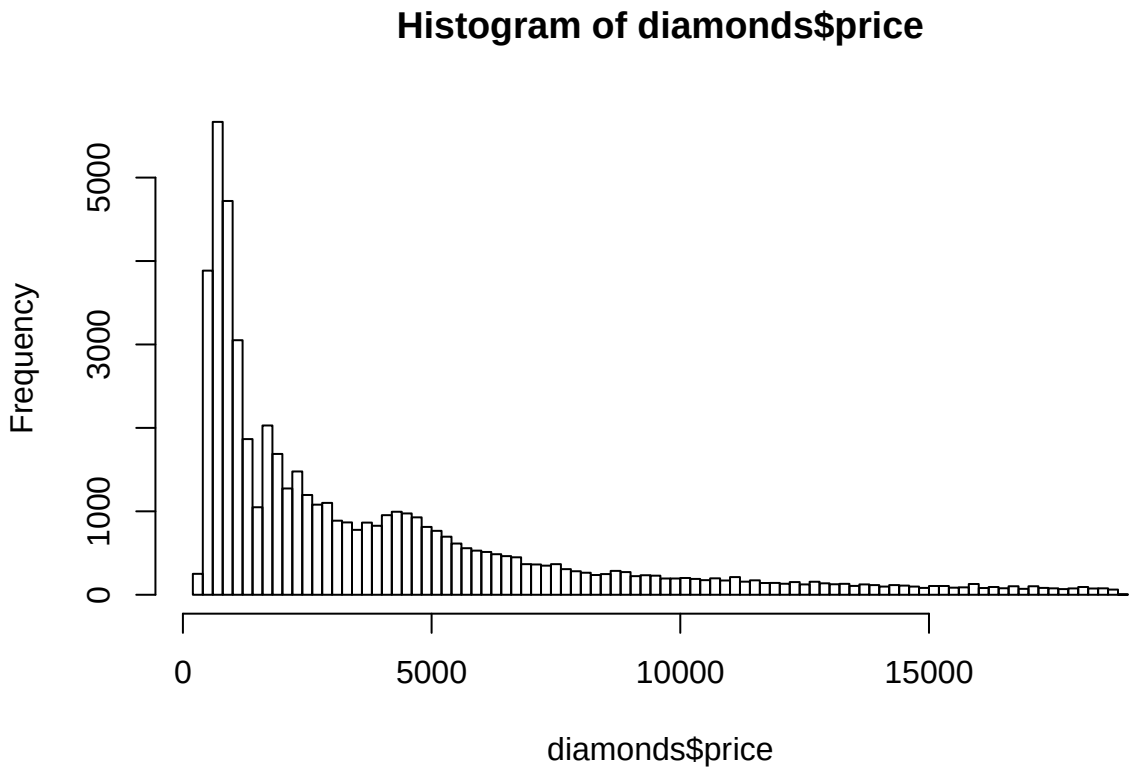
```
hist(diamonds$price, breaks=40, col=topo.colors(10), main="Diamantes", xlab="Precio")
```



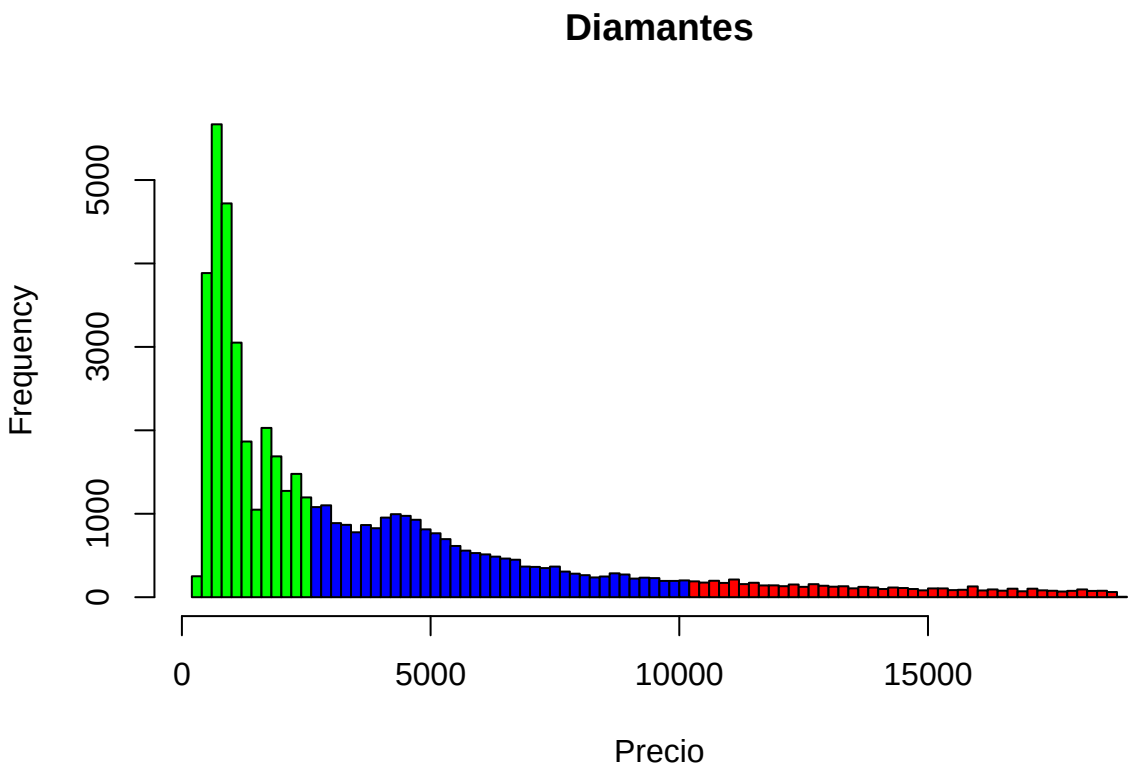
```
ggplot(diamonds, aes(price)) +
  geom_histogram(bins = 40, color="black", aes(fill=..count..)) +
  scale_fill_gradient("Count", low = "blue", high = "red")
```



```r
histograma <- hist(diamonds$price, breaks = 100)
```



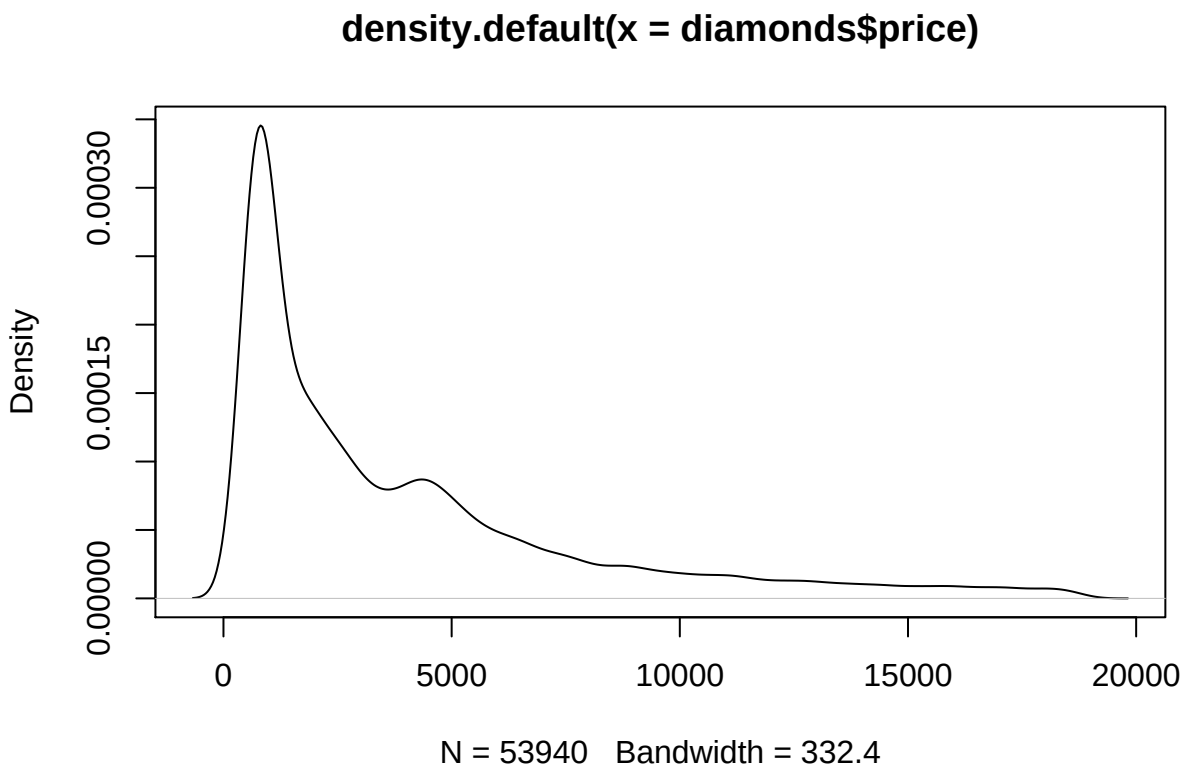
```r
plot(histograma, col =
       ifelse(histograma$breaks < 2500, 'green',
              ifelse(histograma$breaks > 10000,  "red", "blue")),
     main = 'Diamantes', xlab = 'Precio')
```



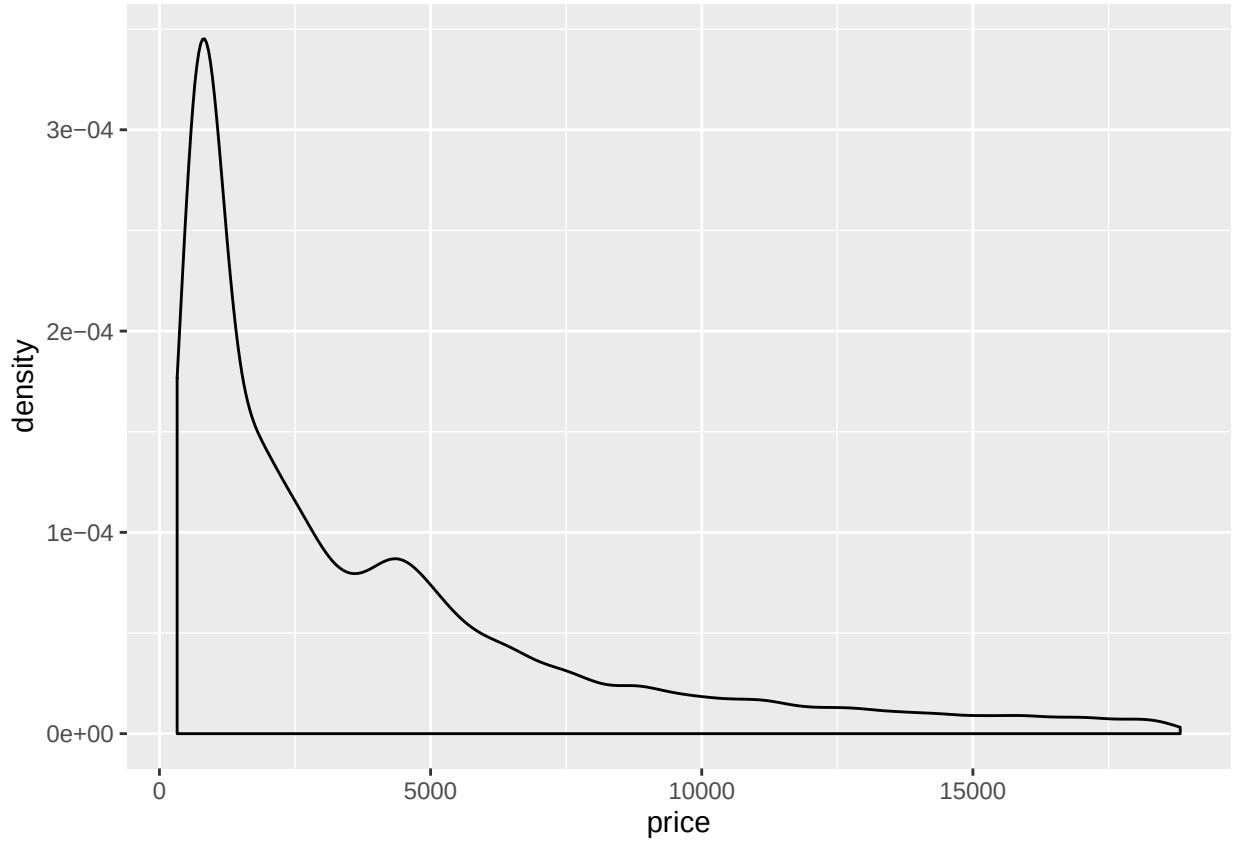
Curva de densidad

Una representación alternativa de la distribución de un conjunto de valores es la curva de densidad. En esta no se dibujan barras, como en el caso del histograma, sino una línea continua que denota la densidad en cada punto de la escala X:

```
plot(density(diamonds$price))
```



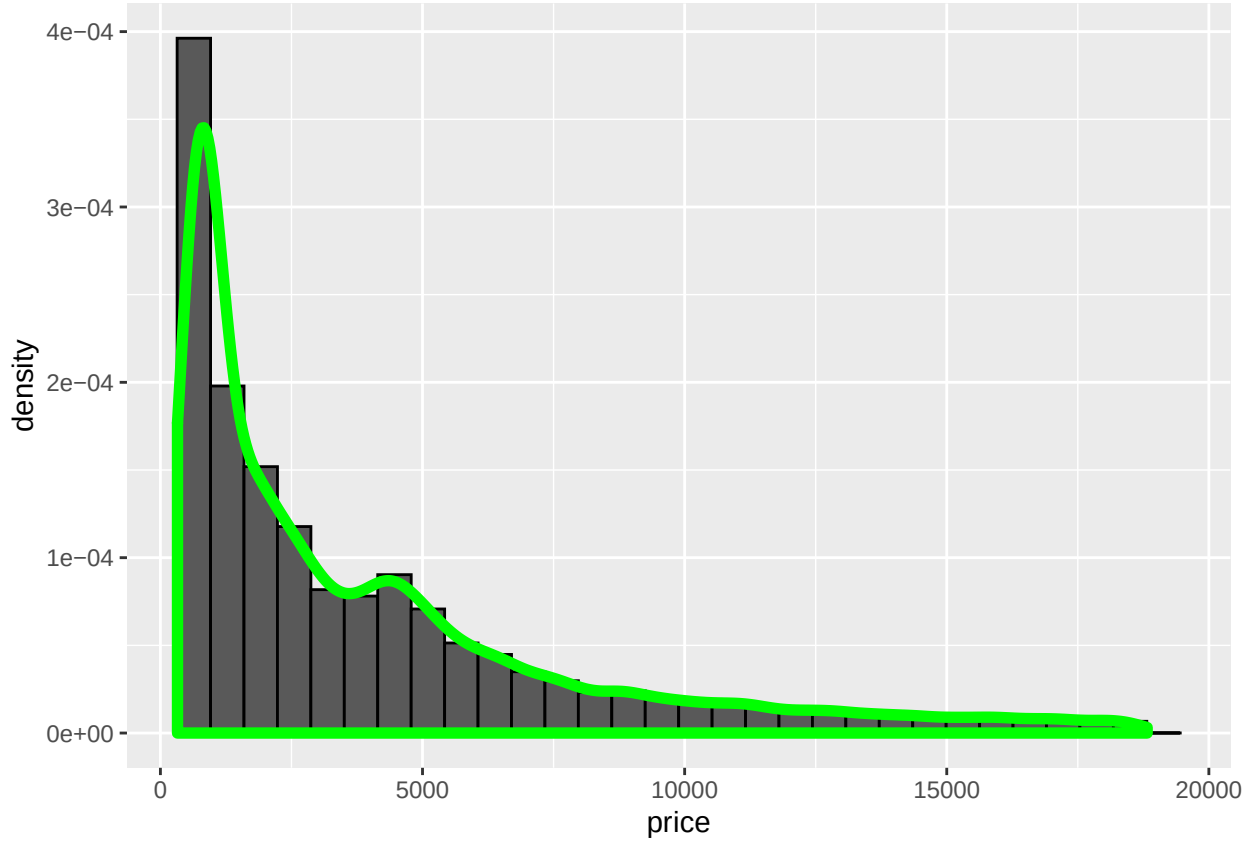
```
ggplot(diamonds, aes(price)) + geom_density()
```



Podemos superponer la curva de densidad al histograma:

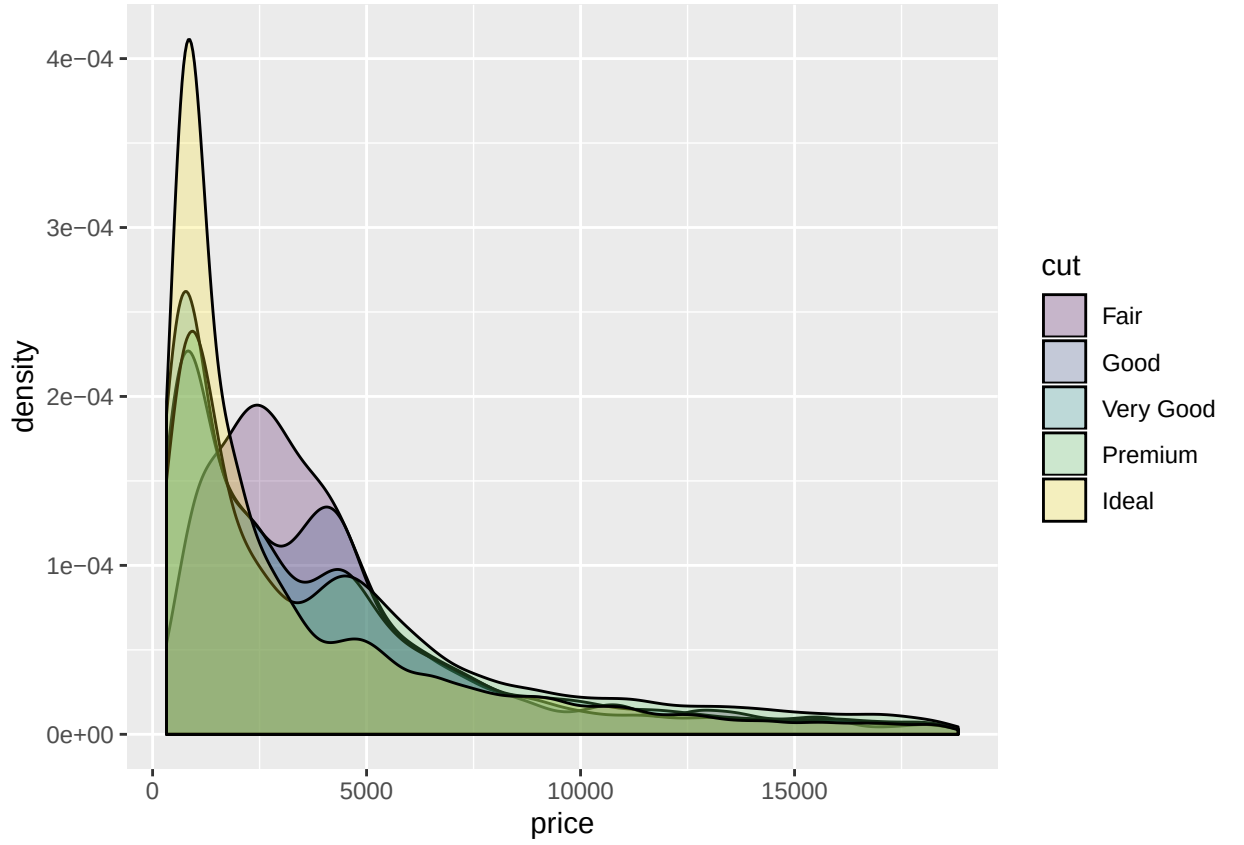
```
ggplot(diamonds, aes(price)) +
  geom_histogram(color="black", aes(y=..density..))  +
  geom_density(color="green", size=2, alpha=0.5)
```

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



Este tipo de gráfica permite representar simultáneamente la distribución de varias variables, por ejemplo agrupando los precios según el tipo de corte:

```
ggplot(diamonds, aes(price, fill=cut)) +
  geom_density(alpha=0.25)
```



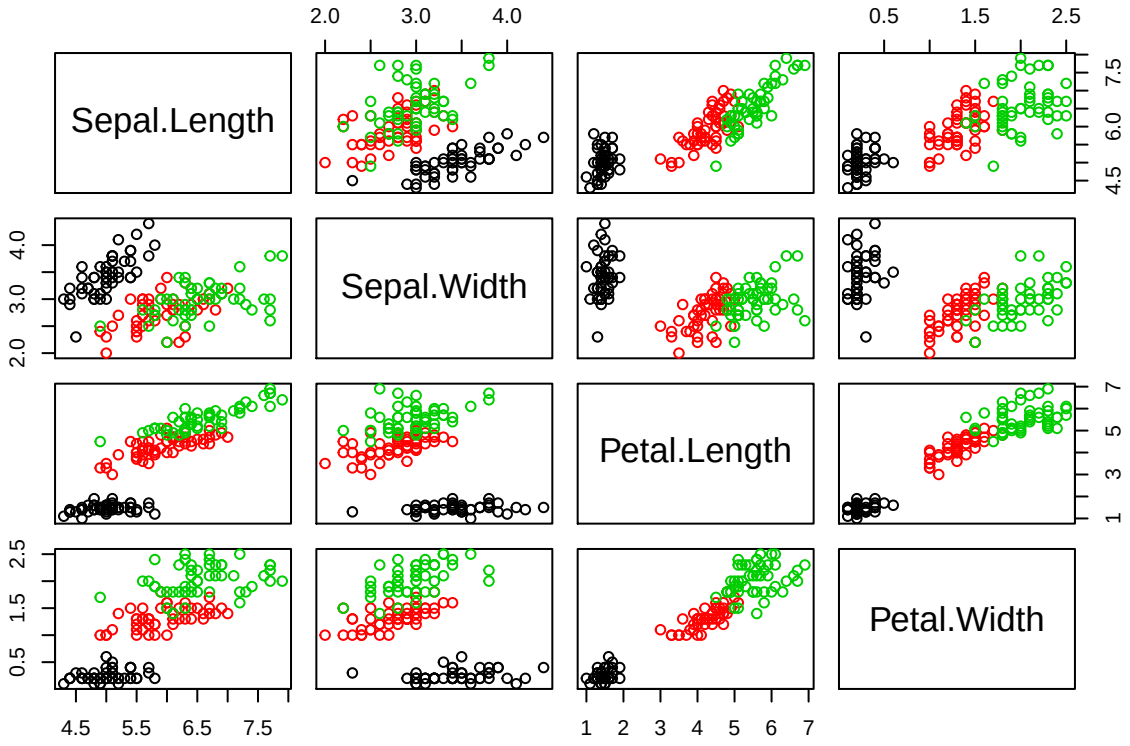
Gráficas avanzadas

Las posibilidades gráficas en R son muy extensas, habiendo disponibles **varias decenas de paquetes** que, como `ggplot`, ofrecen un extenso conjunto de comandos para elaborarlas. A continuación aprenderemos a usar algunas de ellas.

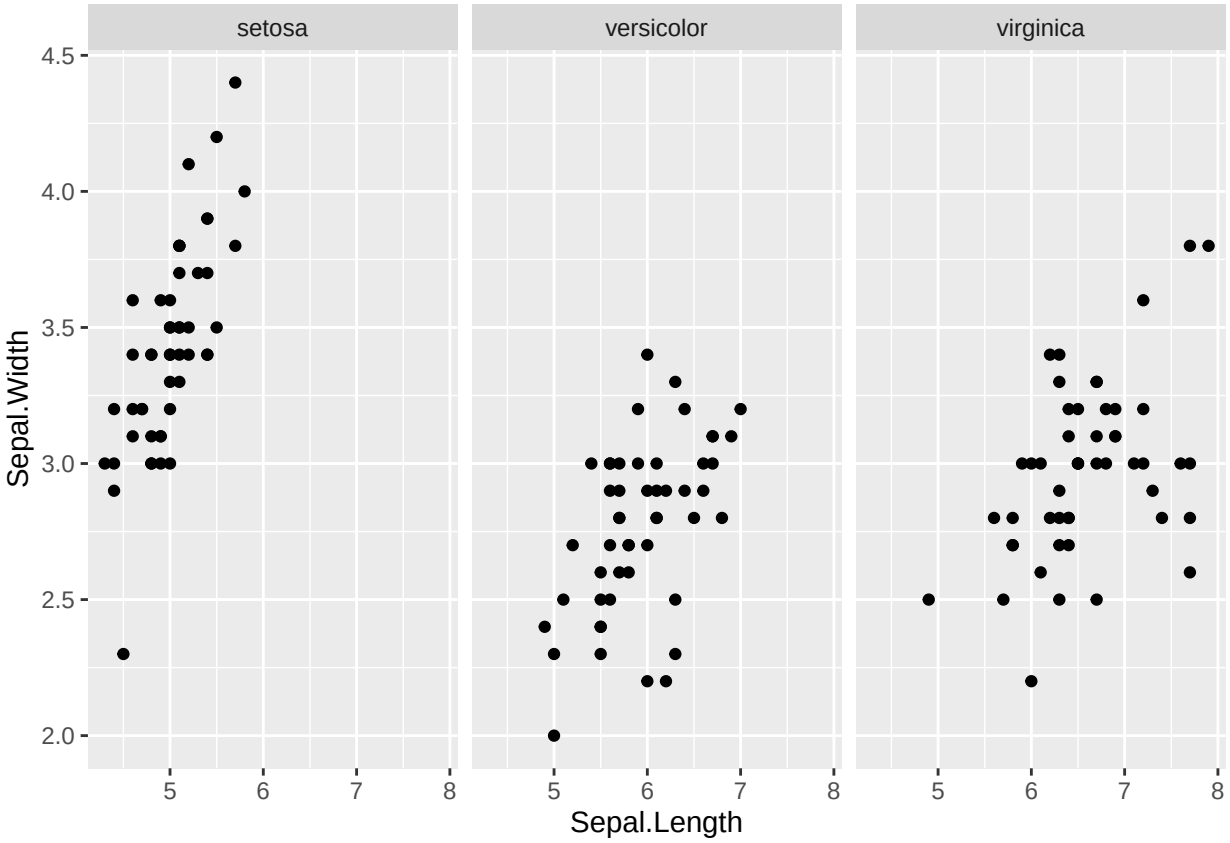
Combinación de múltiples gráficas

La posibilidad de combinar en una misma imagen múltiples gráficas facilita la comparación visual rápida, por ejemplo entre distintas variables.

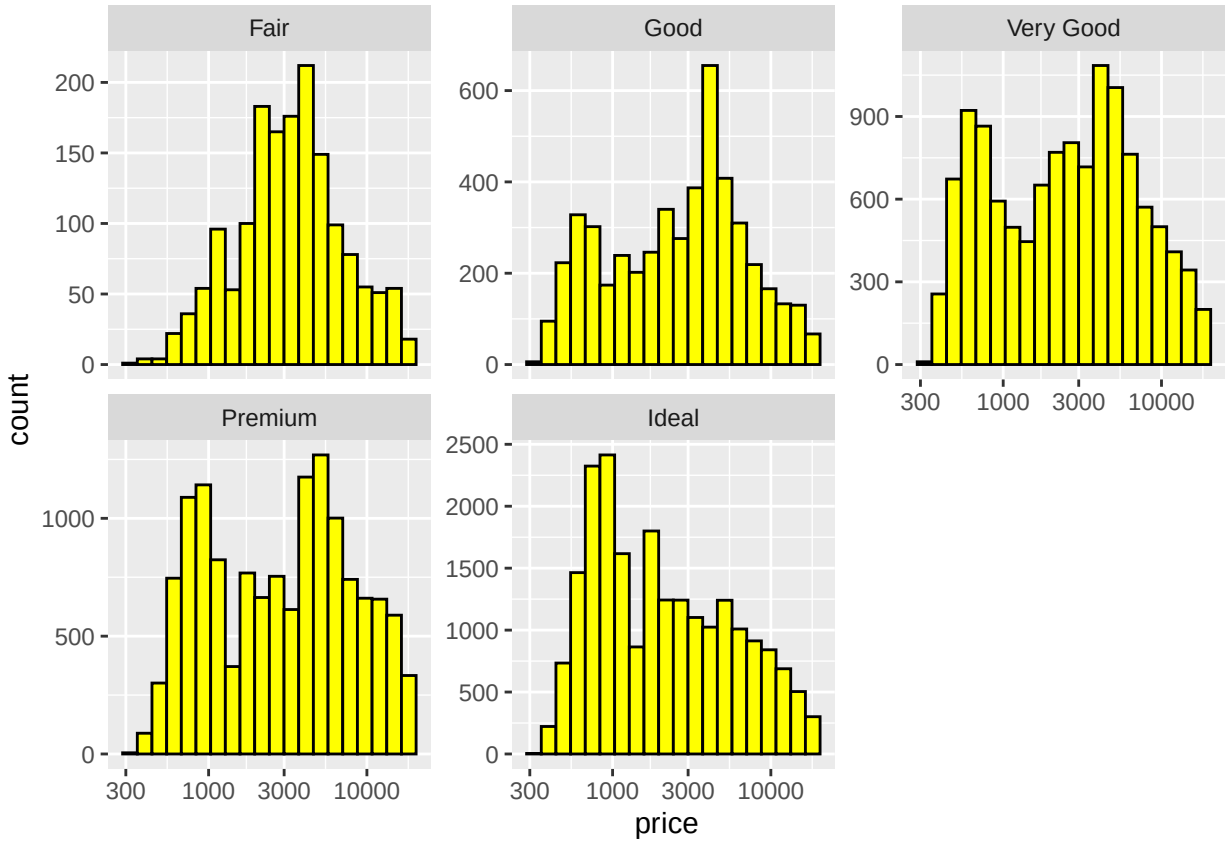
```
plot(iris[ ,-5], col = iris$Species)
```



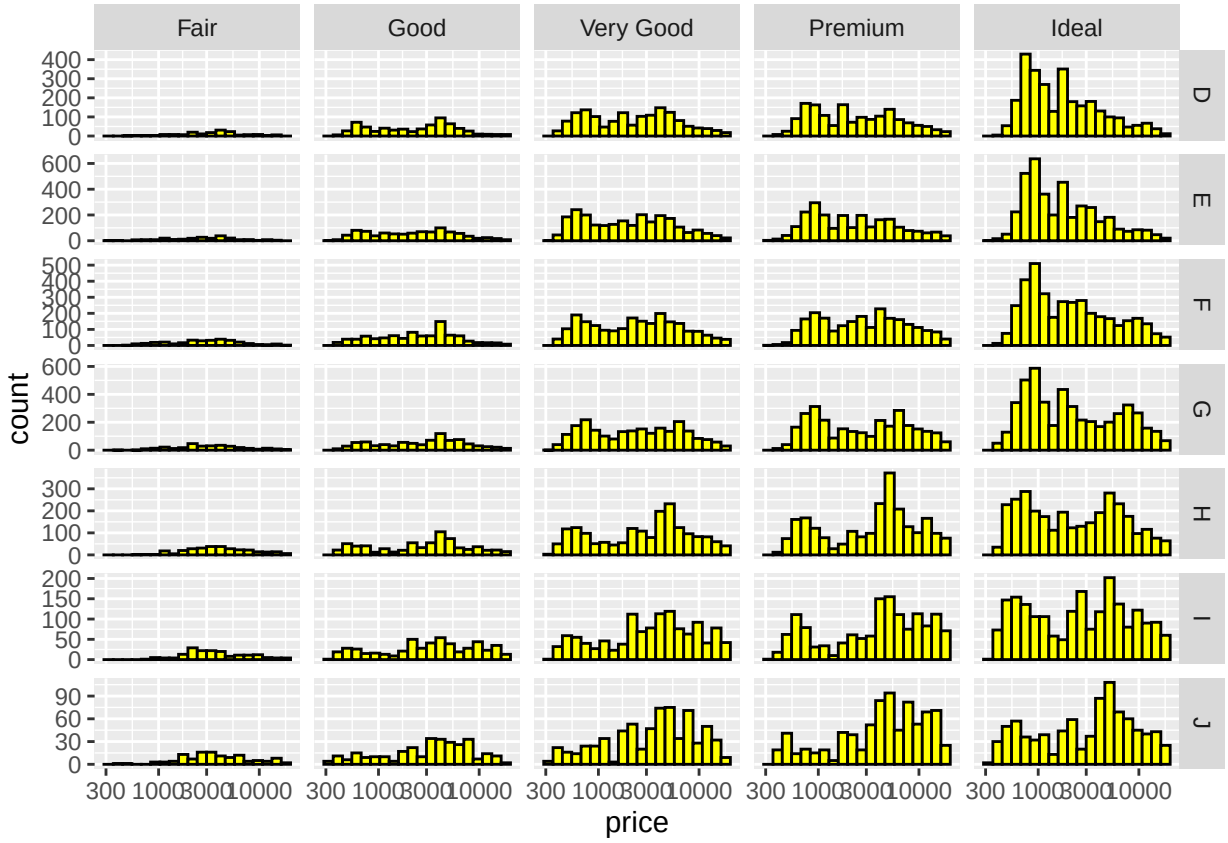
```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() + facet_wrap(~Species)
```



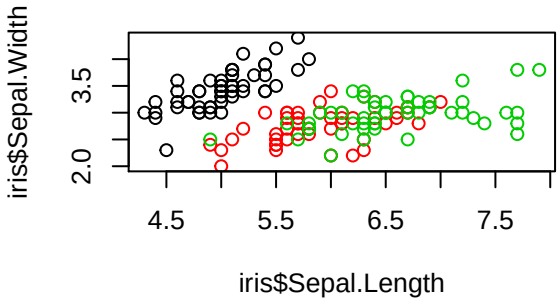
```
grafica <- ggplot(diamonds, aes(price)) +
  geom_histogram(bins=20, color="black", fill="yellow") +
  scale_x_log10()

grafica + facet_wrap(~cut, scales = "free_y")
```

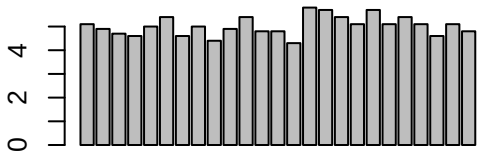


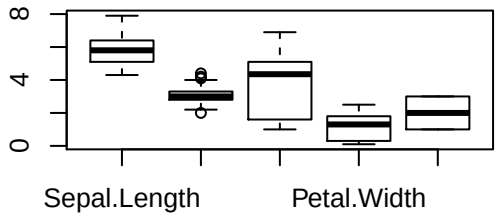
```
grafica + facet_grid(color~cut, scales = "free_y")
```

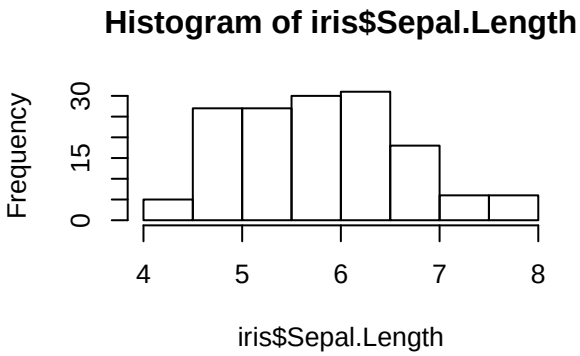


```r
par(mfrow=c(2,2))

plot(iris$Sepal.Length, iris$Sepal.Width, col=iris$Species)
barplot(iris$Sepal.Length[1:25])
boxplot(iris)
hist(iris$Sepal.Length)
```









Representación de relaciones

Las gráficas circulares conocidas como tipo *circos* facilitan la visualización de porcentajes y relaciones entre variables:

```
library(circlize) # Cargamos el paquete necesario para esta gráfica
```

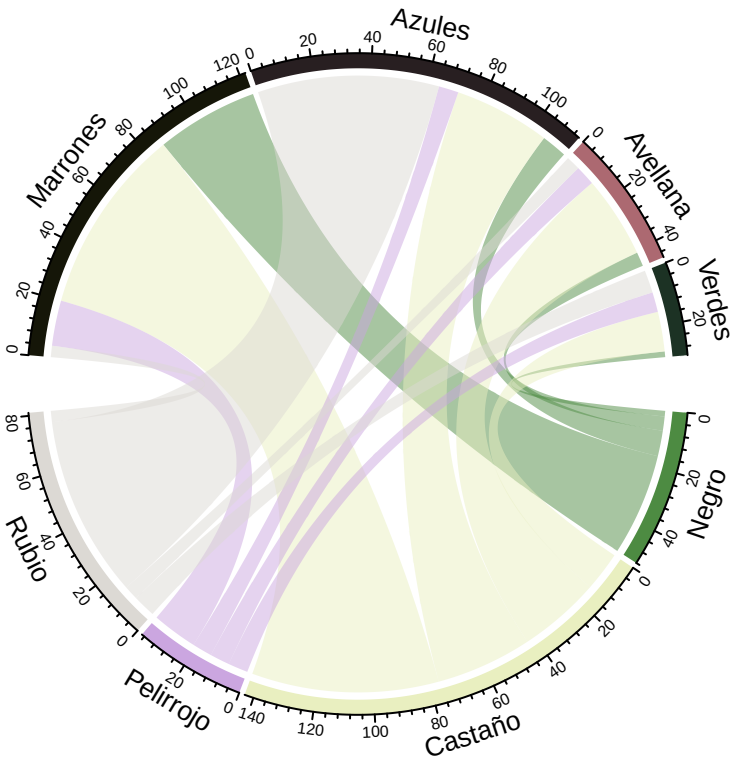
```
## Warning: package 'circlize' was built under R version 3.6.3
```

```
## ========================================
## circlize version 0.4.8
## CRAN page: https://cran.r-project.org/package=circlize
## Github page: https://github.com/jokergoo/circlize
## Documentation: http://jokergoo.github.io/circlize_book/book/
##
## If you use it in published research, please cite:
## Gu, Z. circlize implements and enhances circular visualization
##   in R. Bioinformatics 2014.
## ========================================
```

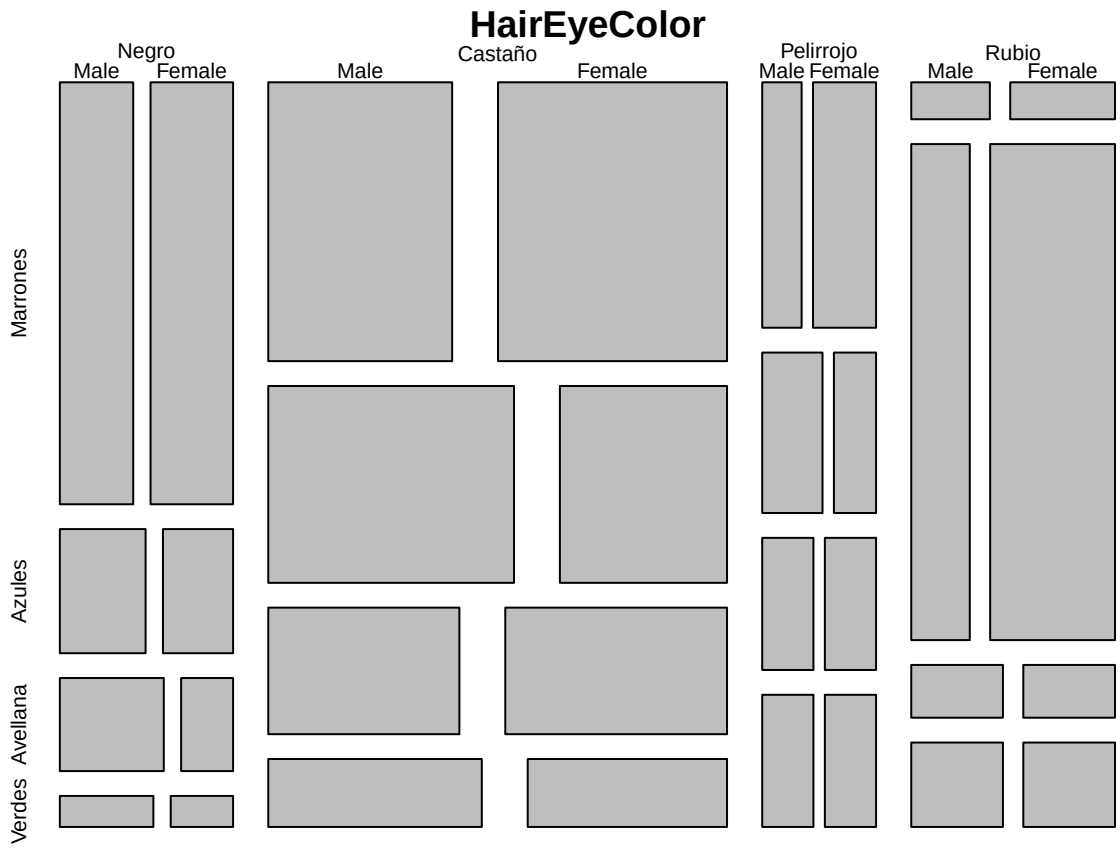
```
HairEyeColor # Examinamos el dataset HairEyeColor
```

```
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

```
# Podemos traducir las denominaciones
attr(HairEyeColor,"dimnames")$Eye <- c("Marrones", "Azules", "Avellana", "Verdes")
attr(HairEyeColor,"dimnames")$Hair <- c("Negro", "Castaño", "Pelirrojo", "Rubio")

chordDiagram(HairEyeColor[,,2])
```


Azules
Avellana
Verdes
Negro
Castaño
Pelirrojo
Rubio
Marrones

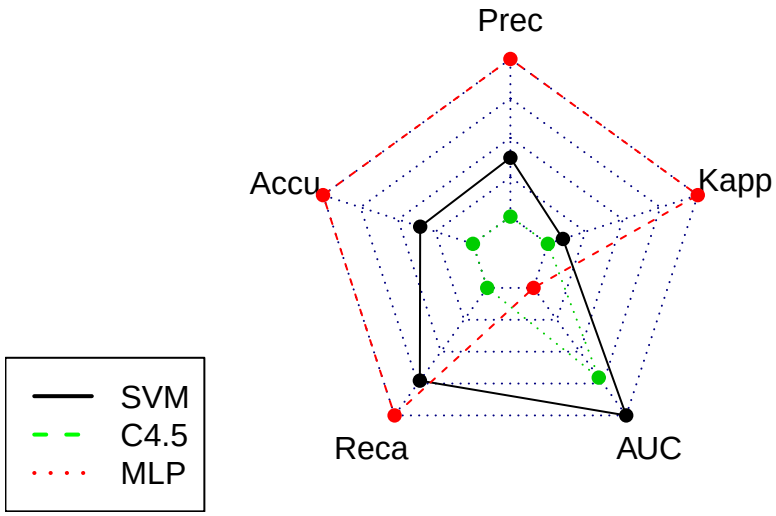
```
mosaicplot(HairEyeColor)
```



Comparación de diferentes alternativas atendiendo a varios criterios

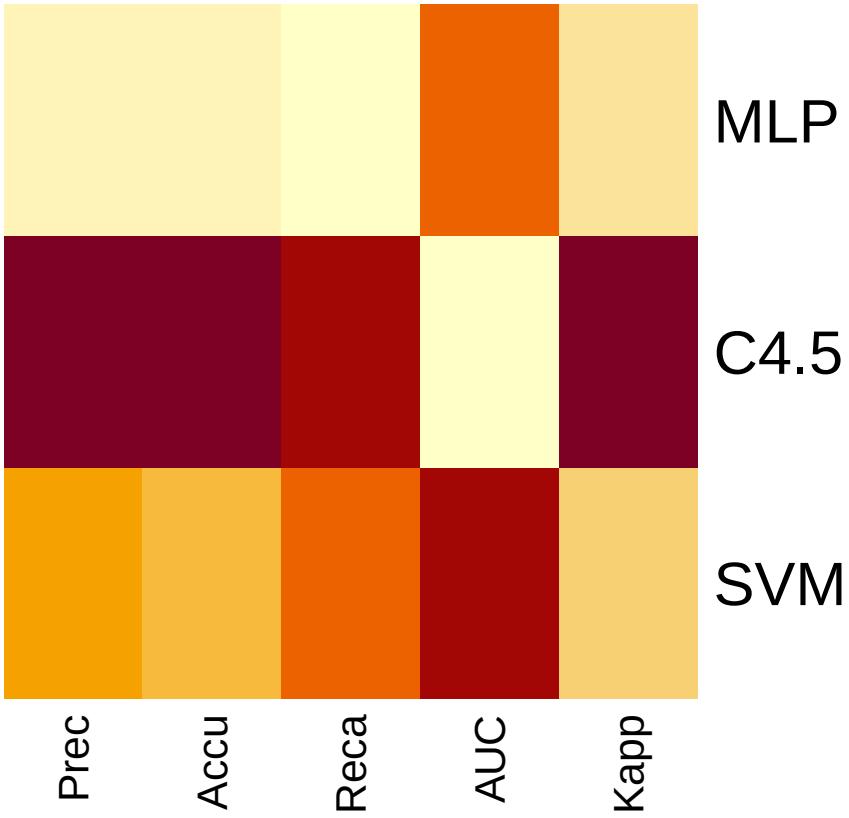
En ocasiones se cuenta con diferentes vías para solucionar un mismo problema y la calidad de cada una de ellas puede evaluarse con distintos criterios, lo cual dificulta el análisis y selección de la mejor opción. La gráfica de tipo *radar* o *spider* está pensada para estos casos.

```
library('fmsb')  # Cargamos el paquete donde está el comando

dat <- data.frame(    # Simulamos tres métodos con cinco criterios de evaluación
  Prec = runif(3, 0, 1),
  Accu  = runif(3, 0, 1),
  Reca    = runif(3, 0, 1),
  AUC       = runif(3, 0, 1),
  Kapp     = runif(3, 0, 1)
  )
rownames(dat) <- c("SVM", "C4.5", "MLP")

radarchart(dat, maxmin = FALSE) # Dibujamos la gráfica

# Agregamos una leyenda para indicar los criterios
legend("bottomleft", rownames(dat),
       col = c('black', 'green', 'red'),
       lty = 1:3, lwd = 2, ncol = 1)
```



Alternativamente, también podría utilizarse una gráfica de tipo *heatmap*. Esta es una matriz en la que se representan todos los posibles valores, asignando un color a cada celdilla en función del valor correspondiente:

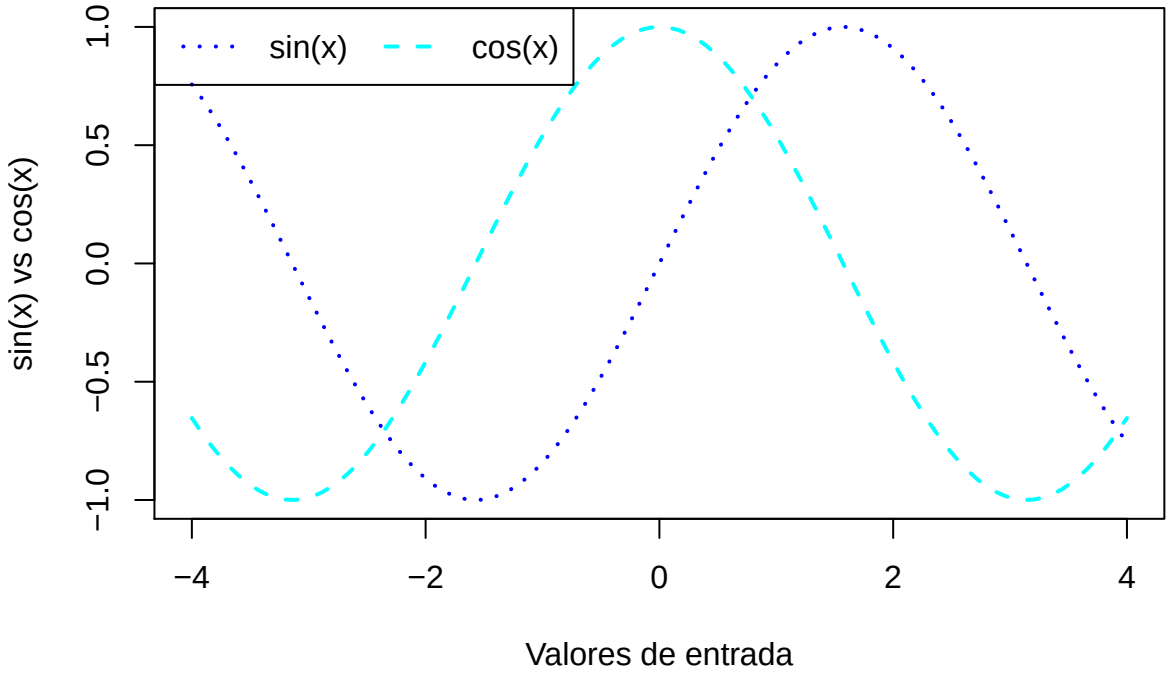
```
heatmap(as.matrix(dat), scale="column", Colv=NA, Rowv=NA)
```



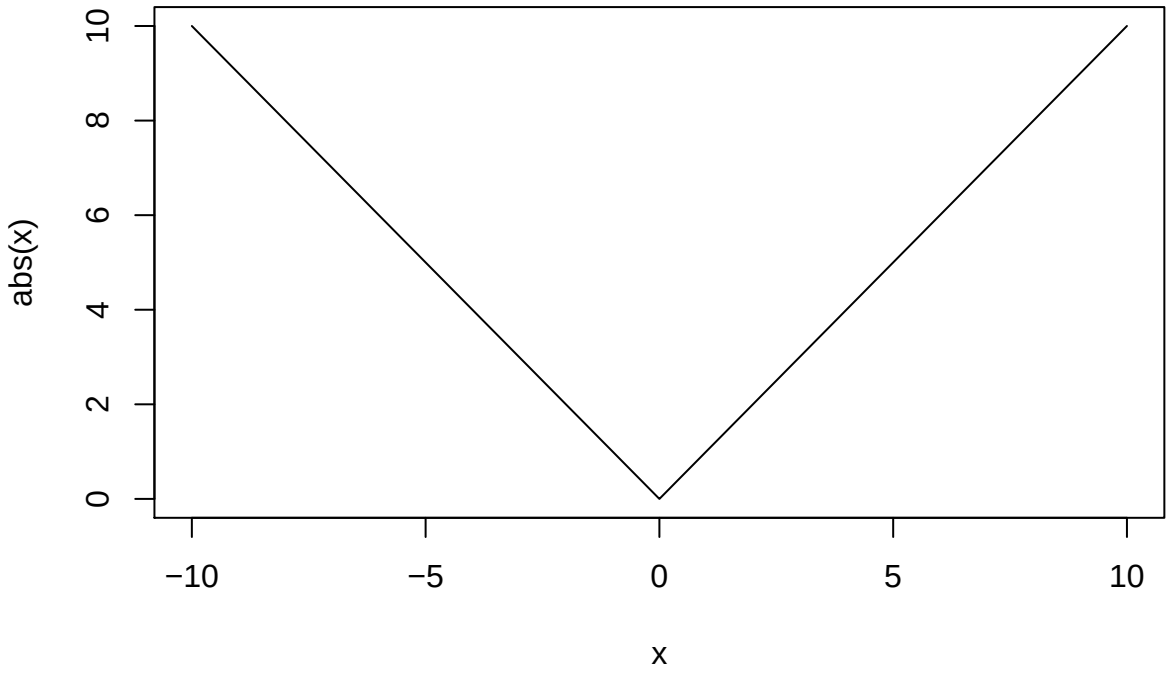
Representación gráfica de funciones

R es capaz de representar cualquier función matemática, ya sea una función incluida en R o cualquier otra que podamos necesitar siempre que la definamos de forma adecuada. Los siguientes son algunos ejemplos:

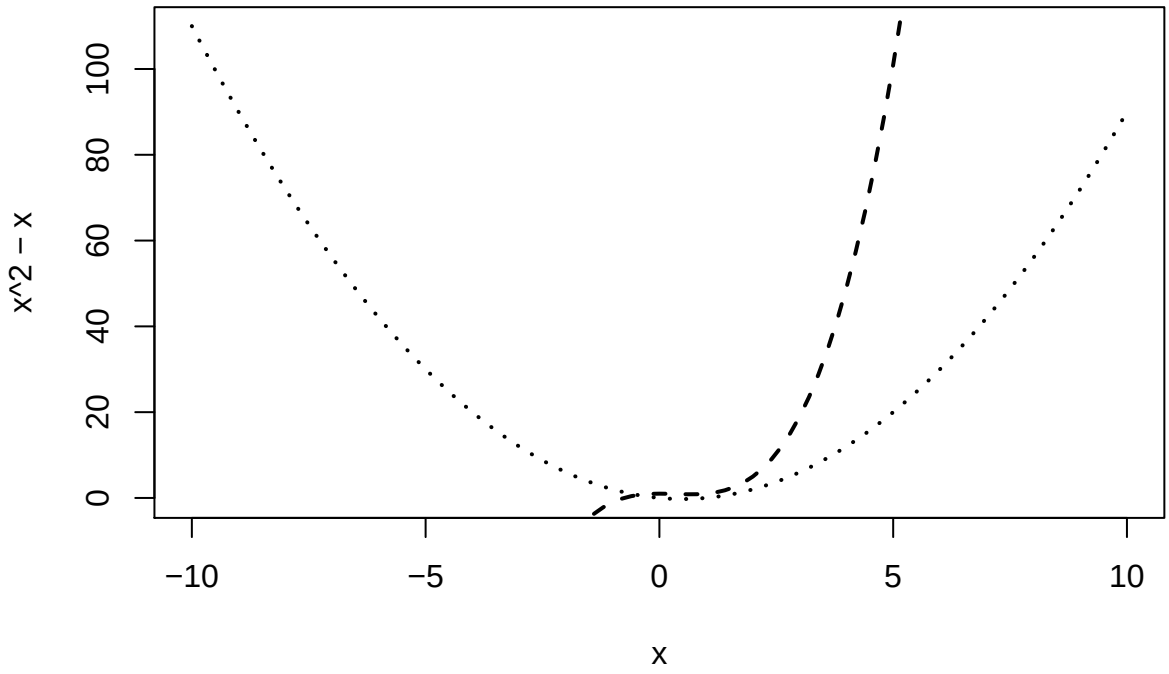
```
# Curva a partir de funciones existentes (sin y cos)
curve(sin, from = -4, to = 4, col = 'blue', lty = 'dotted', lwd = 2,
      ylab='sin(x) vs cos(x)', xname = "Valores de entrada")
curve(cos, from = -4, to = 4, col = 'cyan', lty = 'dashed', lwd = 2, add = TRUE)
legend("topleft", c("sin(x)", "cos(x)"), col = c('blue', 'cyan'),
       lty = c('dotted','dashed'), lwd = 2, ncol = 2)
```



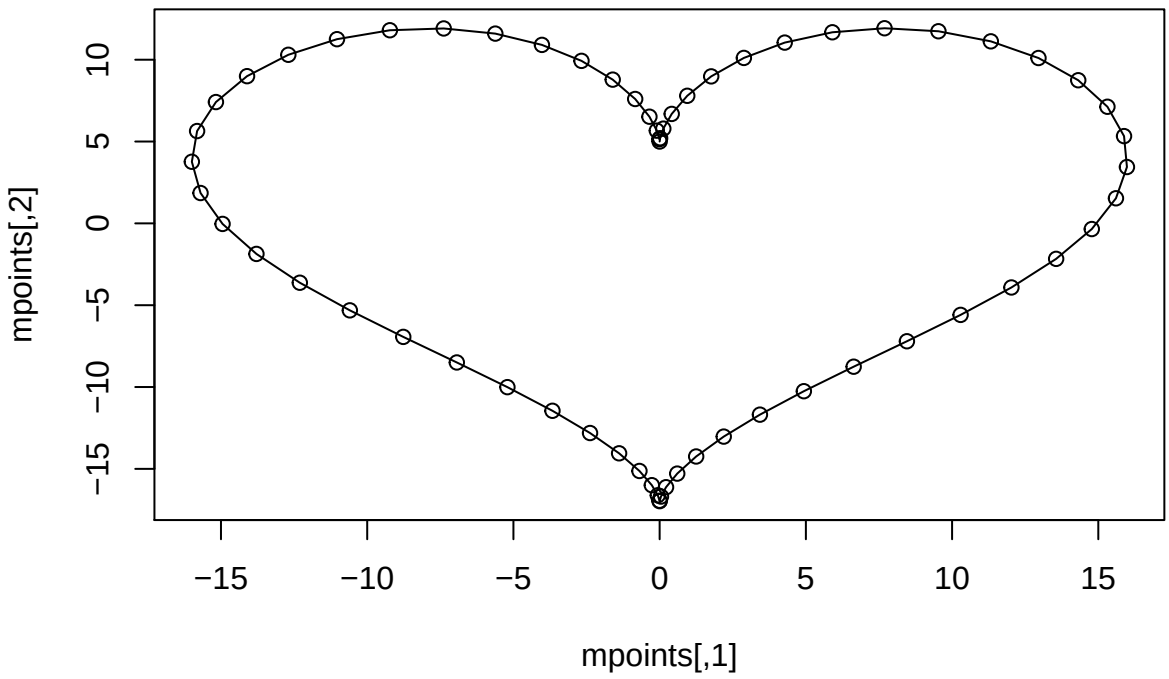
```r
# Curva del valor absoluto
curve(abs, from = -10, to = 10)
```



```r
# Curvas a partir de polinomios
curve(x^2 - x, lty = 3, lwd = 2, from = -10, to = 10)
curve(x^3 - x^2 + 1, lty = 2, lwd = 2, from = -10, to = 10, add = TRUE)
```



```r
# A partir de una función parámetrica definida por el usuario
heart <- function(t) {
  x <- 16 * sin(t)^3
  y <- 13 * cos(t) - 5 * cos(2*t) - 2 * cos(3*t) - cos(4*t)
  c(x,y)
}

mpoints <- matrix(sapply(seq(0,2*pi,0.1), heart), ncol = 2, byrow = TRUE)
plot(mpoints)
lines(mpoints)
```



Geometría con gráficos de tortuga

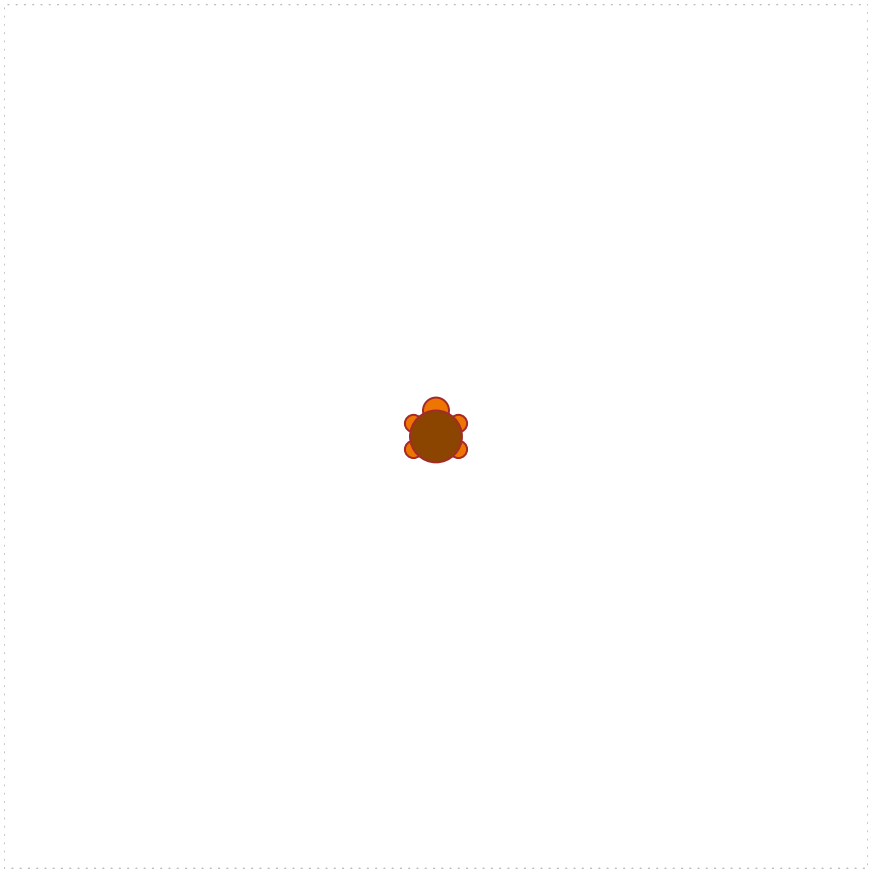
Incluso podemos generar gráficos con geometrías definidas algorítmicamente acorde con el lenguaje Logo, conocidos como gráficos de tortuga:

```
library('TurtleGraphics')
```

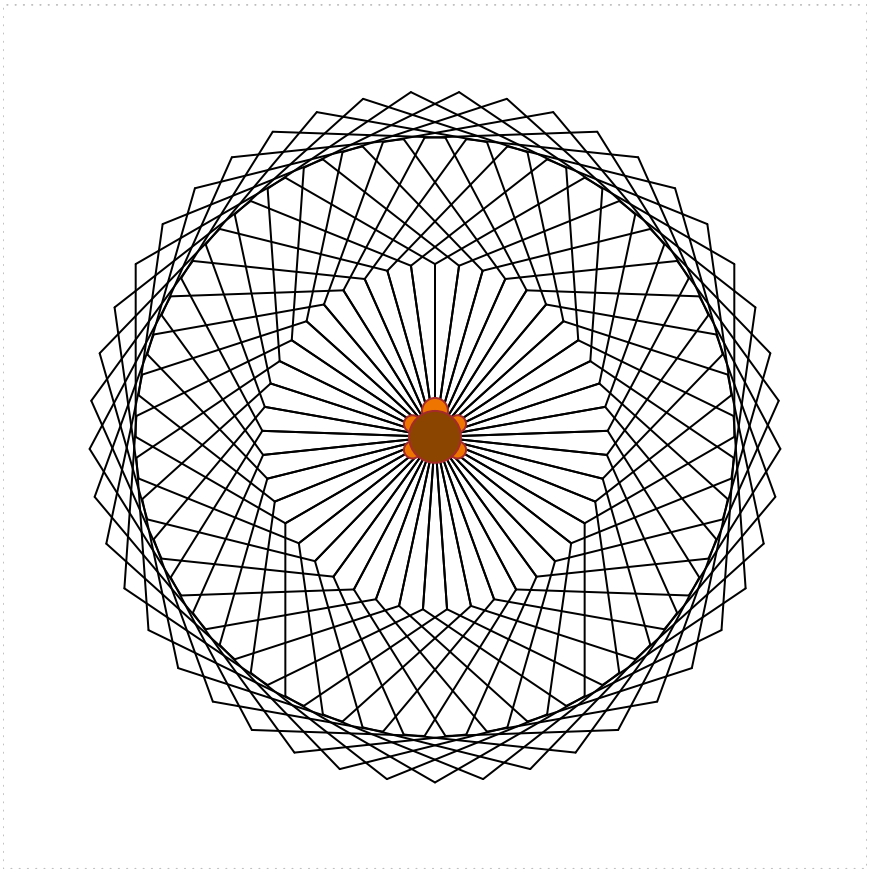
```
## Warning: package 'TurtleGraphics' was built under R version 3.6.3
```

```
## Loading required package: grid
```

```
turtle_init()
```



```
turtle_do({
  for(j in 1:45) {
    for(i in 1:6) {
      turtle_forward(20)
      turtle_right(360/6)
    }
    turtle_right(360/45)
  }
})
```



Almacenamiento de las gráficas

Tras generar una gráfica lo habitual es que nos interese guardarla en algún formato a fin de, posteriormente, imprimirla o incluirla en algún estudio. El panel `Plots` de RStudio, en el que han ido apareciendo las gráficas de los ejercicios, cuenta con un menú `Export` desde el que podemos guardar la gráfica actual tanto en formato PDF como en distintos formatos gráficos.

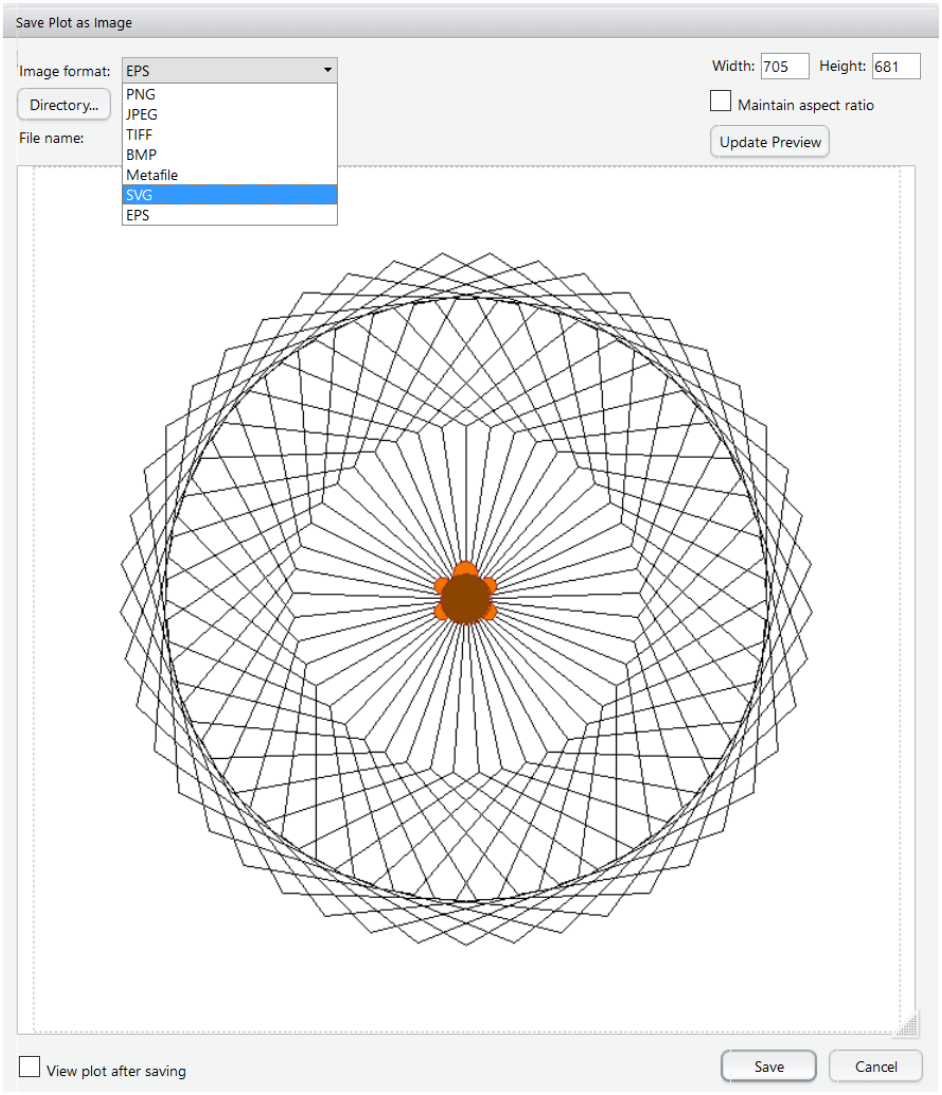


Figure 4: El panel para guardar una gráfica

La mayoría de formatos gráficos son estáticos, por lo que únicamente permiten mostrar una configuración concreta de la información representada. Algunos de ellos, sin embargo, contemplan el uso de *animaciones*. Con el comando `saveGIF()` del paquete `animation` podemos generar una gráfica animada, mostrando cómo cambia la función representada en función de algún parámetro de entrada. En lugar de `saveGIF()` podemos utilizar el comando `saveVideo()` para generar un vídeo MPEG.

```
library(animation)
```

```
## Warning: package 'animation' was built under R version 3.6.2
```

```
saveGIF({
  for(lim in seq(-3.14,3.14,by=0.1))
    curve(sin, from=lim,  to=lim + 9)
}, movie.name="animacion.gif", interval=0.2, ani.width=640, ani.height=640)
```

```
## Output at: animacion.gif
```

```
## [1] TRUE
```

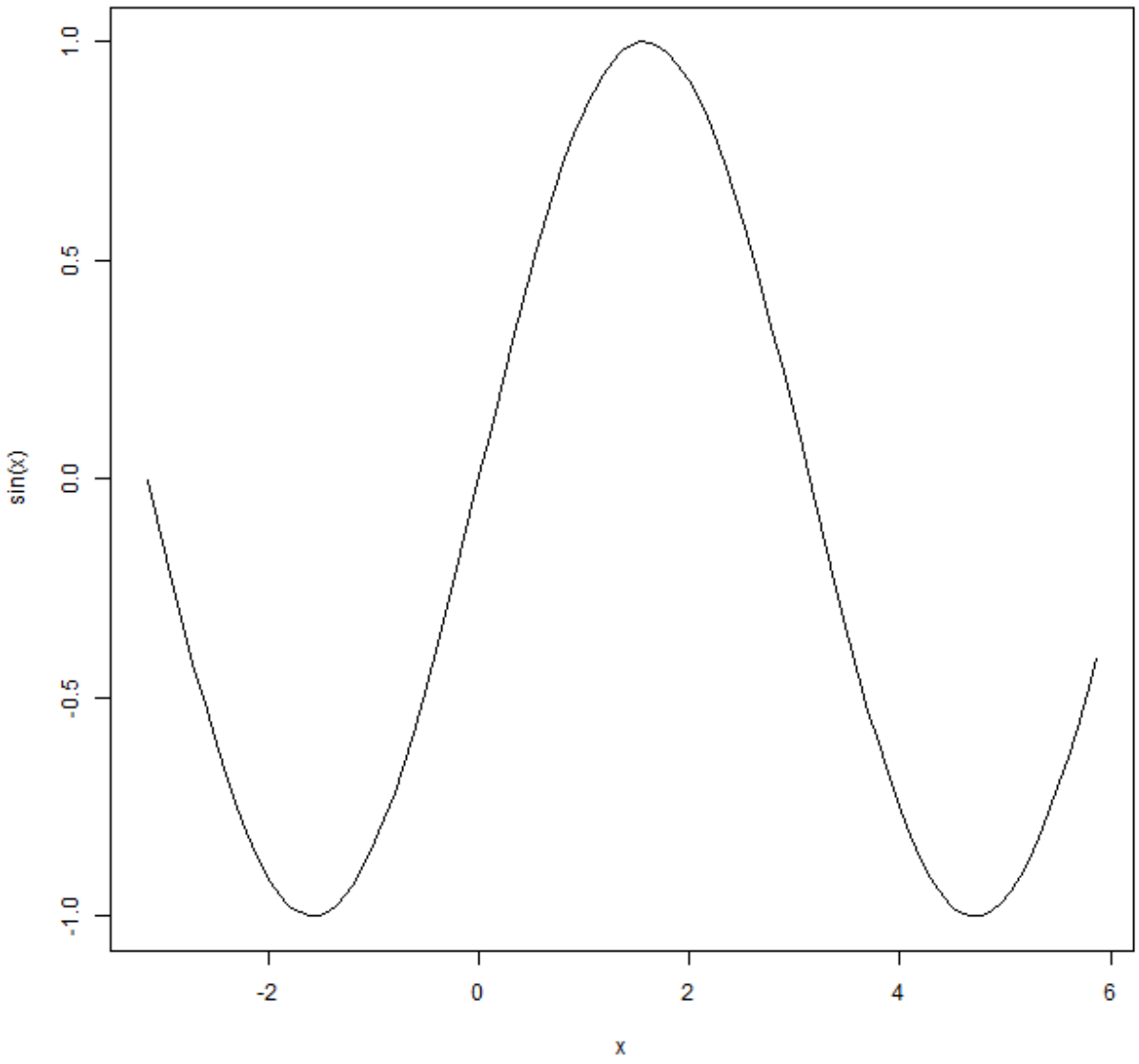


Figure 5: Animación producida por el código previo

Ejercicio

Para finalizar nuestro aprendizaje de R y la forma de usarlo para analizar y visualizar datos se propone realizar el ejercicio que se describe a continuación. Los resultados se resumirán en un *notebook* R que nos servirá como material útil posteriormente, a la hora de recordar cómo realizar cualquier tarea.

1. Instalación del software
    - Descarga la versión de R apropiada para el sistema operativo que tenga tu ordenador: Windows, OS X o GNU/Linux, y completa la instalación
    - Descarga la versión de RStudio apropiada para el sistema operativo de tu ordenador y completa la instalación
    - Inicia RStudio e instala al menos el paquete `ggplot2`
    - Comprueba que todo funciona perfectamente generando alguna gráfica básica con el conjunto de datos `iris`
2. Carga/Selección de los datos
    - Selecciona algún **conjunto de datos propio** relacionado con tu actividad o investigación. Si los datos están en un archivo de texto o una hoja de Excel, usa los comandos explicados en la primera sección. Si están en algún otro formato/aplicación, puedes consultarnos sobre cómo importarlos en R
    - Si no cuentas con datos propios, usa el comando `data()` de R para obtener una lista de conjuntos de datos y selecciona uno o dos que te resulten interesantes
3. Estadística descriptiva
    - Recopila los estadísticos básicos de las variables con que cuente tu conjunto de datos y lleva a cabo un pequeño análisis de la distribución que parecen seguir: ¿es similar para algunas variables? ¿hay correlación entre algunas de ellas?
4. Generación de gráficas
    - Dependiendo de lo que te interese de los datos con que trabajas: analizar la evolución de una variable respecto a otra, compararlas, etc., genera las gráficas que creas adecuadas
    - Guarda las gráficas en archivos a fin de poder incluirlas en el documento a entregar
5. Análisis de la distribución de los datos
    - Agrega a las gráficas previas algunas específicas para estudiar la distribución de las variables que creas adecuadas: diagrama de cajas y bigotes o histogramas
    - Guárdalos también para incluirlos en el documento
6. Completar el documento
    - Completa el documento con las explicaciones que consideres necesarias, en todo caso someras, sobre los aspectos de interés o conclusiones que obtengas