Visualización de datos con ggplot2

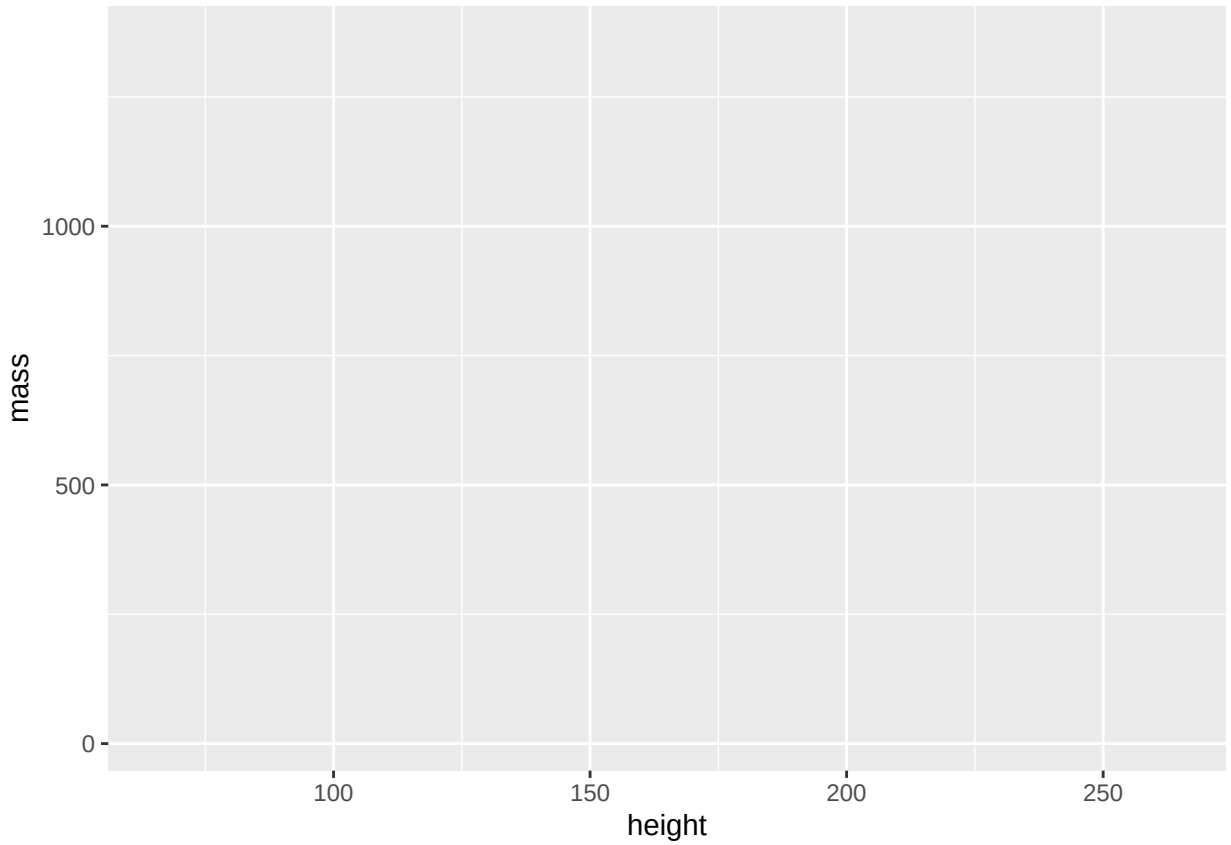
David García Callejas

En esta sesión veremos cómo realizar diferentes tipos de gráficos en R, y algunas opciones de personalización de las más usadas. Usaremos el paquete `ggplot2`, que es seguramente el más usado hoy día para producir gráficos de calidad. R tiene otros paquetes y funciones para generar gráficos, pero `ggplot2` es intuitivo, fácilmente editable, y tiene una documentación muy buena. El acrónimo `ggplot2` viene de "grammar of graphics plots", por el hecho de que este conjunto de funciones esta basado en la llamada "Grammar of Graphics", una sintaxis que estructura la creación de gráficos a partir de capas con diferente información.

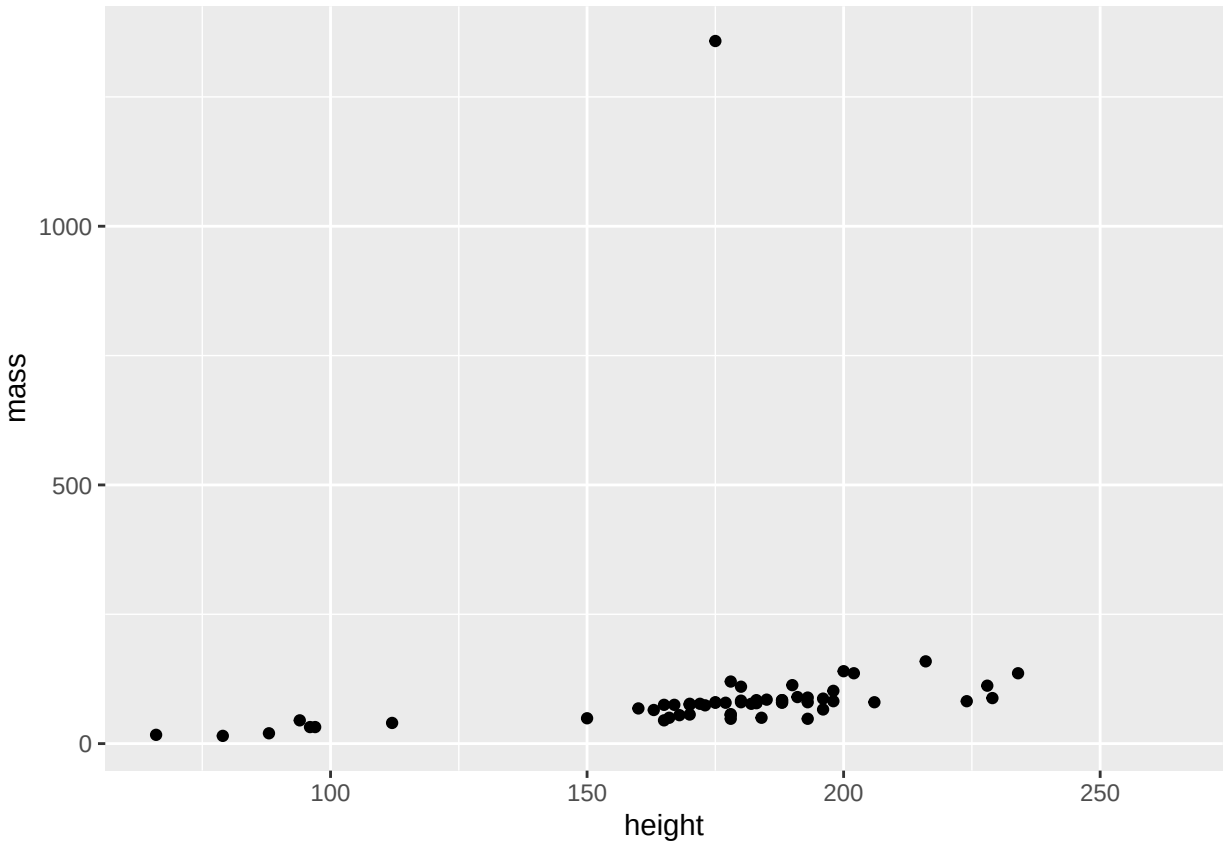
Como primer ejemplo, vamos a crear un gráfico de puntos, el mismo que usamos para la sesión sobre modelos de regresión lineal

```
# este paquete incluye ggplot2
library(tidyverse)

# leemos los datos de personajes de Star Wars
personajes_SW <- read.csv2(file = "../data/starwars_info_personajes.csv",
                           stringsAsFactors = FALSE)

# queremos crear una figura que muestre la altura y el peso de cada personaje,
# donde cada personaje sea un punto

# lo primero que tenemos que entender es que un "plot", una figura,
# también la podemos guardar en una variable.
# Para crear una figura nueva, usamos la función ggplot
# en esta función, tenemos que especificar qué datos
# queremos usar para la figura

figura1 <- ggplot(data = personajes_SW)

# ¿qué contenido tiene este objeto?
figura1 # en Rstudio, aparecerá la zona de "plots" en blanco.
```

```
class(figura1) # este objeto es de clase ggplot

# a partir de aquí, podemos ir añadiendo "capas" de información
# a nuestra figura vacía
# lo primero que debemos pensar en incluir son los ejes de nuestra figura.
# Esto se puede hacer también en la función ggplot

# el argumento "aes" significa "aesthetics", es decir,
# la estética, los detalles de la figura.
figura1 <- ggplot(data = personajes_SW,aes(x = height, y = mass))
figura1
```




```
# ahora deberíamos ver los ejes sobre los que queremos dibujar nuestros puntos.
# Estos puntos, uno por cada personaje, serán una capa nueva de información.
# Las capas se añaden en ggplot2 con las funciones "geom_",
# literalmente añadiendo elementos al objeto que hemos creado
figura1 <- figura1 + geom_point()
figura1
```



Con esto ya tenemos nuestro primer gráfico con ggplot2. Podemos guardar cualquier gráfico que creemos de diferentes maneras. Por ejemplo, Rstudio nos permite exportar cualquier gráfico que aparezca en la pestaña "Plots", usando el botón "Export". Sin embargo, es más recomendable usar la función `ggsave` del propio paquete ggplot2, que da más control y automatización.

```
# ggsave por defecto usa pulgadas como unidades...
# yo suelo ir haciendo ensayo-error con diferentes tamaños
# hasta encontrar el balance adecuado entre tamaño de letra/puntos, y fondo
ggsave(filename = "mi_figura.png",plot = figura1,width = 5,height = 5)
```

Antes de ver otros tipos de figuras, veremos cómo retocar esta figura inicial. Una primera mejora puede ser colorear cada punto según su especie.

```
# lo primero que haremos es, como antes, eliminar a Jabba,
# que nos descuadra toda la imagen.
sw2 <- subset(personajes_SW, name != "Jabba Desilijic Tiure")

# y creamos una figura en la que cada especie tenga un color diferente
# para esto añadimos un "aesthetic" más,
# pero dentro de la función que dibuja los puntos.
figura2 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species))

# la figura funciona, pero hay muchas,
# demasiadas especies como para que sea efectiva.
# podemos agrupar las especies en grupos más grandes, para simplificar.
# ¿Cuántos personajes hay de cada especie?
table(sw2$species)
```

```
# de casi todas las especies hay un solo individuo.
# Dejaremos los droides, los humanos,
# y, por curiosidad, a Yoda. El resto los agruparemos en una categoría "otros"
# primero, creamos una columna idéntica a "species"
sw2$species_group <- sw2$species
# y agrupamos aquellas que no son ni "Human", ni "Droid", ni "Yoda's species"
sw2$species_group[which(!sw2$species %in% c("Human",
                                            "Droid",
                                            "Yoda's species"))] <- "Other"
table(sw2$species_group)

# rehacemos nuestra figura con la nueva clasificación
figura2 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species_group))

# también podemos asignar valores fijos a todos los puntos.
# Fijaos en la diferencia entre la orden anterior y esta
figura2.2 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(color = "darkgreen")
```

Al incluir "color" o cualquier otro argumento dentro de `aes`, estamos diciendo que queremos que el color varíe con respecto a otra columna, en este caso "species_group". Pero si dentro de `geom_point` especificamos directamente un color, todos los puntos quedarán cambiados. Los atributos más comunes que podemos modificar son, además del color, la forma (shape), tamaño (size), o el relleno (fill).

```
figura2.3 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(color = "darkgreen", size = 3, aes(shape = species_group))
```

Las formas que pueden tener los puntos en R vienen dadas por un código numérico. Aquí podéis ver a qué forma corresponde cada código: http://www.cookbook-r.com/Graphs/Shapes_and_line_types/

Al crear la figura, ggplot elimina automaticamente los datos que sean NA. Al lanzar el código en vuestro ordenador, habréis visto un "warning" avisando de que hay 28 filas con NA. Antes de continuar explorando otras opciones para retocar la leyenda, las etiquetas, títulos, ejes, etc, veremos otros tipos de figuras que podemos crear.

```
# gráfico de barras: cuántos individuos hay por cada grupo de especies.
# Para esta figura no necesitamos añadir eje y,
# porque este será el conteo de cada grupo.
# el conteo se calcula con geom_bar, añadiendo el argumento "stat = "count""
figura3 <- ggplot(sw2, aes(x = species_group)) +
  geom_bar(stat="count")

# como antes, podemos colorear cada grupo.
# Para colorear areas, usamos la orden "fill",
# que significa, literalmente, "relleno".
figura3 <- ggplot(sw2, aes(x = species_group)) +
  geom_bar(stat="count", aes(fill = species_group))

# gráfico de cajas (boxplot)
figura4 <- ggplot(sw2, aes(x = species_group, y = height)) +
  geom_boxplot(aes(fill = species_group))

# distribución de puntos
figura5 <- ggplot(sw2, aes(x = species_group, y = height)) +
```

```
  geom_jitter(aes(color = species_group))

# puntos y cajas
figura6 <- ggplot(sw2, aes(x = species_group, y = height)) +
  geom_jitter(aes(color = species_group), alpha = .8) +
  geom_boxplot(aes(fill = species_group), alpha = .5)

# histogramas
figura7 <- ggplot(sw2, aes(x = height)) +
  geom_histogram(bins = 30)

figura8 <- ggplot(sw2, aes(x = height)) +
  geom_density()
```

También podemos, como vimos en la clase sobre modelos estadísticos, añadir regresiones a nuestras figuras. Para ello usamos la orden `geom_smooth`. Este ejemplo también nos sirve para comprobar, de nuevo, que podemos "apilar" todas las capas que queramos.

```
# Por defecto usa un método de regresión polinomial (loess)
figura9 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species_group)) +
  geom_smooth()

# pero podemos especificar otros modelos
figura10 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species_group)) +
  geom_smooth(method = "lm")

# igual que dividir el color por grupos, podemos dibujar
# una regresión por cada grupo
figura11 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species_group)) +
  geom_smooth(method = "lm", aes(color = species_group))

# algunos detalles
figura12 <- ggplot(sw2, aes(x = height, y = mass)) +
  geom_point(aes(color = species_group)) +
  geom_smooth(method = "lm",
              aes(color = species_group),se = FALSE)
```

Ahora podemos entender mejor por qué ggplot2 trabaja con datos en formato "long". En este formato, los "grupos" de datos están en una columna, en vez de divididos en varias columnas.

```
head(sw2)

sw2.ejemplo <- sw2[,c("name","height","mass","species_group")]
sw2.ejemplo$valor <- TRUE

sw3.ejemplo <- pivot_wider(data = sw2.ejemplo,
                           id_cols = c(name,height,mass),
                           names_from = species_group,
                           values_from = valor,
                           values_fill = list(valor = FALSE))
head(sw3.ejemplo)
```

Al dividir la especie en columnas, sería imposible crear una figura con todas las categorías de manera sencilla: ¿cuál sería la columna para especificar la agrupación?

```
sw3.plot <- ggplot(sw3.ejemplo, aes(x = height, y = mass)) +
  geom_point(aes(color = Human)) # esto sólo nos diferencia humanos de otros
```

Para hacer gráficos de líneas usamos los datos de terremotos. Vamos a dibujar el número de terremotos por siglo. Este ejemplo es un poco más complejo por la preparación previa que hacemos de los datos.

```
eq <- read.csv2(file = "../data/Earthquake_data.csv",dec = ".",stringsAsFactors = FALSE)

# necesitamos crear una columna "century"
# en principio vacía
eq$century <- NA

# hay varias maneras de traducir el año a siglo.
# Por lo que hemos visto hasta ahora, podemos hacerlo con un bucle,
# que vaya fila por fila,
# calcule a qué siglo corresponde el año,
# y rellene el valor de cada terremoto.

for(i in 1:nrow(eq)){

  my.century <- NA

  if(eq$Year[i] < 1500){
    my.century <- "< XVI"
  }else if(eq$Year[i] >= 1500 & eq$Year[i] < 1600){
    my.century <- "XVI"
  }else if(eq$Year[i] >= 1600 & eq$Year[i] < 1700){
    my.century <- "XVII"
  }else if(eq$Year[i] >= 1700 & eq$Year[i] < 1800){
    my.century <- "XVIII"
  }else if(eq$Year[i] >= 1800 & eq$Year[i] < 1900){
    my.century <- "XIX"
  }else if(eq$Year[i] >= 1900){
    my.century <- "XX"
  }

  eq$century[i] <- my.century

}

table(eq$century)

# lo más claro es crear un segundo dataframe sólo con esta información
terremotos_siglo <- data.frame(terremotos = table(eq$century))
# al usar el resultado de "table" para crear un dataframe,
# generamos dos columnas,
# una con el nombre del siglo y otra con el número de terremotos
terremotos_siglo
# los nombres del dataframe no son del todo correctos
names(terremotos_siglo) <- c("siglo","terremotos")
# y el orden no es adecuado, no debería ser alfabético.
# En este caso, necesitamos un factor
```

```
terremotos_siglo$siglo <- factor(terremotos_siglo$siglo,
                                 levels = c("< XVI",
                                            "XVI",
                                            "XVII",
                                            "XVIII",
                                            "XIX",
                                            "XX"))
```

Una vez tenemos los datos agrupados como nos interesa, creamos la figura

```
# a veces, R tiene comportamientos extraños.
# Si no entendéis porqué hace falta "group = 1",
# no os preocupéis... yo tampoco.
# En el "cookbook" (enlace abajo), dice:
# For line graphs, the data points must be grouped so that it knows
# which points to connect.
# In this case, it is simple - all points should be connected, so group=1.
figura13 <- ggplot(terremotos_siglo, aes(x = siglo,
                                         y = terremotos,
                                         group = 1)) +
  geom_line()
```

Como antes con los puntos, podemos especificar diferentes parámetros de la línea (de nuevo, ver http://www.cookbook-r.com/Graphs/Shapes_and_line_types/)

```
figura14 <- ggplot(terremotos_siglo, aes(x = siglo,
                                         y = terremotos,
                                         group = 1)) +
  geom_line(linetype = "dashed", size = 3, color = "darkred")
```

A veces, en vez de colorear puntos de diferente color, podemos pensar en una figura diferente para cada categoría de datos. Por ejemplo, volviendo a Star Wars

```
figura15 <- ggplot(sw2, aes(x = height, y = mass))+
  geom_point() +
  geom_smooth(method = "lm") +
  facet_grid(species_group ~ .)
```

La capa "facet_grid" divide nuestra figura en función a la columna o columnas que queramos, tanto en el eje vertical como horizontal. La sintaxis es "división_vertical ~ división_horizontal". Si sólo queremos un eje de división, como en el ejemplo de arriba, usamos un punto en el otro eje (species_group ~ .)

```
figura16 <- ggplot(sw2, aes(x = height, y = mass))+
  geom_point() +
  facet_grid(gender~species_group)
```

Estas divisiones se pueden combinar con otros atributos como colores, etc.

```
figura17 <- ggplot(sw2, aes(x = height, y = mass))+
  geom_point(aes(color = species_group, shape = gender)) +
  facet_grid(gender~species_group)
```

En este último ejemplo vemos cómo, por cada atributo que incluimos en "aes", se crea una leyenda nueva. En este caso, os habréis dado cuenta de que las leyendas son reiterativas, porque la información de cada categoría ya está en las etiquetas horizontales y verticales. Pero como ejemplo sirve. Desde esta base, vamos a retocar un poco la figura mejorando las leyendas, el texto de los ejes, los colores que usamos para cada categoría, y el título de la figura.

```
# primero, eliminamos los NA para mejorar la visualización
sw3 <- subset(sw2, !is.na(height) & !is.na(mass) & !is.na(gender))

figura18 <- ggplot(sw3, aes(x = height, y = mass))+
  geom_point(aes(color = species_group, shape = gender), size = 2) +
  facet_grid(gender ~ species_group) +
  # una nueva capa que especifica título y, opcionalmente, subtítulo
  ggtitle(label = "Peso y altura de personajes de Star Wars",
          subtitle = "según especie y género") +
  # otra capa en la que especificamos el texto de los ejes
  labs(x = "altura (cm)", y = "peso (kg)") +
  # leyendas
  # color
  scale_color_manual(name = "Tipo de especie",
                     labels = c("androide","humano", "otro", "Yoda"),
                     values = c("grey30","darkorange",
                                "darkred","darkgreen")) +
  # shape (forma)
  scale_shape_manual(name = "Género",
                     labels = c("Femenino", "Masculino", "Otro/ninguno"),
                     values = c(1,5,6))
```

Las capas que especifican los atributos vienen especificadas por la sintaxis `scale_atributo_tipo`. En este ejemplo, modificamos el atributo `color` y el atributo `shape`, y lo que hacemos es darle valores manuales, de ahí el tipo `manual`. Especificamos el nombre (name) y las etiquetas (labels) de cada valor de la leyenda, y los valores que queremos que tomen (values). Los colores se pueden especificar por su código hexadecimal, o muchos de ellos, por un nombre estandarizado (ver http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf).

Hay muchas mas opciones que se pueden modificar. La apariencia general de las figuras (los colores de fondo, las cuadrículas, etc) vienen dadas por lo que se llama "temas". En ggplot2 hay una serie de temas ya estructurados, que se pueden usar fácilmente (ver ejemplos de los temas por defecto en https://ggplot2.tidyverse.org/reference/ggtheme.html)

```
figura19 <- ggplot(sw3, aes(x = height, y = mass))+
  geom_point(aes(color = species_group, shape = gender), size = 2) +
  facet_grid(gender ~ species_group) +
  # una nueva capa que especifica título y, opcionalmente, subtítulo
  ggtitle(label = "Peso y altura de personajes de Star Wars",
          subtitle = "según especie y género") +
  # otra capa en la que especificamos el texto de los ejes
  labs(x = "altura (cm)", y = "peso (kg)") +
  # leyendas
  # color
  scale_color_manual(name = "Tipo de especie",
                     labels = c("androide","humano", "otro", "Yoda"),
                     values = c("grey30","darkorange","darkred","darkgreen")) +
  # shape (forma)
  scale_shape_manual(name = "Género",
                     labels = c("Femenino", "Masculino", "Otro/ninguno"),
                     values = c(1,5,6)) +
  # cambiamos al tema "bw", black and white
  theme_bw()
```

Además de dividir una figura por categorías, R nos permite juntar varias figuras en una sola imagen. Para ello instalamos y cargamos el paquete `patchwork`.

```
install.packages("patchwork")
library(patchwork)

figura.combinada <- figura9 + figura10
```

Fijaos en la diferencia entre el tema estándar y el bw. Por último, vamos a dar unas pinceladas sobre cómo dibujar mapas en ggplot2. El paquete tiene la función `map_data` que permite recuperar contornos de paises o regiones. A su vez, estos contornos podemos dibujarlos como polígonos, con la función `geom_polygon` (recordad que los puntos los dibujamos con `geom_point`, líneas con `geom_line`, etc.)

```
# necesitamos cargar el paquete "maps"
library(maps)

# bajamos el contorno de todos los países especificando "world" en map_data
# fijaos que podemos especificar alguna región concreta (p.ej. España)
# con el argumento "region".

world_map <- map_data("world")
# world_map <- map_data("world",region = "Spain")

# ¿qué es este objeto? simplemente un dataframe
# con diferentes puntos de latitud y longitud
head(world_map)

# latitud y longitud en los ejes
mapamundi <- ggplot(world_map, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill="gray") +
  theme_bw()

# vamos a añadir una capa de puntos que sea el punto
# donde se registraron terremotos
# para esto necesitamos dibujar dos dataframes:
# uno, world_map, nuestro mapa en formato polígono
# dos, los datos de los puntos asociados a terremotos.
# Esto se puede hacer en ggplot2 especificando,
# en cada llamada a "geom_", qué datos usamos.

# en este caso dejamos la función principal sin argumentos
puntos_terremotos <- ggplot() +
  geom_polygon(data = world_map,aes(x = long,
                                    y = lat,
                                    group = group),
               fill = "gray") +
  geom_point(data = eq, aes(x = Lon, # cuidado con el nombre de columnas...
                            y = Lat),
             color = "darkred", size = 1.2) +
  theme_bw()
```

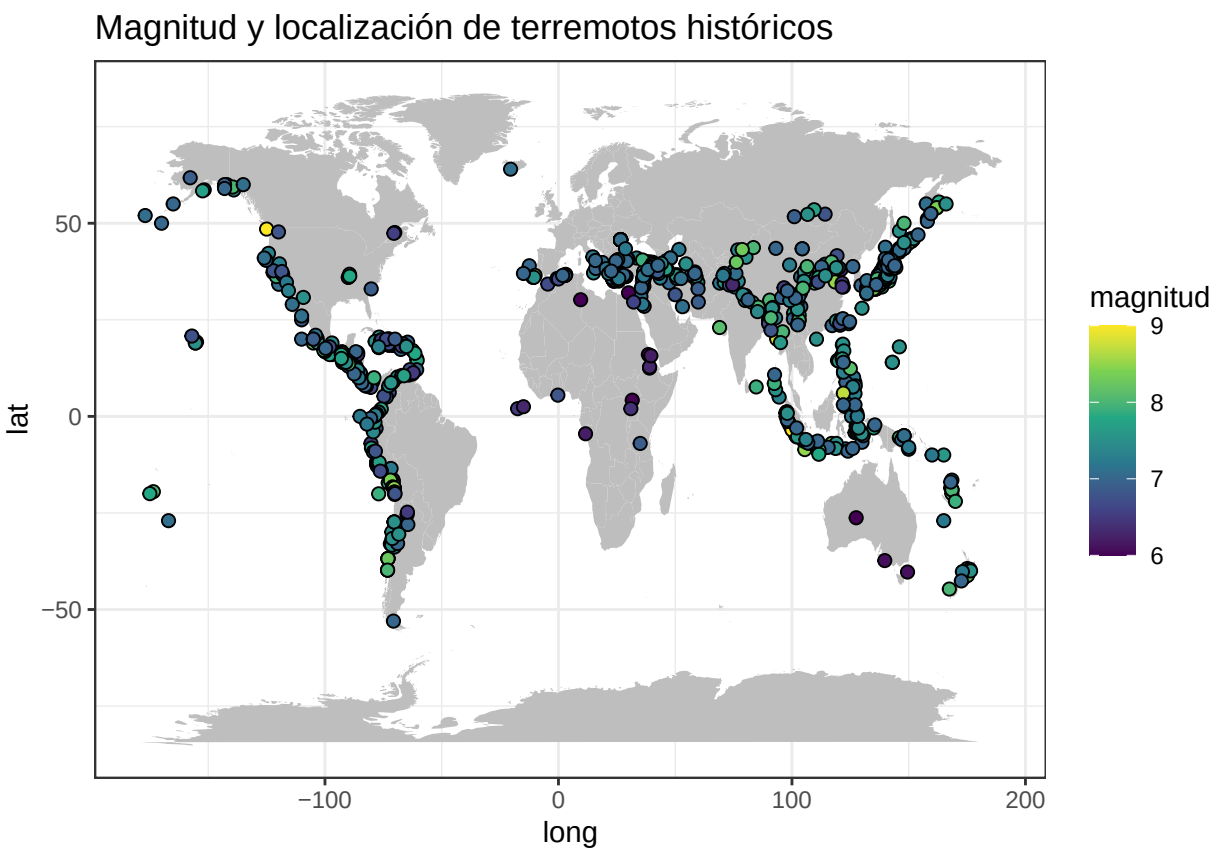
Para terminar, podemos probar a hacer el color de los puntos variable según la magnitud del terremoto

```
puntos_terremotos2 <- ggplot() +
  # mapamundi
  geom_polygon(data = world_map,aes(x = long,
                                    y = lat,
                                    group = group),
```

```
              fill = "gray") +
  # puntos
  geom_point(data = eq, aes(x = Lon,
                            y = Lat,
                            color = M),
             size = 1.2) +
  # tema
  theme_bw()
```

¿Podemos mejorarlo? Retoquemos la escala, incluyamos un título, y cambiemos el tipo de punto por uno con borde negro y relleno coloreado.

```
puntos_terremotos2 <- ggplot() +
  # mapa
  geom_polygon(data = world_map,aes(x = long, y = lat, group = group),
               fill = "gray") +
  # puntos
  geom_point(data = eq, aes(x = Lon, y = Lat, fill = M),
             shape = 21, size = 2) +
  # escala
  scale_fill_continuous(name = "magnitud", type = "viridis") +
  # título
  ggtitle("Magnitud y localización de terremotos históricos")+
  # tema
  theme_bw()

puntos_terremotos2
```



En este último ejemplo, no hemos dado valores manuales al relleno, sólo hemos dicho que use la escala "viridis". Como son datos continuos, y el atributo que queremos retocar es el relleno (fill), la orden es `scale_fill_continuous`. La escala "viridis" es muy recomendable para gradientes continuos (ver https://ggplot2.tidyverse.org/reference/scale_colour_continuous.html)

R, y ggplot2, ofrecen muchas más posibilidades para la integración de datos geográficos, por ejemplo, usando el paquete `sf`, que permite trabajar con datos vectoriales (puntos, líneas y polígonos).