R para preparación y visualización de datos

Doctorado en Neurociencia Social y Cognición, UAI

Gorka Navarrete

2024-09-12

Table of contents

Introducción

El objetivo de este seminario es aprender a usar R para preparar y visualizar datos, además de generar reportes reproducibles. Está pensado para alumnos de postgrado con conocimientos básicos de programación.

También conoceremos jsPsychR, un conjunto de herramientas creado por miembros del CSCN para ayudar a crear paradigmas experimentales con jsPsych, simular participantes, y estandarizar el proceso de preparación y análisis de datos.

R es un lenguaje de programación abierto, con una gran comunidad, orientado al trabajo, visualización y modelado de datos en contextos científicos y técnicos. Nos introduciremos de manera práctica a R, resolviendo problemas que encontramos habitualmente durante el quehacer científico, focalizándonos en el trabajo abierto, colaborativo y reproducible.

Objetivos

Dar las herramientas básicas a los alumnos para que puedan trabajar de manera autónoma con R y RStudio para el proceso de importación, transformación, visualización y reporte de datos.

Al finalizar el curso deberíamos ser capaces de:

- Importar archivos de datos, transformar los datos, crear nuevas variables.
- Realizar análisis de datos exploratorios, visualizar distribuciones y comparar grupos.
- Generar reportes reproducibles con Quarto/RMarkdown
- Crear paradigmas experimentales y un pipeline completo para la preparación de datos con jsPsychR.

Como empezar

Si ya has completado los pasos A-B-C y otras dependencias a instalar indicados en **preparando nuestro sistema**, puedes lanzar el siguiente código en tu ordenador para descargar los materiales del curso:

```
if (!require('usethis')) install.packages('usethis'); library('usethis')
usethis::use_course("gorkang/R_preparacion_visualizacion_datos")
```

Sigue las instrucciones que aparecen en la Consola para tener un **nuevo proyecto de RStudio con todos los materiales del curso**. El código anterior creará una carpeta llamada `R_preparacion_visualizacion_datos-master`.

La carpeta `R_preparacion_visualizacion_datos-master` contiene varias cosas. Las mas importantes son:

- `R_preparacion_visualizacion_datos.Rproj`: para abrir el proyecto de RStudio del curso. **Ábrelo siempre usando este archivo**.
- Carpeta `docs`: puedes abrir `docs/index.html` en tu navegador para ver el "libro" de este curso. Alternativamente, puedes consultar una version online del libro
- Carpeta `qmd`: En esa carpeta esta el código fuente de los capítulos del libro
- Carpeta `data`: Cuando usemos archivos de datos, vendrán de aquí

> En ocasiones encontraras un recuadro como este. En la versión online del libro, si haces click sobre el, aparecerá una pista sobre como resolver el ejercicio.
>
> ¡No hagas click sin antes haber intentado resolver el ejercicio sin ayuda!

Bibliografía

Bryan, J., & Hester, J. What They Forgot to Teach You About R. https://whattheyforgot.org/

Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc. https://r4ds.had.co.nz/

Wickham, H. (2014). Advanced r. Chapman and Hall/CRC. https://adv-r.hadley.nz/

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). R Markdown: The Definitive Guide. CRC Press. https://bookdown.org/yihui/rmarkdown/

Yihui Xie (2018). bookdown: Authoring Books and Technical Documents with R Markdown https://bookdown.org/yihui/bookdown/markdown-syntax.html

Preparando nuestro sistema

Antes de empezar, si tienes dificultades o curiosidad sobre el manejo básico de R, te recomiendo que veas algún tutorial de R y RStudio para principiantes como éste, o éste.

Empezando en A-B-C

Para poder iniciar el workshop necesitamos tener R y RStudio instalados, además de algunas librerías. **Para tener un sistema funcional, completa los pasos A, B y C**. Si ya tienes R y Rstudio instalados (recientemente), puedes pasar directamente al paso **(C)**.

(A) Instalar R

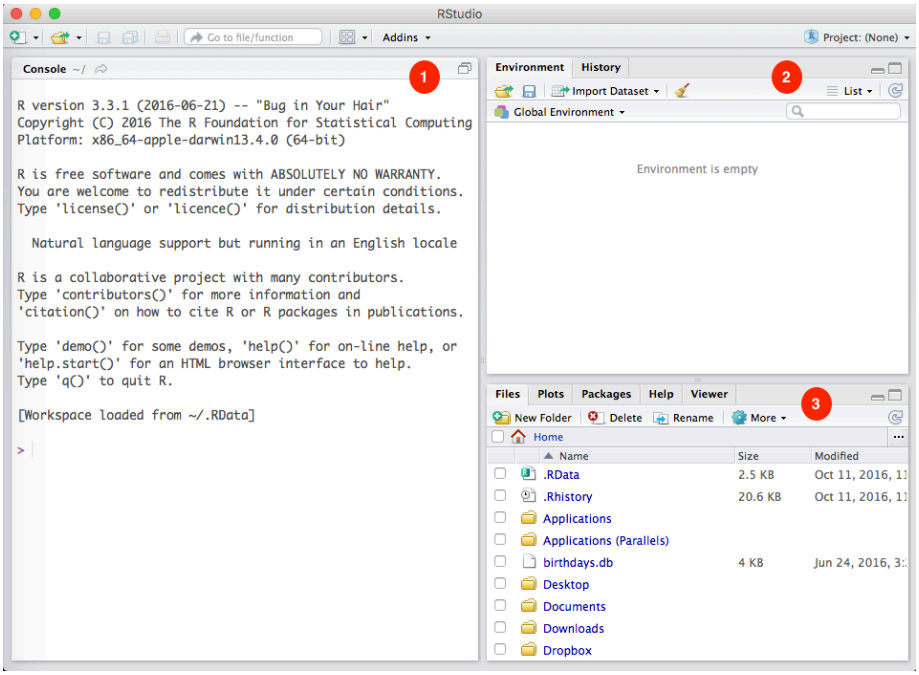
R, es un lenguaje de programación especializado en la computación estadística y visualización de datos. Es recomendable tener instalada la última versión de R (necesitarás al menos la versión 4.2). Puedes usar uno de los enlaces siguientes:

- **Windows**: Descargar e instalar R para Windows
- **Mac**: Descargar e instalar R para Mac
- **Ubuntu Linux**: más detalles en la web de R.
- **En un terminal**: `sudo apt install r-base`

(B) Instalar RStudio

RStudio es un entorno integrado de desarrollo (IDE) para la programación R.

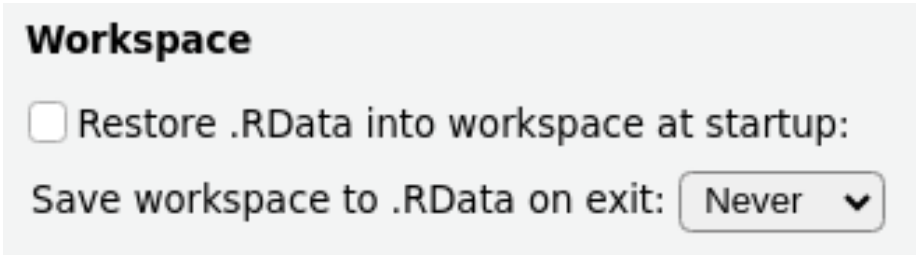
- Descargar e instalar RStudio.
- Una vez descargado e instalado, **abre RStudio**. Deberías ver algo parecido a lo siguiente:



- Si encuentras un error de instalación en ubuntu, tendrás que instalar RStudio manualmente:
  - `sudo dpkg -i rstudio-[VERSION_NUMBER]-amd64.deb`
  - `sudo apt --fix-broken install`

Configuración básica de RStudio

Una vez abierto RStudio, puedes ir a `Tools -> Global options`, y dejar la configuración del **Workspace** como se ve abajo. Esto facilita trabajar con entornos limpios, lo que hace más probable que nuestros scripts funcionen.



(C) Paquetes para el workshop

Para instalar los paquetes del workshop, ejecuta el código de más abajo (sección sombreada en gris claro) en la consola de RStudio.

Copia y pega el código de abajo en la consola de RStudio y ejecútalo [tecla ENTER]:

```
if (!require('rlang')) install.packages('rlang'); library('rlang')
rlang::check_installed(
  pkg = c("afex", "broom.mixed", "correlation", "corrr", "cowplot", "dplyr", "DT", "esquisse
          "gapminder", "geomtextpath", "ggplot2", "ggrain", "ggraph", "ggridges", "ggthemes"
          "ggtext", "googlesheets4", "grateful", "gtsummary",
          "haven", "here", "hexbin", "inspectdf", "janitor", "knitr", "lme4",
          "papaja", "parameters", "performance", "plotly", "purrr",
          "quarto", "readODS", "readr", "readxl", "remotes", "renv", "report",
          "rticles", "see", "sjPlot", "stargazer", "tidyr", "usethis", "writexl"),
  reason = "to run the initial setup")
```

Si falla el codigo de arriba, puedes intentar esto

```
if (!require('pak')) install.packages('pak'); library('pak')

pak::pak(
  pkg = c("afex", "broom.mixed", "correlation", "corrr", "cowplot", "dplyr", "DT", "esquis
          "gapminder", "geomtextpath", "ggplot2", "ggrain", "ggraph", "ggridges", "ggtheme
          "ggtext", "googlesheets4", "grateful", "gtsummary",
          "haven", "here", "hexbin", "inspectdf", "janitor", "knitr", "lme4",
          "papaja", "parameters", "performance", "plotly", "purrr",
          "quarto", "readODS", "readr", "readxl", "remotes", "renv", "report",
          "rticles", "see", "sjPlot", "stargazer", "tidyr", "usethis", "writexl"))
```

Otros paquetes que necesitaremos. Para que corran estas líneas tenemos que haber completado el paso previo.

```
if (!require('remotes')) install.packages('remotes'); remotes::install_github('gorkang/jsPsy
if (!require('regexplain')) remotes::install_github("gadenbuie/regexplain"); library('regexp
```

Usaremos un buen número de paquetes en el workshop. El proceso de instalación requiere Internet y tardará un buen rato (*en algunos sistemas puede llegar a 1 hora*).

Hay algunos meta-paquetes que simplifican la instalación de múltiples paquetes (e.g. pacman, pak, renv, …), pero dejaremos eso para más adelante.

Otras dependencias a instalar

Instalar Quarto

Quarto es un sistema de publicación de código abierto que funciona con diferentes lenguajes de programación como R o python. Lo usaremos a partir del capítulo 6.

Descarga e instala Quarto

Instalar latex

Para generar pdf's necesitaremos tener instalado Latex. tinytex nos ayudará a simplificar el proceso:

```
if (!require('tinytex')) install.packages('tinytex'); library('tinytex')
tinytex::install_tinytex() # Llevará un buen rato
```

Docker

Necesitaremos Docker para simular datos de participantes online.

Instala Docker en:

- Linux
- Mac
- Windows

Adicionalmente:

- **Windows**: Update wsl (in a command prompt): wsl - update
- **Ubuntu**:
  - En un terminal: `sudo apt install libssl-dev libcurl4-openssl-dev libxml2-dev docker`
  - Si los monos hacen su trabajo pero no aparecen los csv's, asegúrate que el usuario `docker` tiene acceso al directorio ~/Downloads

Para más detalles, puedes consultar jsPsychMonkeys setup

Git

Ver instrucciones para Windows, Mac y Linux.

Importante: en el paso *Adjusting your PATH environment* en en Windows, selecciona *Git from the command line and also from 3rd-party software*

Algo más sobre la instalación de paquetes

Los paquetes de R son una colección de funciones, datos y documentación que amplían las capacidades básicas de R.

Gran parte de las funciones y paquetes que utilizaremos en este workshop se encuentran contenidas en el meta-paquete tidyverse (este es un paquete de paquetes). No lo instalamos en **(C)**, pero si quisieras instalarlo solo tendrías que ejecutar la siguiente linea en la **consola** de RStudio:

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

Para instalar otro paquete diferente de "tidyverse", remplaza su nombre entre comillas dentro de la función:

```
install.packages("NOMBRE_DE_PAQUETE")
```

Una vez instalado un paquete, no es necesario volver hacerlo, a menos que reinstales R.

Cargar paquetes

Las funciones, datos y documentación dentro de nuestros paquetes no podrán ser utilizadas hasta que se carguen en R. Una vez instalados, para cargar los paquetes se usa la función `library()`:

```
library(ggplot2)
```

En realidad las funciones también pueden ser llamadas usando su referencia absoluta `::`, sin necesidad de cargarlas antes. Por ejemplo: `dplyr::tibble(columna = 1)`. En general: `nombre_paquete::nombre_de_funcion(parametros)`).

Todo en uno

El siguiente código simplifica lo anterior. Comprueba que el paquete esta instalado; Si no se encuentra instalado, lo instala. Finalmente lo carga.

```
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
```

Para instalar múltiples paquetes, podemos repetir la linea de mas arriba tantas veces como sea necesario, o usar una versión algo más sofisticada como el código del apartado **(C)**:

```
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
if (!require('bookdown')) install.packages('bookdown'); library('bookdown')
```

Al principio de cada capítulo, verás una sección llamada `Paquetes para este capítulo`. Si pegas el contenido de esa sección en un script de R al empezar cada capítulo, te asegurarás de tener disponibles todas las funciones que usaremos.

Instalar paquetes de Github

En ocasiones querremos instalar directamente la versión en desarrollo del paquete desde Github. Para eso podemos usar la función `install_github()` del paquete `remotes`. Por ejemplo, para instalar el paquete {BayesianReasoning} desde su repositorio de Github:

```
if (!require('remotes')) install.packages('remotes'); library('remotes')
remotes::install_github("gorkang/jsPsychMaker")
```

Bibliografía

Algunos de los manuales que vamos a usar para el workshop son los siguientes:

Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc. https://r4ds.had.co.nz/

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). R Markdown: The Definitive Guide. CRC Press. https://bookdown.org/yihui/rmarkdown/

Bryan, J., & Hester, J. What They Forgot to Teach You About R. https://whattheyforgot.org/

1 Introducción a R

1.1 Introducción: ¿por qué la visualización de datos es importante?

"These 13 datasets (the Datasaurus, plus 12 others) each have the same summary statistics (x/y mean, x/y standard deviation, and Pearson's correlation) to two decimal places, while being drastically different in appearance." (Matejka, J., & Fitzmaurice, G., 2017).

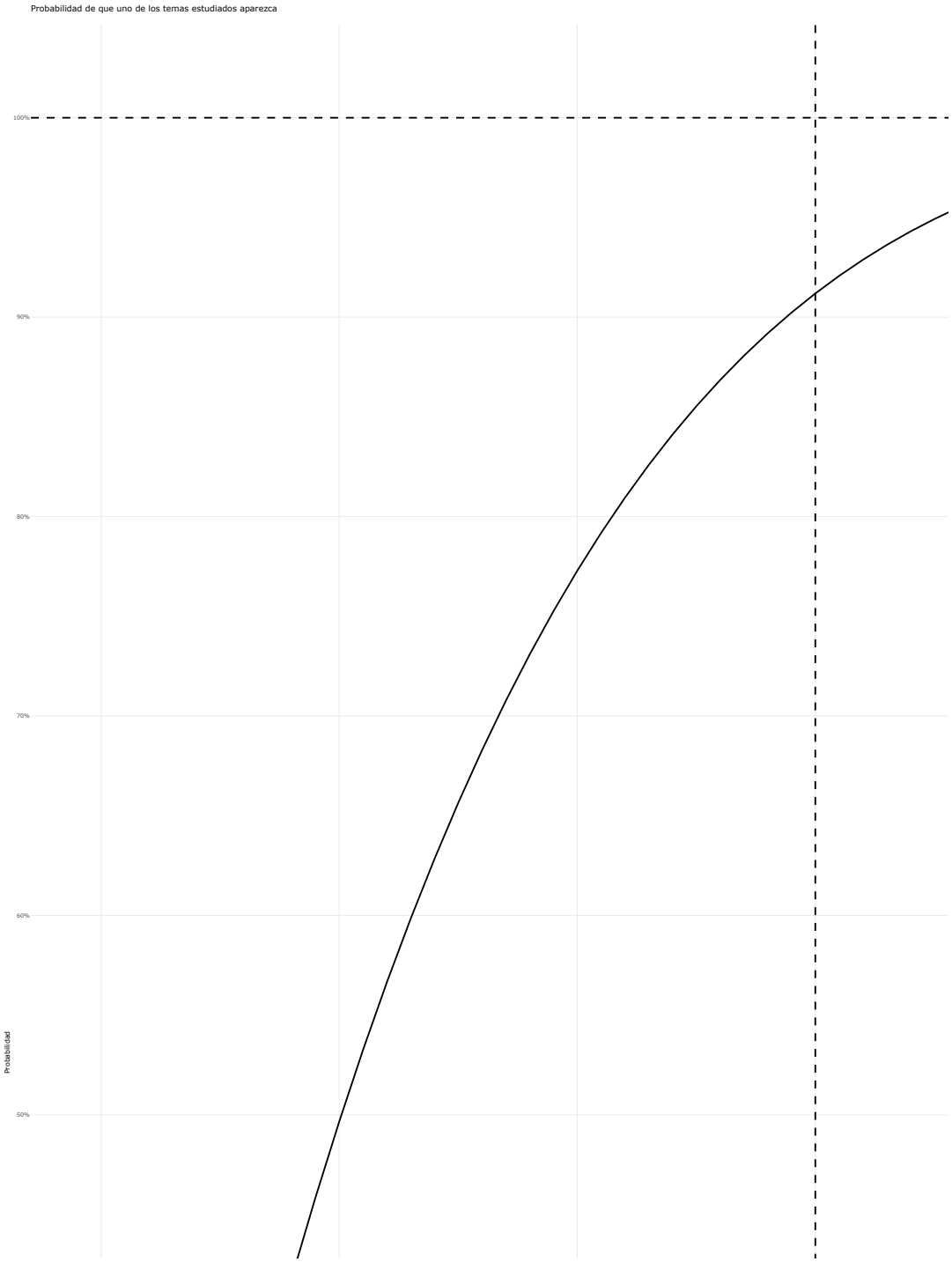
See Datasaurus

1.1.1 Ejemplo del mundo real: ¿cuantos temas deberia estudiar?

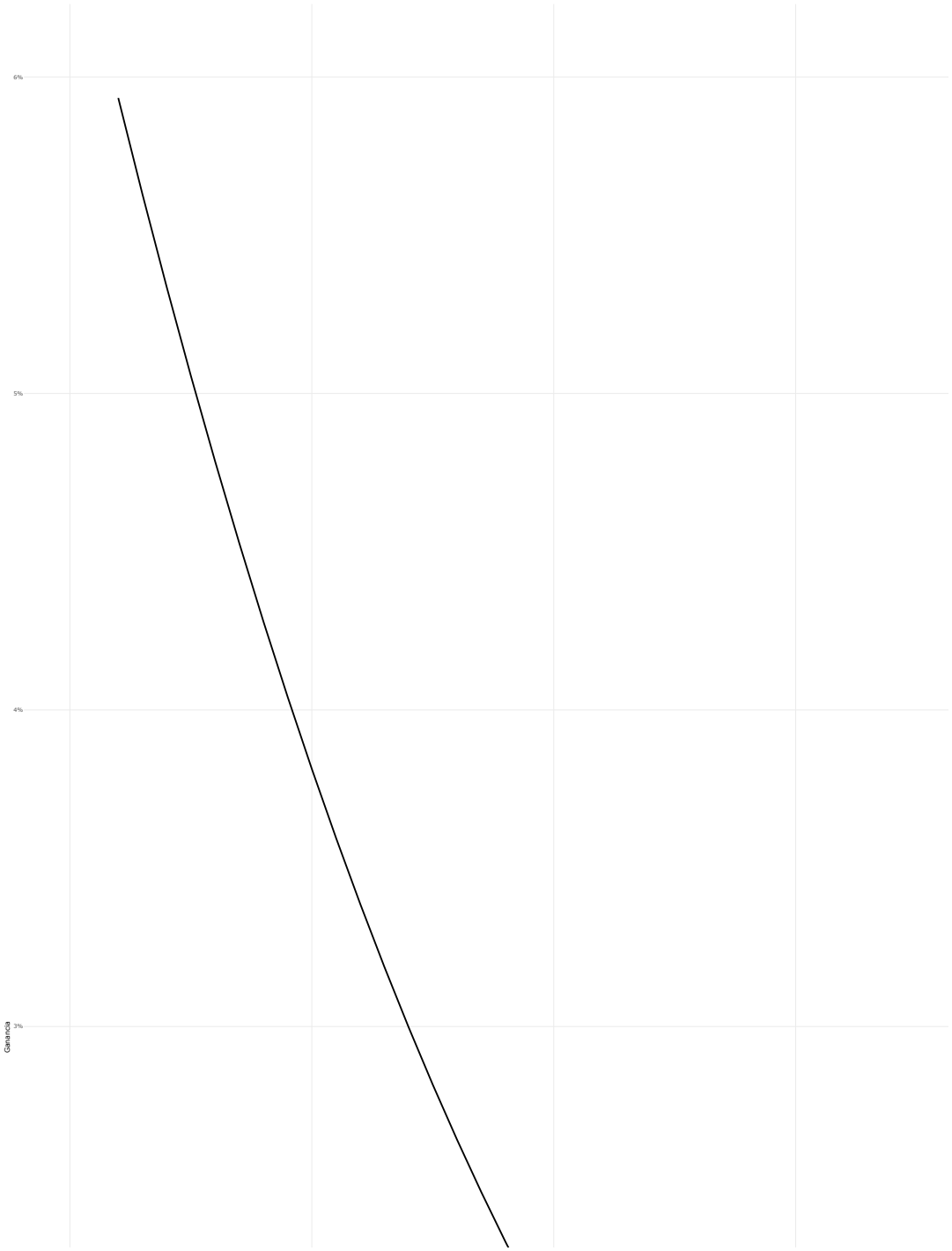
Este ejemplo viene de un experimento que realizamos junto con Carlos Santamaría hace algún tiempo. Presentamos una tarea sobre cálculo de probabilidades a personas que estaban entrando a un examen para convertirse en trabajadores del estado.

Simplificando algo, digamos que la materia para el examen eran 80 temas. No es posible estudiar con profundidad todos los temas, así que los opositores se concentraban en un subconjunto de esos temas (e.g. 30 de 80). Al empezar el examen, se seleccionaban al azar 5 de los 80 temas, y cada persona elegía uno de ellos para desarrollar.

Abajo se puede ver como cambia la probabilidad de que uno de los temas estudiados aparezca dentro de los 5 seleccionados al azar. Con 30 de los 80 temas estudiados, la probabilidad de que uno de ellos salga en la prueba es del 91%. Si estudiáramos 47, subiríamos a una probabilidad del 99%.



La esencia del dilema al que se enfrentan los opositores se puede condensar en este gráfico. La ganancia (en probabilidad de que salga un tema estudiado) va disminuyendo con cada tema adicional, hasta llegar a un punto en el que es negligible. El problema es que esto es muy poco intuitivo…



En el experimento le preguntamos a los participantes por la probabilidad de que les apareciera alguno de los temas estudiados en la prueba. Comparamos las siguientes dos preguntas:

- ¿Cuál es la probabilidad de que **salga uno** de los temas que has estudiado?

- ¿Cuál es la probabilidad de que **no salga ninguno** de los temas que has estudiado?

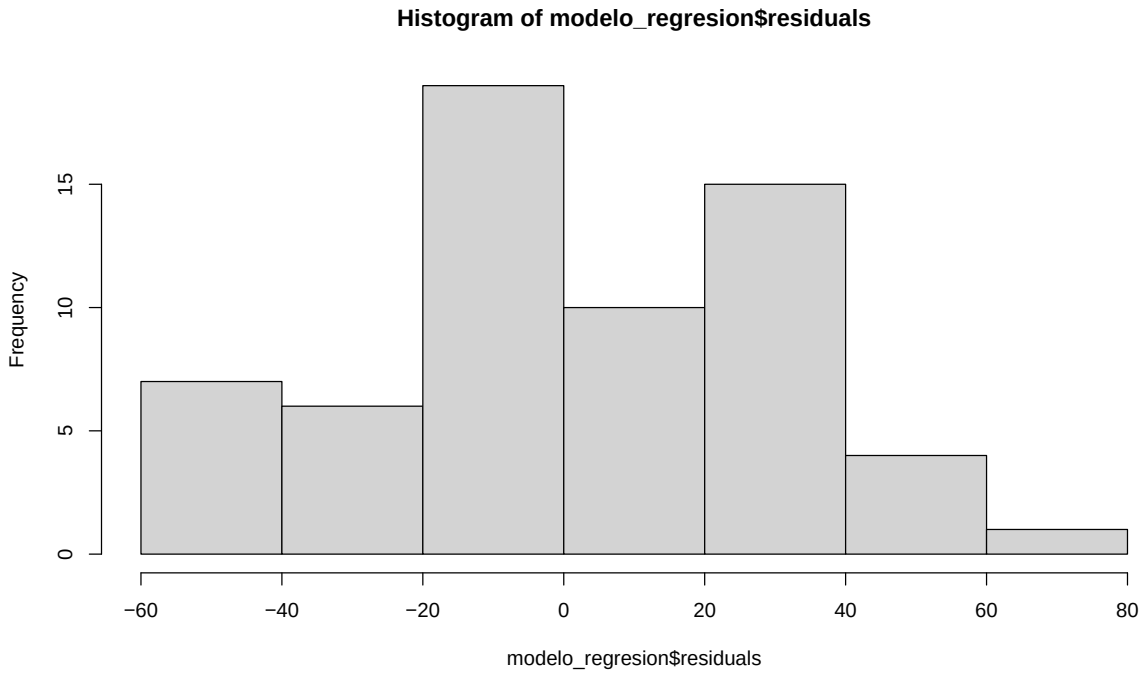
Miramos el error promedio en función de la pregunta (cuanto se han alejado de la probabilidad correcta), y vimos que nuestra manipulación había tenido un efecto considerable:

Question	Error_promedio	SD	N
p (salga uno)	-30.741936	20.01494	31
p (no salga ninguno)	4.016129	35.82469	31

Hay una diferencia notable entre condiciones. Pasamos de un error promedio del -30.7% a tan solo 4%, simplemente cambiando la pregunta. Hagamos un sencillo análisis de regresión para ver si la diferencia es significativa, y cuanta varianza explica nuestro modelo.

Predictors	Estimates	Error CI	p
(Intercept)	4.02	-6.41 – 14.44	0.444
Question [p (salga uno)]	-34.76	-49.50 – -20.02	**<0.001**
Observations	62		
R^2 / R^2 adjusted	0.270 / 0.258		

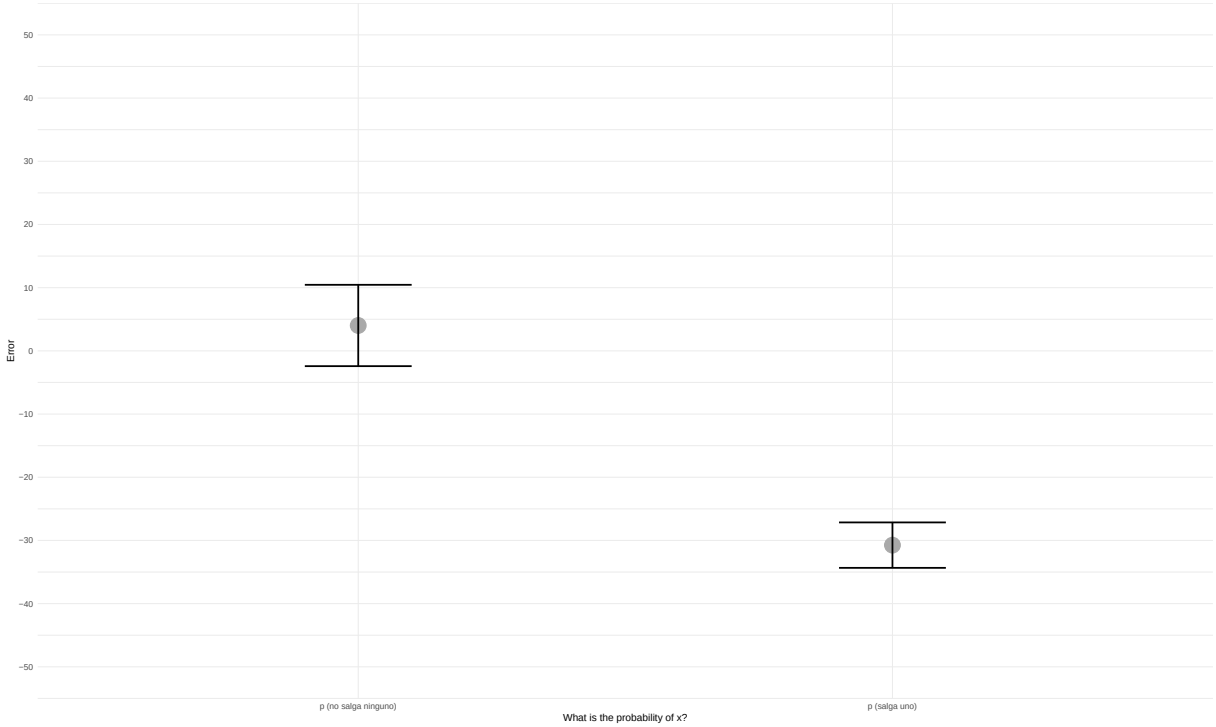
```
#>
#>  Shapiro-Wilk normality test
#>
#> data:  modelo_regresion$residuals
#> W = 0.96215, p-value = 0.0532
```



Todo es hermoso. Tenemos un efecto claramente significativo de la pregunta (y con un R2-ajustado de .258, no está nada mal), y además, nuestro modelo no incumple el supuesto de normalidad de residuos (¡por los pelos!).

> Nota importante sobre las pruebas de normalidad. *Hack click para leerme.*
>
> Las pruebas de normalidad son muy sensibles al tamaño de la muestra. Como el tamaño de la muestra es pequeño en este caso, no es esperable que el resultado sea significativo.

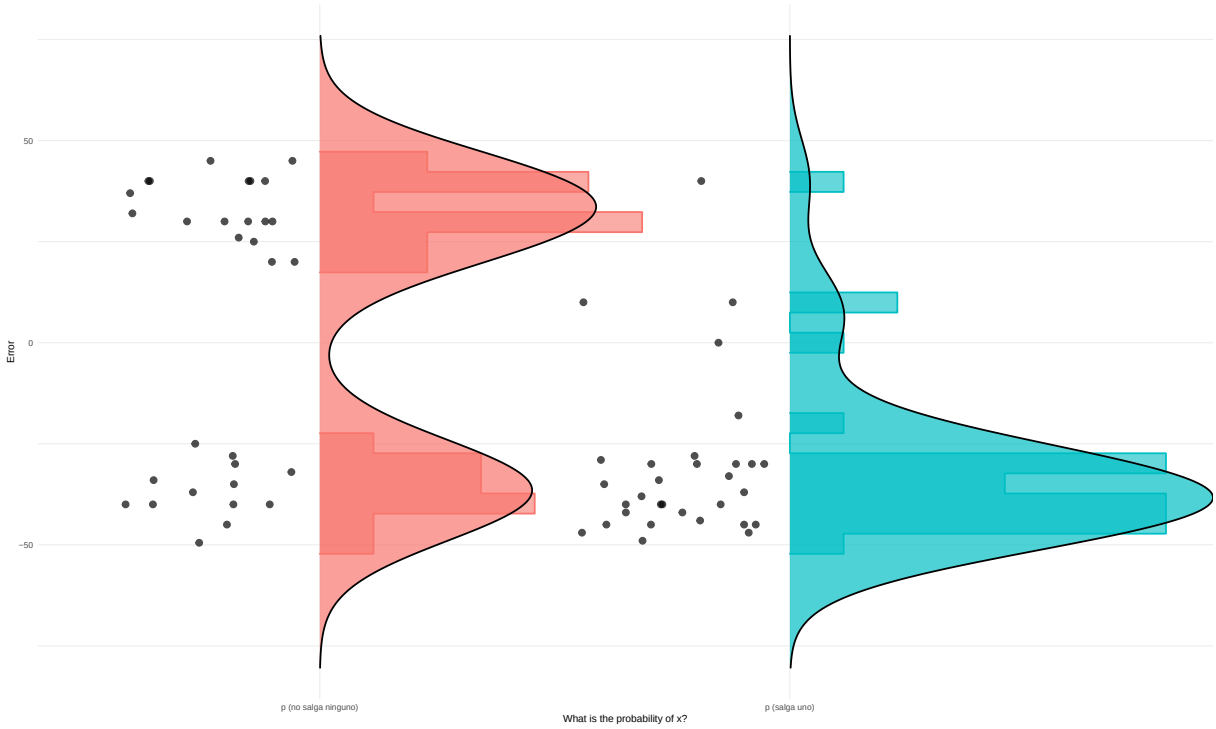
Preparamos un plot con promedios y barras con error standard para nuestro paper.



Estamos listos para escribir el paper. Preparemos la tabla con descriptivos…

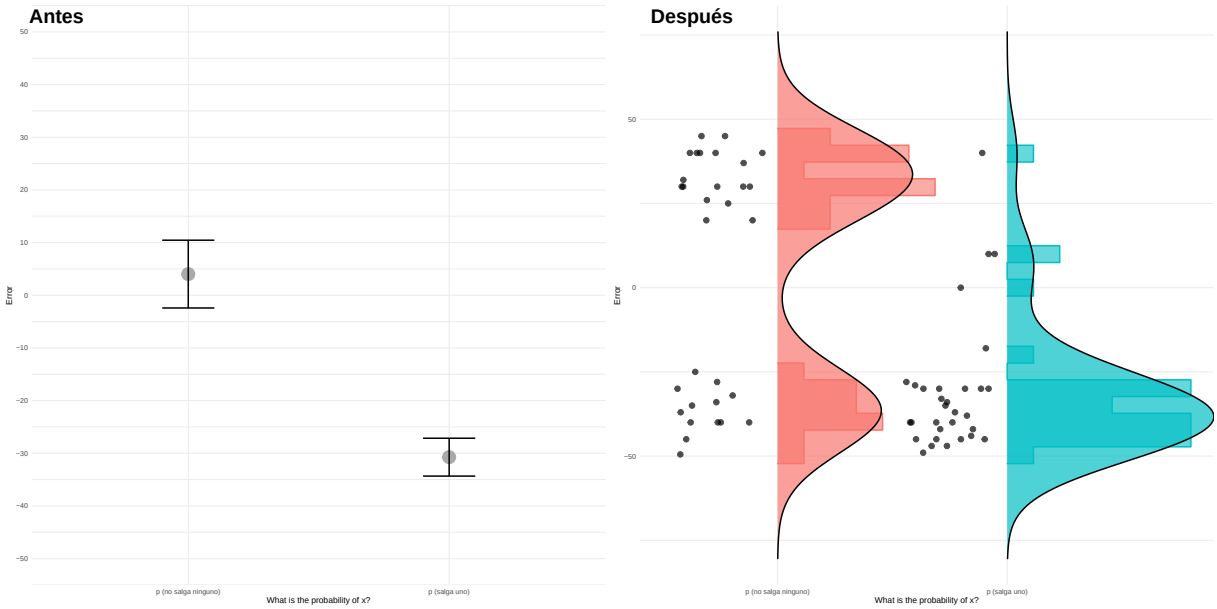
Question	Error_promedio	SD	N
p (salga uno)	-30.741936	20.01494	31
p (no salga ninguno)	4.016129	35.82469	31

Es curioso que la desviación estándar sea mayor en el grupo con menos error promedio… Visualicemos las respuestas de todos los participantes, junto con la distribución de los datos.



Como se puede apreciar en la gráfica, cuando usamos la pregunta `¿Cuál es la probabilidad de que no salga ninguno de los temas que has estudiado?` no estamos reduciendo el error, sino **convirtiendo una distribución de respuestas unimodal en bimodal**.

TLDR: La manera en la visualizamos la información determina las conclusiones a las que llegamos. En una sola gráfica:



Moraleja: es importante mostrar los datos individuales y/o la distribución de los datos

See Datasaurus

1.2 ¿Por qué R?

R es uno de los programas para data science mas populares, especialmente usado en la academia. El numero de paquetes que ofrecen funcionalidades de todo tipo no ha dejado de crecer. En 2024 el numero de paquetes en R-cran ha superado los 25,000, y el ritmo de crecimiento nos acerca a la singularidad… ;)

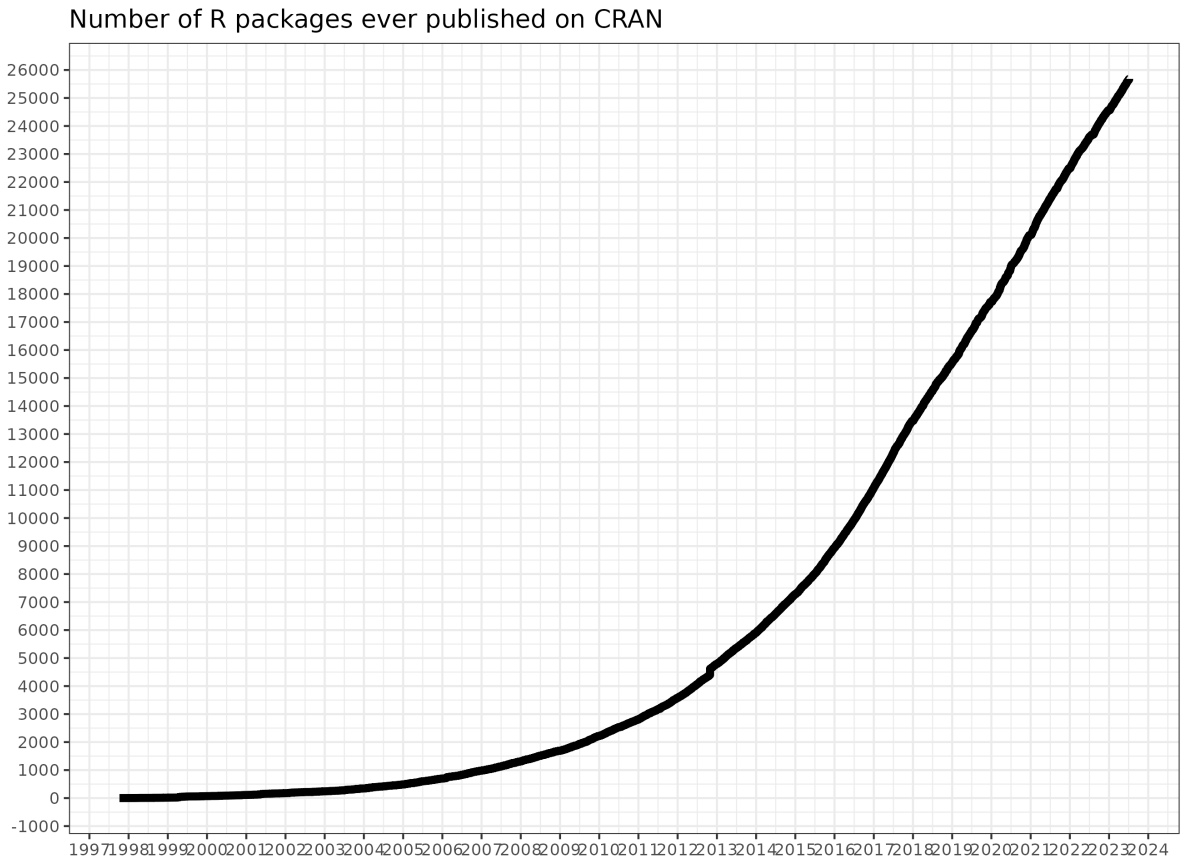


Figure 1.1: SOURCE: https://gist.github.com/daroczig/3cf06d6db4be2bbe3368

Además de lo anterior, R es un programa de código abierto (algo esencial para poder hacer ciencia reproducible), con una comunidad de usuarios muy acogedora, y con un importante foco en la inclusividad.

La importancia de la comunidad es difícil de apreciar. Por ejemplo, es relativamente habitual que uno abra un issue en Github pidiendo una nueva característica en un paquete, y que los creadores la implementen (e.g. correlation, gtsummary, rorcid), que uno reporte un error y lo corrijan (e.g. sjPlot, gtsummary), recibir correcciones y mejoras en tus repositorios (e.g. html2latex, 2019-Chile), o poder contribuir a repositorios de otros (e.g. jsPsych, gtsummary).

Sus funciones de visualización son muy potentes (ver la r-graph-gallery para algunos ejemplos), siendo usadas como herramienta principal en medios como la BBC.

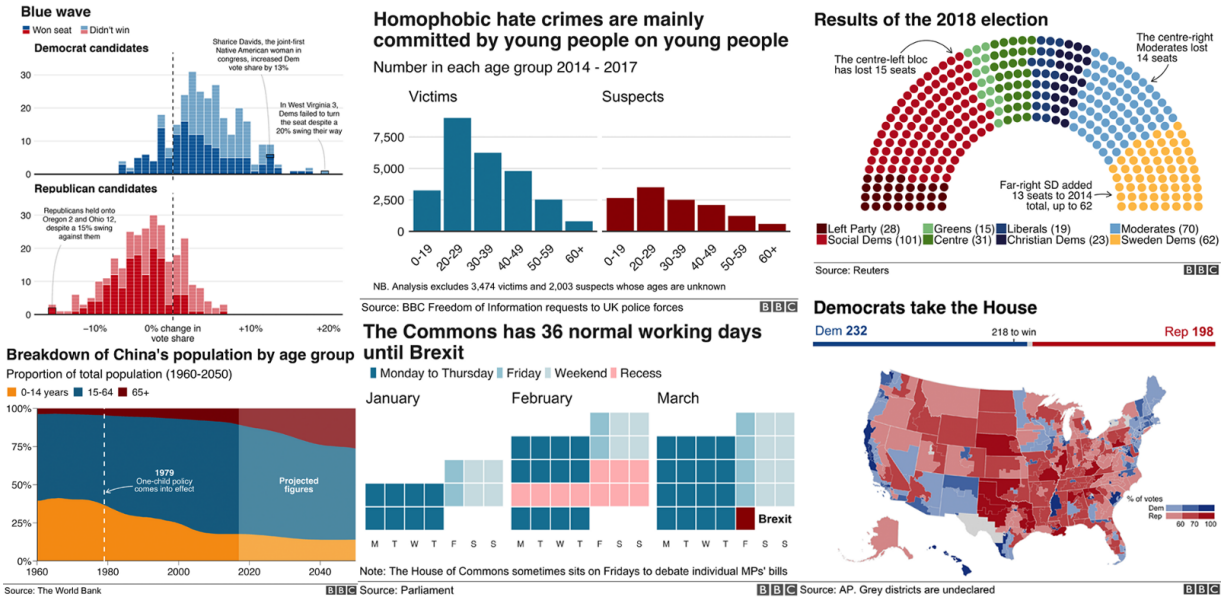


Figure 1.2: SOURCE: BBC

No menos importante, hay una gran cantidad de cursos, tutoriales, presentaciones y libros de una calidad excelente, con los que podemos aprender de manera autónoma. Por ejemplo:

- psyTeachR team at the University of Glasgow
- A Gentle Guide to the Grammar of Graphics with ggplot2
- resulumit.com Rmd workshop
- R for Data Science
- Advanced R

Para ver una compilación de libros disponibles (> 300): Big Book of R

1.3 ¿Para qué sirve R?

Con R puedes recoger datos interactivamente con shiny, preparar datos (o extraerlos de paginas web con rvest o RSelenium), visualizar datos estáticos con ggplot, animarlos con gganimate, visualizarlos con interactivamente con plotly o shiny.

Puedes también analizar los datos con todas las técnicas imaginables, desde anovas con afex a modelos mixtos con lmer y/o afex, pasando por meta-análisis con metafor, SEM, Path analysis, mediación, con lavaan, análisis Bayesianos con brms o bayesfactor, y un larguísimo etc.

Puedes llevar tus visualizaciones y análisis a reportes automáticos en múltiples formatos (pdf, html, docx) con Rmarkdown, o quarto, crear libros como este con bookdown, páginas web con blogdown o distill, e incluso papers completamente reproducibles (preparación y análisis de datos) en formato APA con papaja.

1.3.1 Bienvenida al tidyverse

El tidyverse es un conjunto de paquetes que nos permitirán hacer de manera (habitualmente) intuitiva muchas tareas de preparación y visualización de datos.

1.3.1.1 Tidyverse vs Base R

Muchas de las funciones que existen en el Tidyverse tienen un equivalente en base-R (la instalación por defecto de R). El Tidyverse tiene ventajas y desventajas. La ventaja fundamental es que el código resulta (habitualmente) más fácil de leer, los nombres de las funciones son más intuitivos, y la forma de hacer las cosas tiene a ser consistente. La desventaja fundamental es que incrementamos el numero de dependencias (paquetes) de nuestro código.

Veamos un ejemplo extraído de aquí.

La misma operación con base-R o con tidyverse:

Filter rows with conditions evaluated within groups: iris flowers with maximum "Petal.Width" for each "Species"

Tidyverse

```
result1 = iris |>
  group_by(Species) |>
  filter(Petal.Width == max(Petal.Width))
```

Base-R

```
# First operate in the data.frame by group (split-apply)
widest_petals <- by(iris,
                    INDICES = iris$Species,
                    FUN = function(x){
                      x[x$Petal.Width == max(x$Petal.Width), ]
                    })

# Then combine the results into a data.frame
result2 = do.call(rbind, widest_petals)
```

```
waldo::compare(result1, result2, ignore_attr = TRUE)
#> v No differences

result1
#> # A tibble: 5 x 5
#> # Groups:   Species [3]
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1          5           3.5          1.6         0.6 setosa
#> 2          5.9         3.2          4.8         1.8 versicolor
#> 3          6.3         3.3          6           2.5 virginica
#> 4          7.2         3.6          6.1         2.5 virginica
#> 5          6.7         3.3          5.7         2.5 virginica
```

1.3.2 Antes de empezar

Programar es como tener un superpoder. Pero llegar adquirir ese superpoder es **muy difícil**. Todos necesitamos ayuda. Contar con una comunidad robusta con la que compartir, preguntar, contribuir, hace el proceso más agradable, y aumenta tus probabilidades de éxito.

Inicialmente "programaremos" usando la técnica conocida como Copy and paste programming, y poco a poco aprenderemos a descomponer el código, adaptarlo a nuestras necesidades, hasta que en algún momento lleguemos a escribir código propio. Este manual os debería servir de referencia para las primeras fases. Familiarizaros con su estructura, y acostumbraros a copiar, pegar y correr el código que aparece en los recuadros grises.

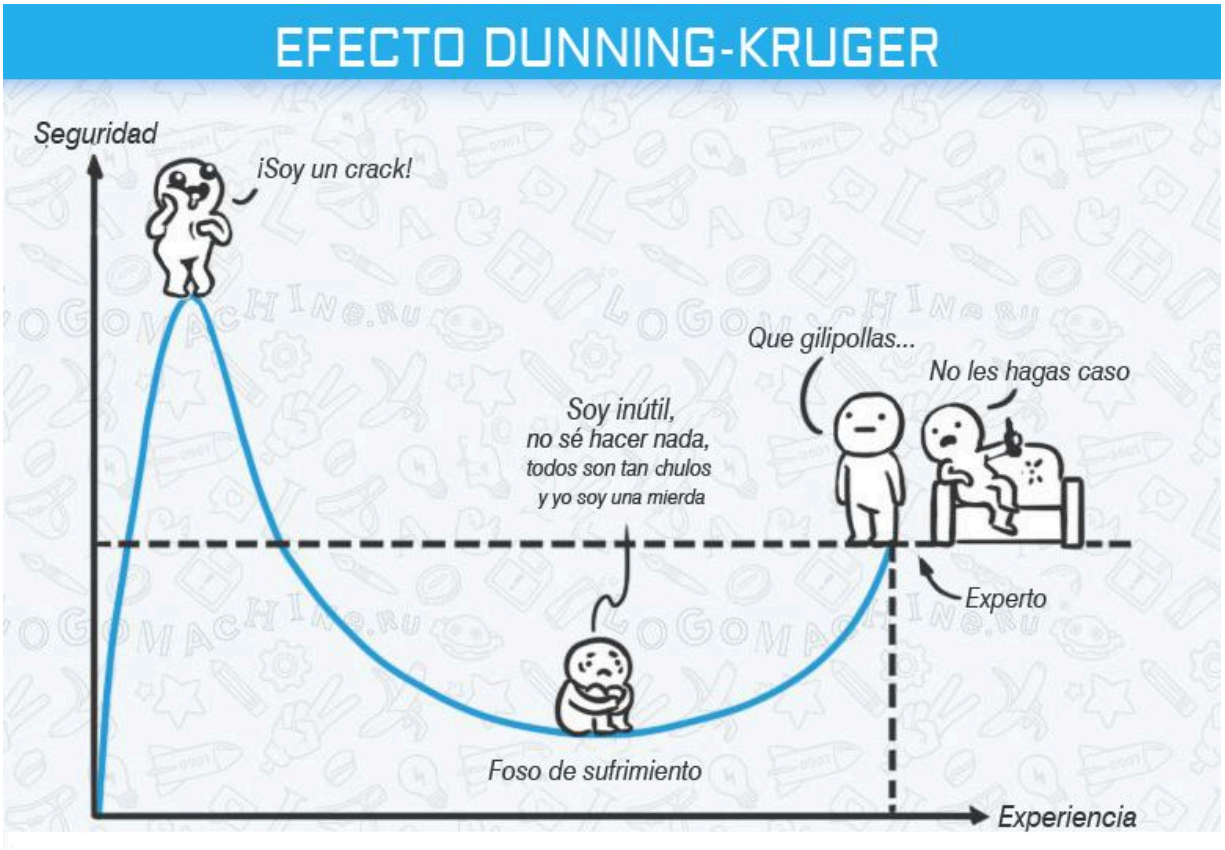


Figure 1.3: SOURCE: http://www.keywordbasket.com/ZWZlY3RvIGR1bm5pbmcta3J1Z2Vy/

1.3.3 Recursos adicionales

Hay algunos recursos que son **imprescindibles**. Nadie sabe como *los antiguos* podían programar antes de la llegada de Stackoverflow:

- **Stack overflow**
- Google: avoid scientific notation R: `options(scipen=999)`

Y otros recursos que resultan muy útiles:

- Comunidad de usuarios de Rstudio
- Mastodon! Por ejemplo:

  – @thomas_mock (#TidyTuesday)

  -  @rivaquiroga
  -  @RLadiesGlobal
  -  @coolbutuseless
- Webs como R bloggers

Bibliografía

- Matejka, J., & Fitzmaurice, G. (2017, May). Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (pp. 1290-1294). ACM.
- https://bbc.github.io/rcookbook/
- https://github.com/bbc/bbplot
- Big Book or R : https://www.bigbookofr.com/index.html

2 Introducción a la visualización de datos

! Paquetes para este capítulo

Para poder ejecutar en tu ordenador el código de los ejemplos y ejercicios de este capítulo vas a necesitar los paquetes del recuadro siguiente.
Cuando empecemos **cada capítulo**:

1) Abre un script de R y guárdalo con el nombre del capítulo: `capitulo2.R`
2) Copia las líneas de abajo (click en el icono del cuadro gris de abajo) y pégalas en el script
3) Ejecútalas: CNTRL + ENTER para ejecutar linea a linea, o CNTRL + ALT + R para ejecutar todo

```
if (!require('cowplot')) install.packages('cowplot'); library('cowplot')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('esquisse')) install.packages('esquisse'); library('esquisse')
if (!require('gapminder')) install.packages('gapminder'); library('gapminder')
if (!require('geomtextpath')) install.packages('geomtextpath'); library('geomtextpath')
if (!require('gghighlight')) install.packages('gghighlight'); library('gghighlight')
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
if (!require('ggrain')) install.packages('ggrain'); library('ggrain')
if (!require('ggthemes')) install.packages('ggthemes'); library('ggthemes')
if (!require('ggridges')) install.packages('ggridges'); library('ggridges')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('plotly')) install.packages('plotly'); library('plotly')
if (!require('purrr')) install.packages('purrr'); library('purrr')
if (!require('readr')) install.packages('readr'); library('readr')
if (!require('sjPlot')) install.packages('sjPlot'); library('sjPlot')
if (!require('tidyr')) install.packages('tidyr'); library('tidyr')
```

2.1 R para visualización de datos

ggplot2 es el paquete por excelencia para visualización de datos. Su potencia va asociada a un

nivel de complejidad considerable, hasta el punto que hay Cheat sheets oficiales, Cheat sheets buscables, y decenas de miles de preguntas en Stack Overflow.

2.1.1 Primeros pasos - con training wheels

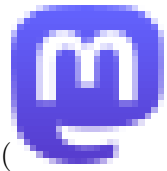
Para empezar a trabajar con ggplot sin tener que preocuparnos de su complejidad, podemos usar la función `esquisse::esquisser()` del paquete esquisse. Esta nos permite usar la potencia de ggplot para explorar una base de datos de manera muy sencilla.

See Esquisse animation

La manera fácil (1, 2, 3), usando esquisse:

```
# 1) Asegúrate que hemos instalado el paquete esquisse
  if (!require('esquisse')) install.packages('esquisse'); library('esquisse')

# 2) Lanza el wizard esquisser
  esquisse::esquisser(iris)

# 3) Crea el gráfico que quieras, exporta el código...
```

2.1.2 Aprendamos con Garrick



Garrick Aden-Buie (@grrrck) ha creado una excelente introducción a ggplot2 y la gramática de gráficos. Os recomiendo revisarla para familiarizaros con las funcionalidades de ggplot.

2.2 Visualización de datos con ggplot2

2.2.1 Componentes de una gráfica

En esta sección vamos a ver algunos de los componentes que usaremos cuando visualicemos datos. Muchos de los ejemplos que usaremos vienen de R for data science.

i Ingredientes esenciales de una gráfica

- **Aesthetic mappings** (aes): Variables, colores, rellenos, formas, ...
- **Geoms** (geom_): puntos, líneas, boxplots, ...
- **Facets** (facet_): paneles con diferentes gráficos para cada nivel de una variable categórica, ...
- **Transformaciones estadísticas**: calcular promedios, barras de error, ...

Figure 2.1: SOURCE: https://skillgaze.com/2017/10/31/understanding-different-visualization-layers-of-ggplot/

2.2.2 Mi primera gráfica en A-B-C

Para crear una gráfica con ggplot, tenemos que:

- indicar donde están nuestros datos y que mostraremos en los ejes `x` e `y`
- añadir la o las geometrías (`geoms`) que queramos

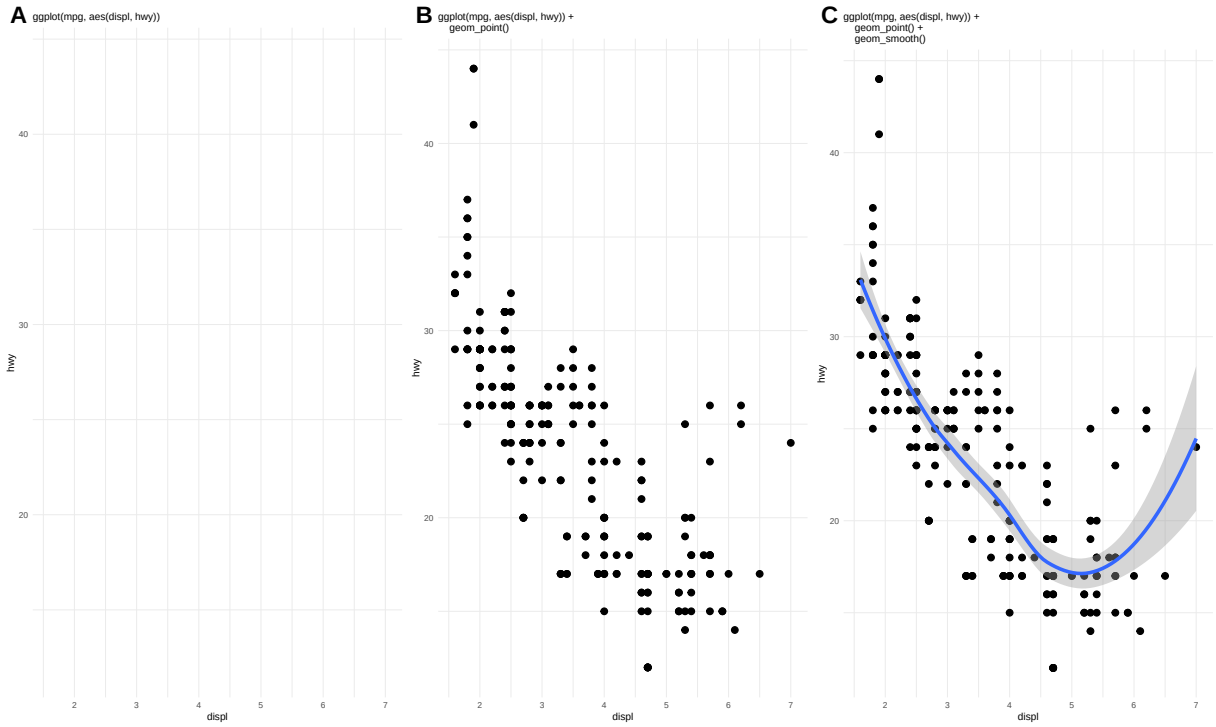
Usaremos `+` para sumar instrucciones, con una lógica de capas superpuestas.

Por ejemplo:

A) Indicamos los datos y coordenadas: `ggplot(data = mpg, aes(x = displ, y = hwy))`

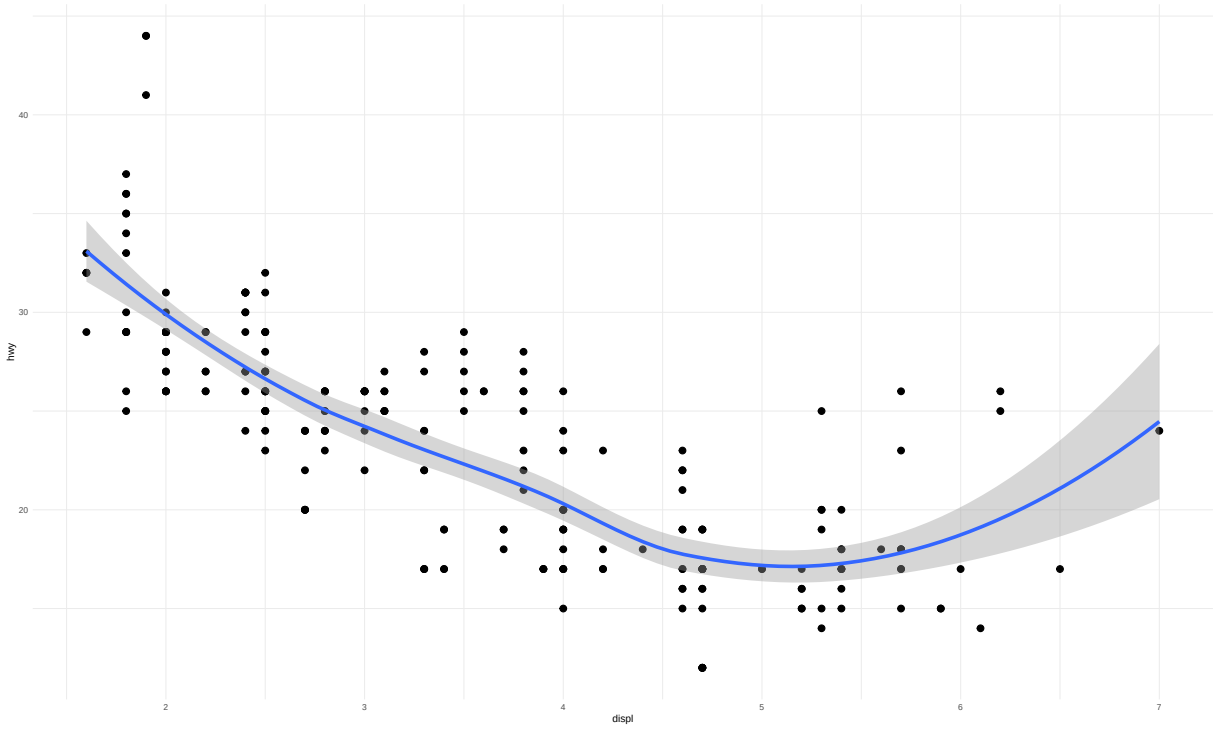
B) Añadimos el `geom` de puntos para mostrar la relación entre x e y: `+ geom_point()`

C) Añadimos un segundo `geom` para trazar una línea de tendencia: `+ geom_smooth()`



El código de la gráfica final sería el siguiente. Si respetamos el orden de las variables, podemos simplificar nuestro código, evitando el `data =`, `x =` e `y =`:

```
# Los datos están en mpg. Queremos ver la relación entre las variables `displ` y `hwy`
# Usamos geom_point para mostrar puntos
# Usamos geom_smooth para dibujar linea de tendencia

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth()
```



2.2.3 Aesthetic mappings

En **aes()** vamos a indicar las variables que queremos en los ejes `x` e `y`, el color de los puntos o líneas, el relleno de las barras, la forma de los puntos, el tipo de linea, la agrupación de los datos, etc.

> Parámetros estéticos
>
> - **x**: x = gdpPercap
> - **y**: y = lifeExp
> - **color**: color = continent; color = "red"; color = "#FAA627"
> - **fill**: fill = continent; fill = "red"; fill = "#FAA627"
> - **alpha**: alpha = continent; alpha = 0.2
> - **size**: size = continent; size = 5
> - **shape**: shape = continent; shape = 0 ver codigo de las distintas formas
> - **linetype**: linetype = continent; linetype = "dashed"
> - **group**: group = continent

Veamos algunos de los parámetros...

2.2.3.1 x-y

Algo esencial es decirle a ggplot qué queremos mostrar en los ejes `x` e `y` de nuestra gráfica.

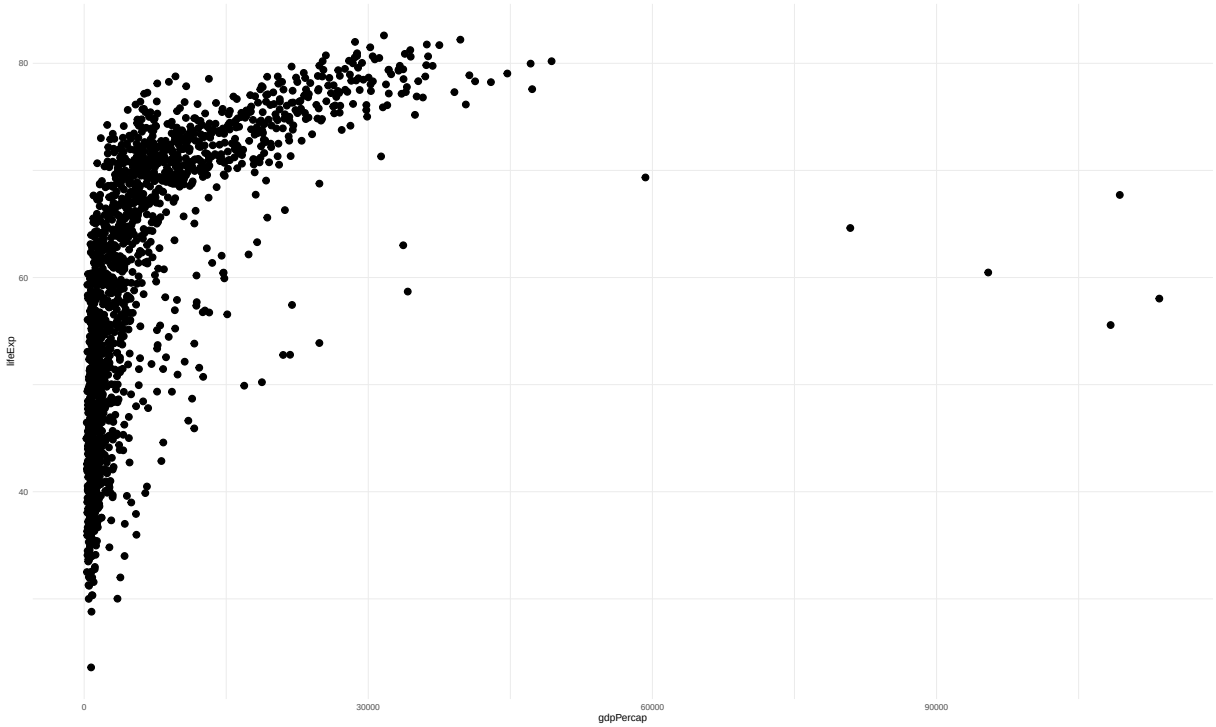
Empezaremos usando los datos de `gapminder` Vamos a ver qué variables tenemos en el data frame:

> ! Error: object 'gapminder' not found
>
> Si te aparece el error: `Error: object 'gapminder' not found`, asegurate de hacer los pasos indicados en el recuadro **Paquetes para este capítulo** arriba.

```
gapminder
#> # A tibble: 1,704 x 6
#>   country     continent  year lifeExp      pop gdpPercap
#>   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#> 1 Afghanistan Asia       1952    28.8  8425333      779.
#> 2 Afghanistan Asia       1957    30.3  9240934      821.
#> 3 Afghanistan Asia       1962    32.0 10267083      853.
#> 4 Afghanistan Asia       1967    34.0 11537966      836.
#> 5 Afghanistan Asia       1972    36.1 13079460      740.
#> 6 Afghanistan Asia       1977    38.4 14880372      786.
#> # i 1,698 more rows
```

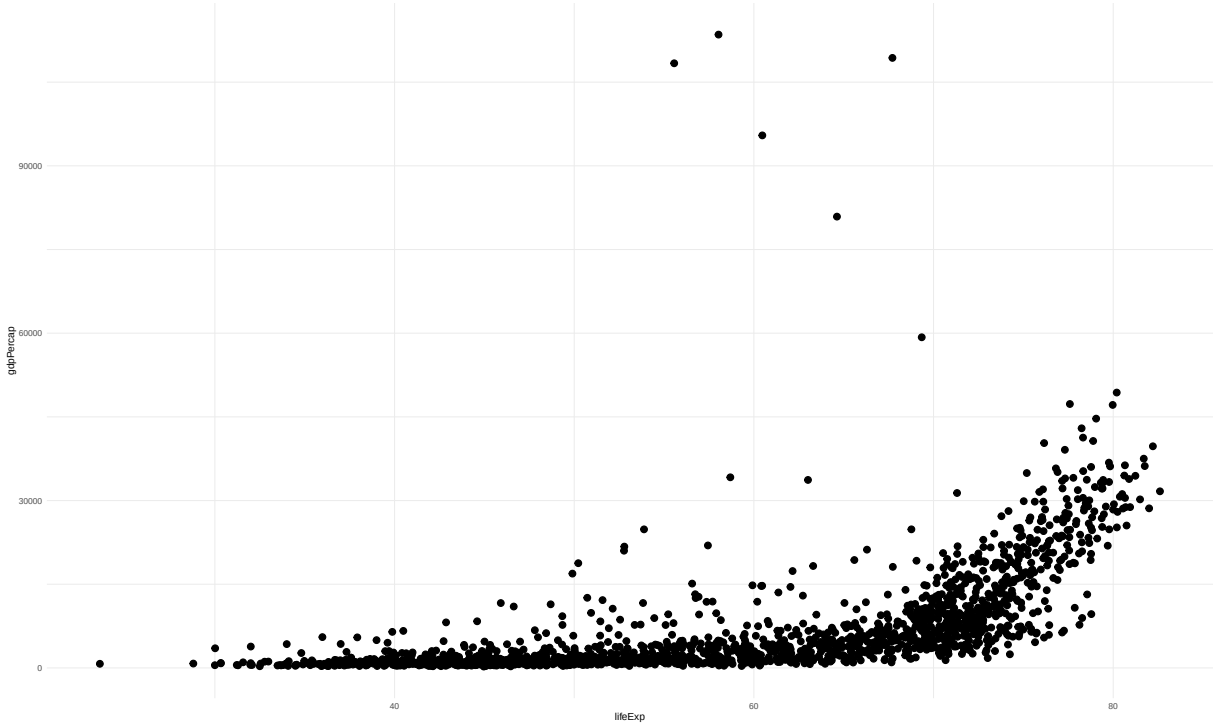
Visualizamos la relación entre gdpPercap (eje x), y lifeExp (eje y):

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point()
```



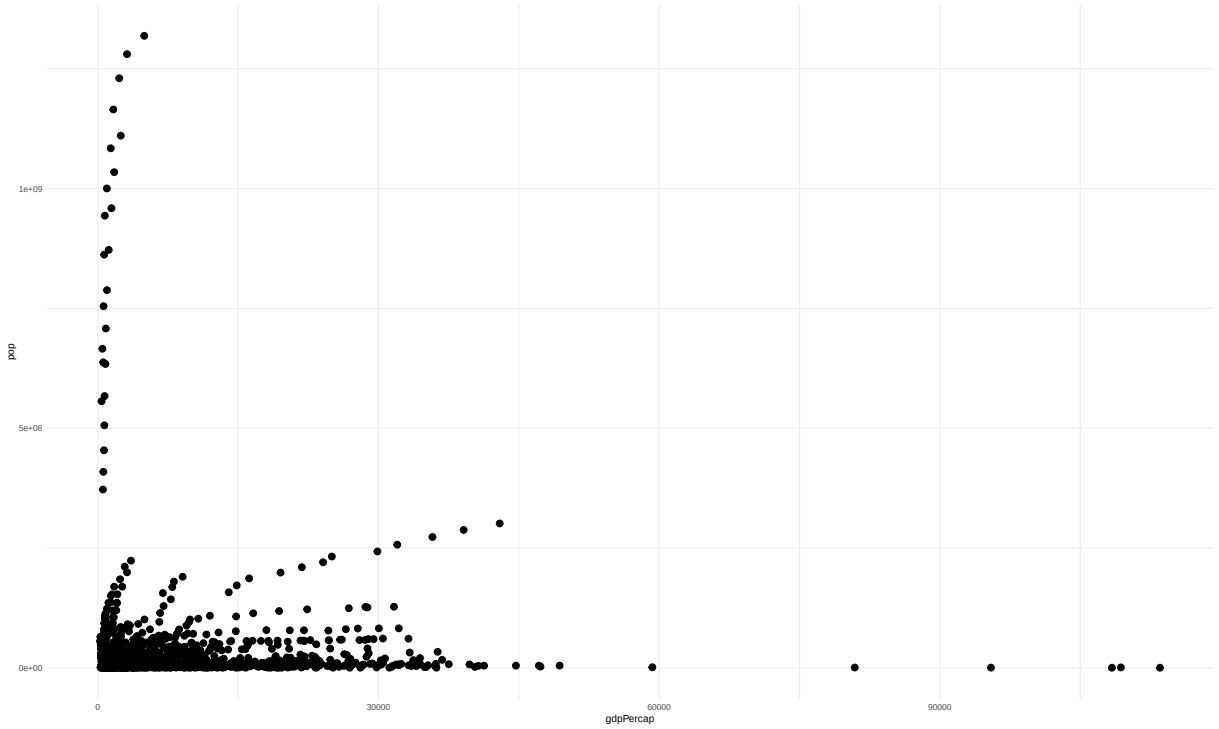
Dentro de `aes()`, el primer parámetro se refiere al eje `x` y el segundo al eje `y`. Si cambiamos el orden del código de arriba, podemos ver de nuevo la relación entre lifeExp y gdpPercap, con los ejes invertidos.

```
ggplot(gapminder, aes(lifeExp, gdpPercap)) +
  geom_point()
```



Ejercicio

Usando `gapminder`, ¿podrías crear un gráfico de gdp per cápita por población como éste?



Solución

dentro de `aes()` tenemos que poner `gdpPercap` y `pop`

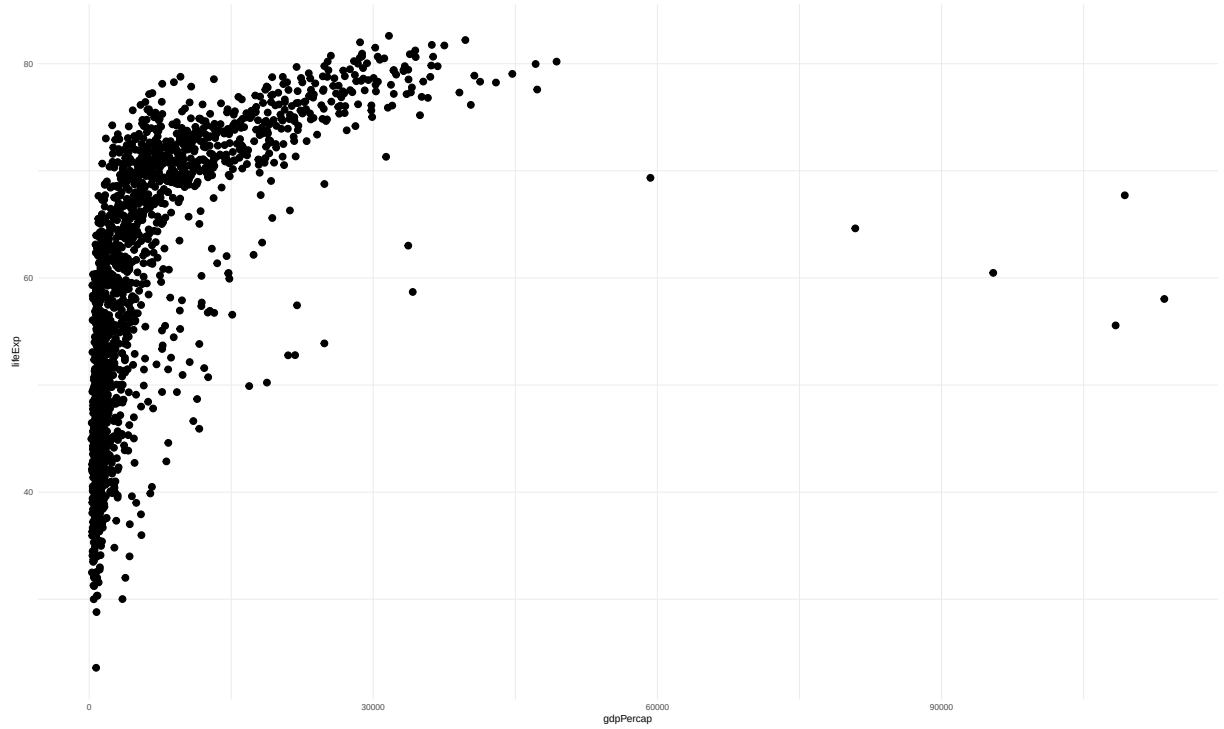
2.2.3.2 Color, alpha, size

Para asignar colores podemos usar nombres de colores en inglés, o algo llamado código HEX:

- Escribe `colors()` en la Consola de RStudio (aparecerá un listado con $>$ 600 colores)
- Ver el código HEX de los colores

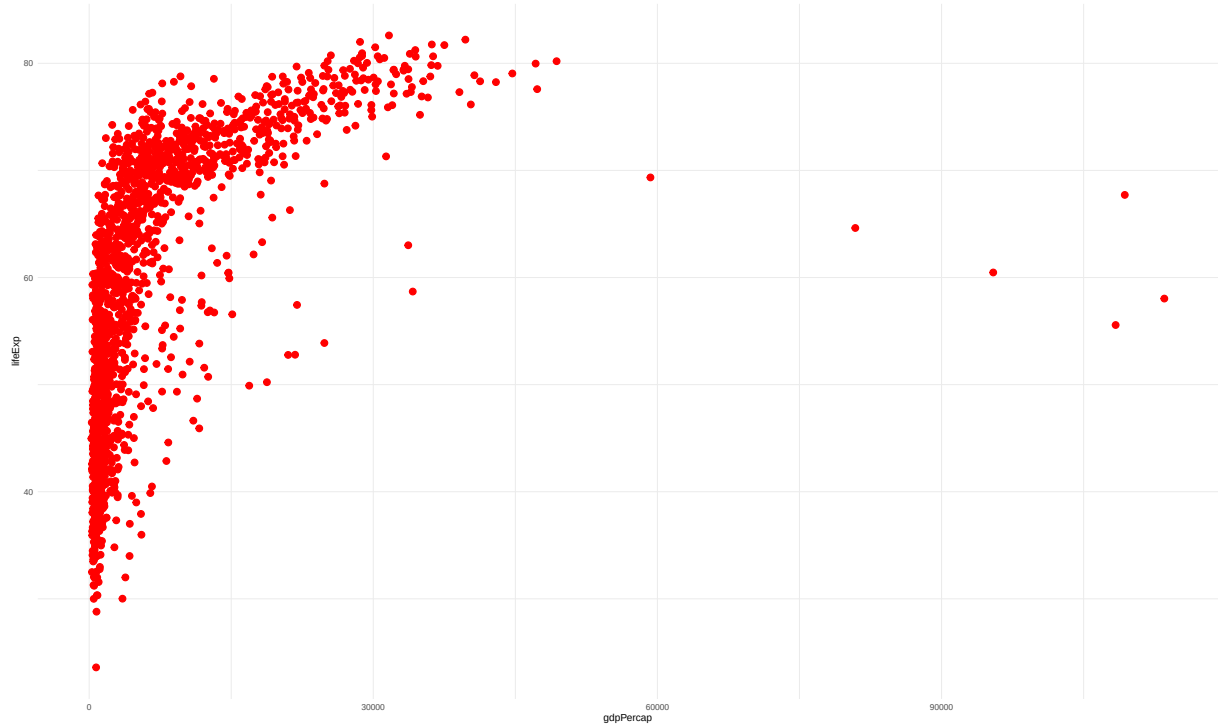
Empecemos a cambiar parámetros de nuestro gráfico inicial:

```
# Gráfico inicial
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point()
```



Color “rojo” para los puntos con `color = "red"`.

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point(color = "red")
```

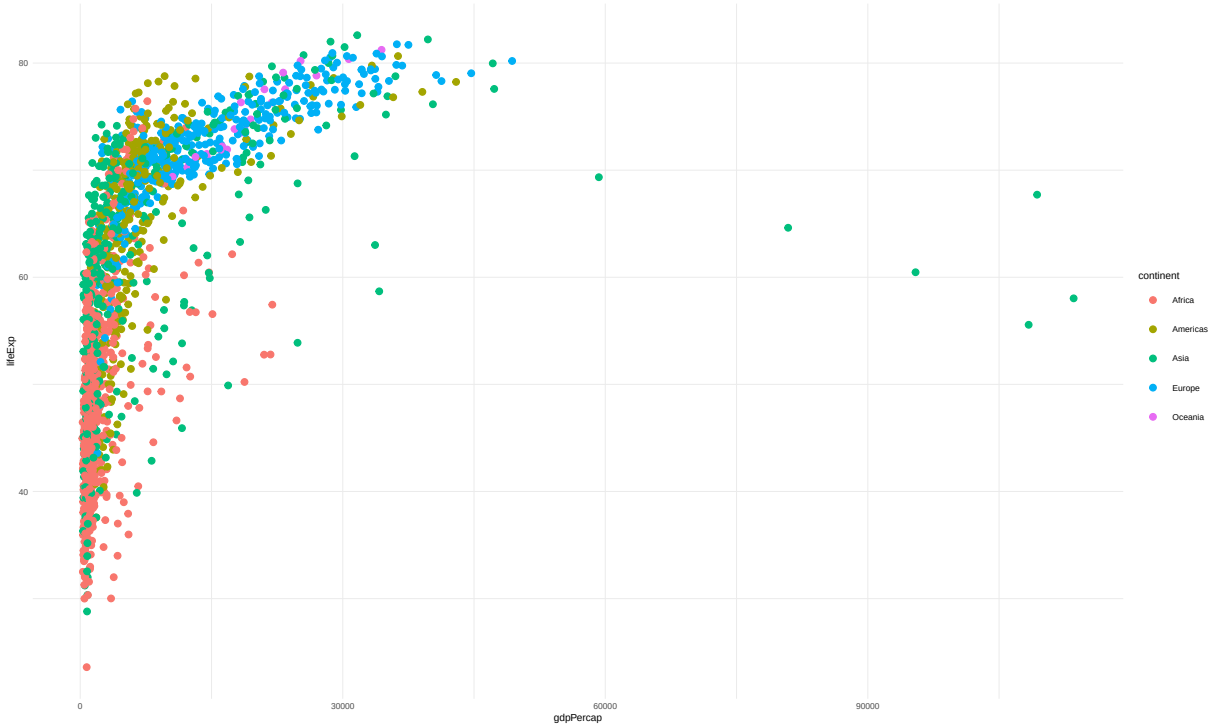


Color en función de la variable ‘continent’. Al usar un nombre de variable, tenemos que ponerlo dentro de `aes()`.

! Error: object 'continent' not found

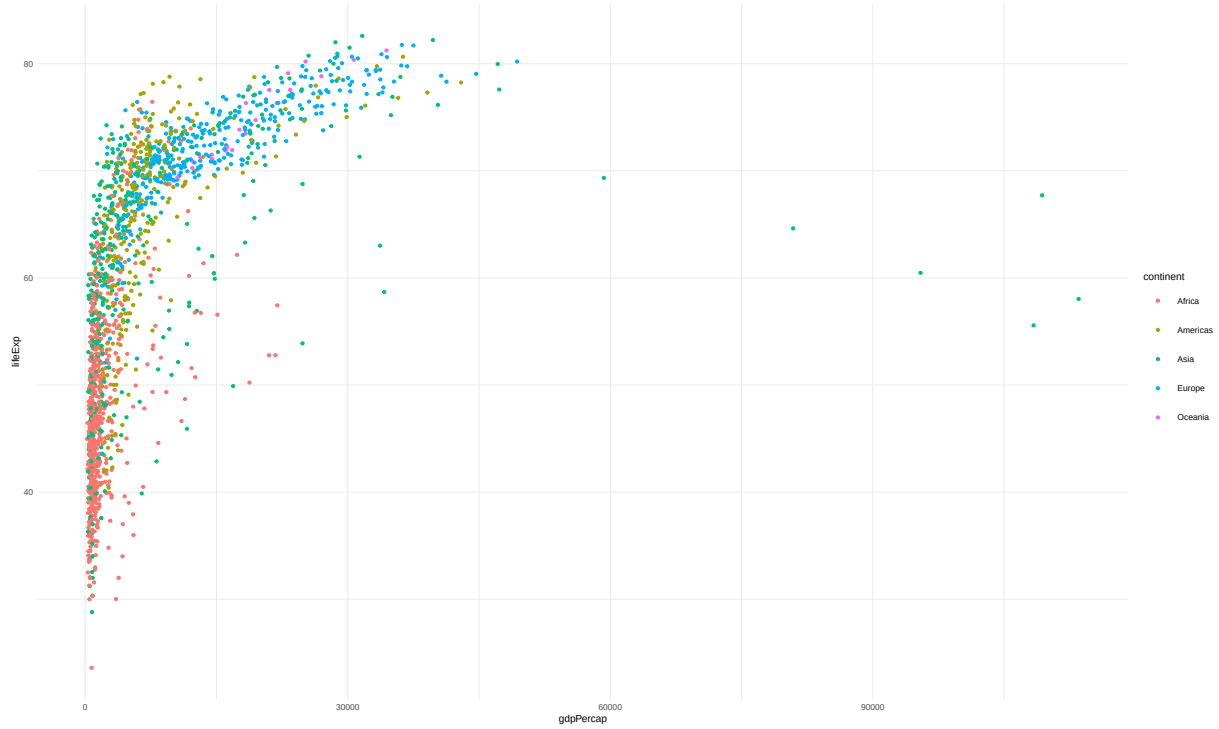
`continent` es una columna de `gapminder`, no un color. Siempre que usemos nombres de variables, tienen que estar dentro de `aes()`

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point()
```



Color en función de la variable 'continent'. Cambiamos el tamaño de los puntos a 2.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(size = .5)
```



Color en función de la variable 'continent'. Cambiamos el tamaño de los puntos a 2. Añadimos transparencia usando el parámetro 'alpha'.

```
ggplot(gapminder, aes(gdpPercap, lifeExp,
                      color = continent,
                      alpha = .1)) +
  geom_point(size = .5)
```

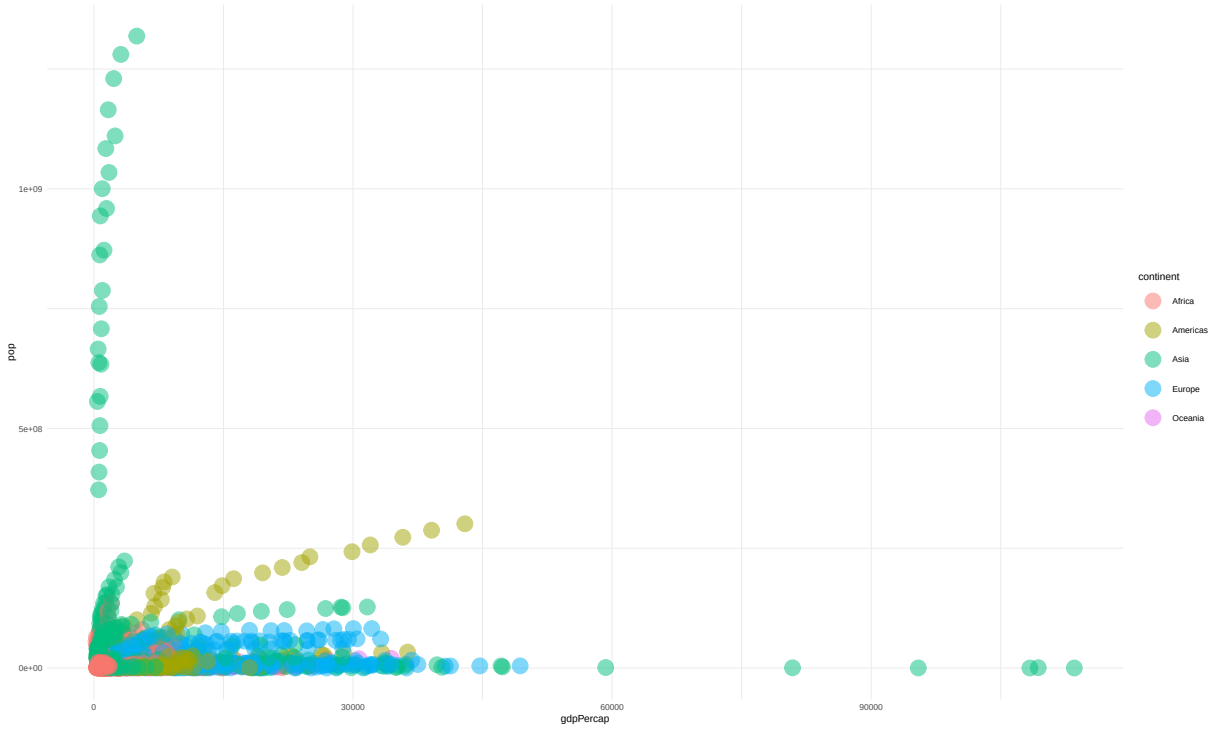
Ejercicios

Usando como base el plot siguiente (GDP x población):

```
ggplot(gapminder, aes(gdpPercap, pop)) +
    geom_point()
```

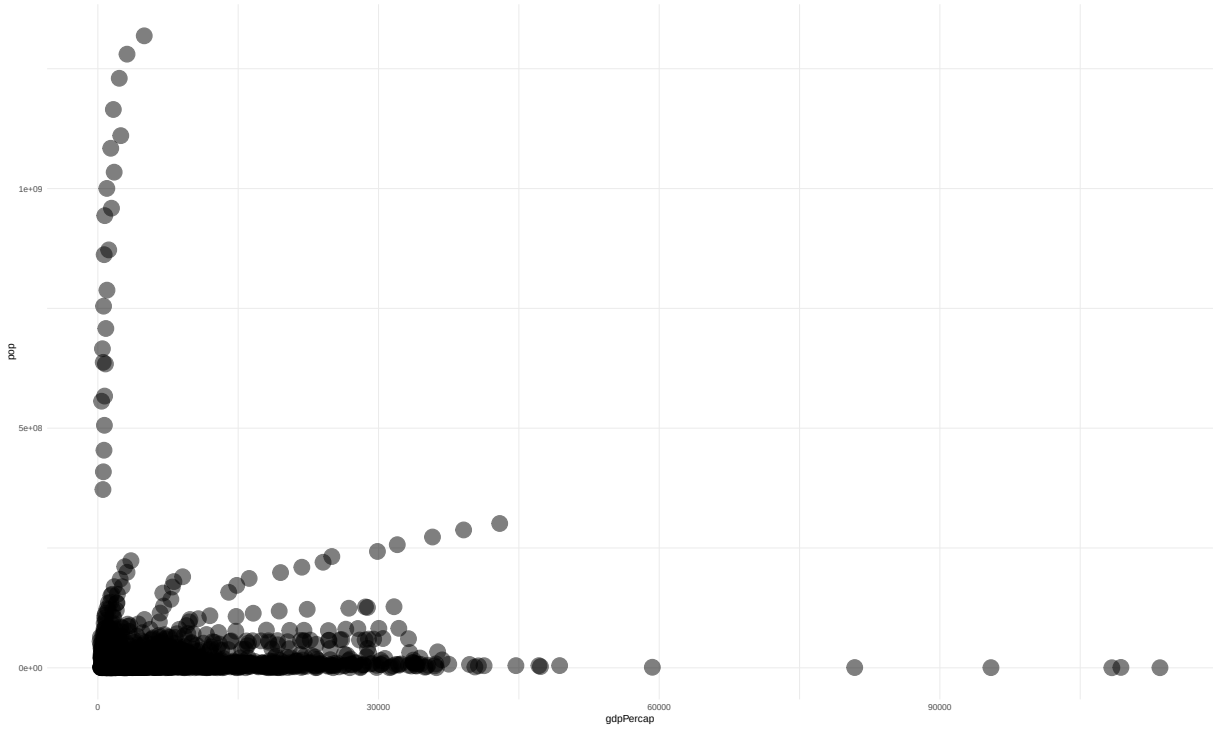
¿Podrías hacer lo siguiente?:

- Colorear los puntos por continente
- Tamaño del punto 4
- Alpha 0.5



Cada uno de los siguientes gráficos tiene un error. ¿Sabrías corregirlos?

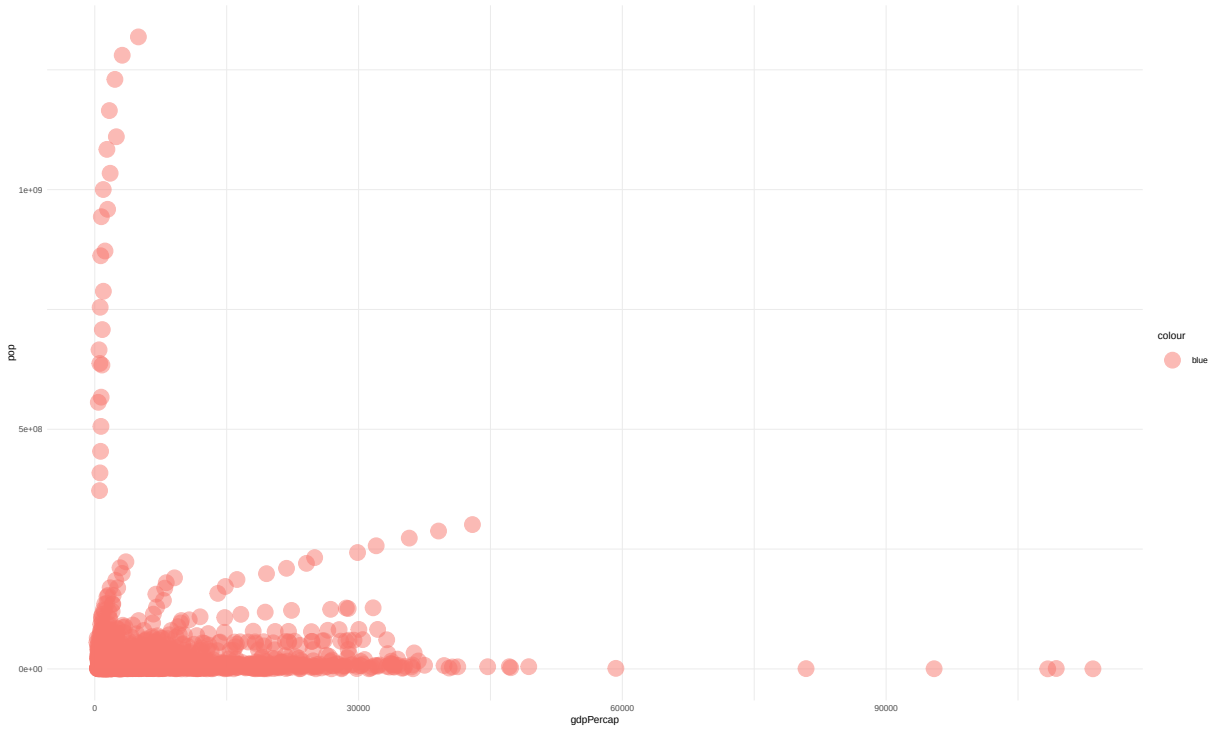
```
ggplot(gapminder, aes(gdpPercap, pop), color = continent) +
    geom_point(size = 4, alpha = .5)
```



Solución

color = continent debe ir dentro de aes()

```
ggplot(gapminder, aes(gdpPercap, pop, color = "blue")) +
    geom_point(size = 4, alpha = .5)
```



Solución

color = “blue” debe ir fuera de aes()

2.2.3.3 Shape

Códigos para las distintas formas

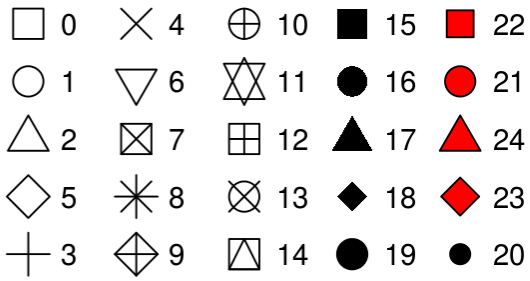
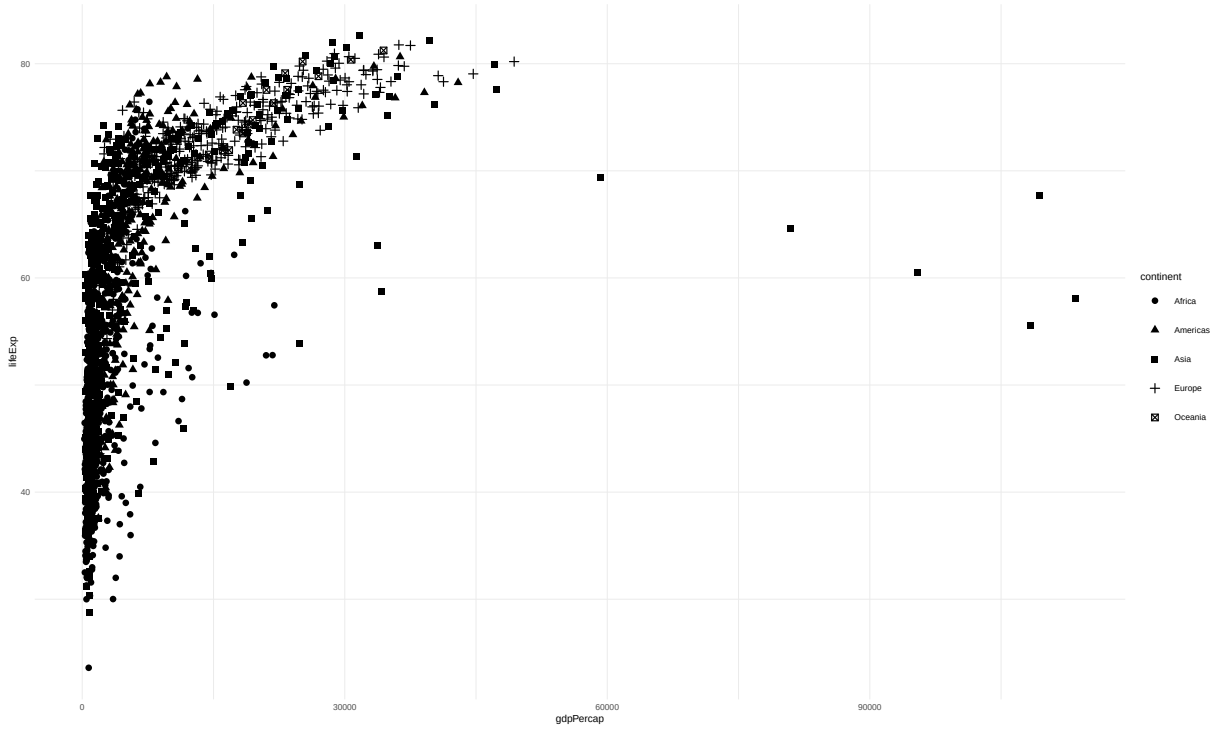


Figure 2.2: SOURCE: https://r4ds.had.co.nz/data-visualisation.html#aesthetic-mappings

En este ejemplo usamos la variable `continent` para asignar una forma diferente a cada uno de los continentes.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, shape = continent)) +
  geom_point()
```



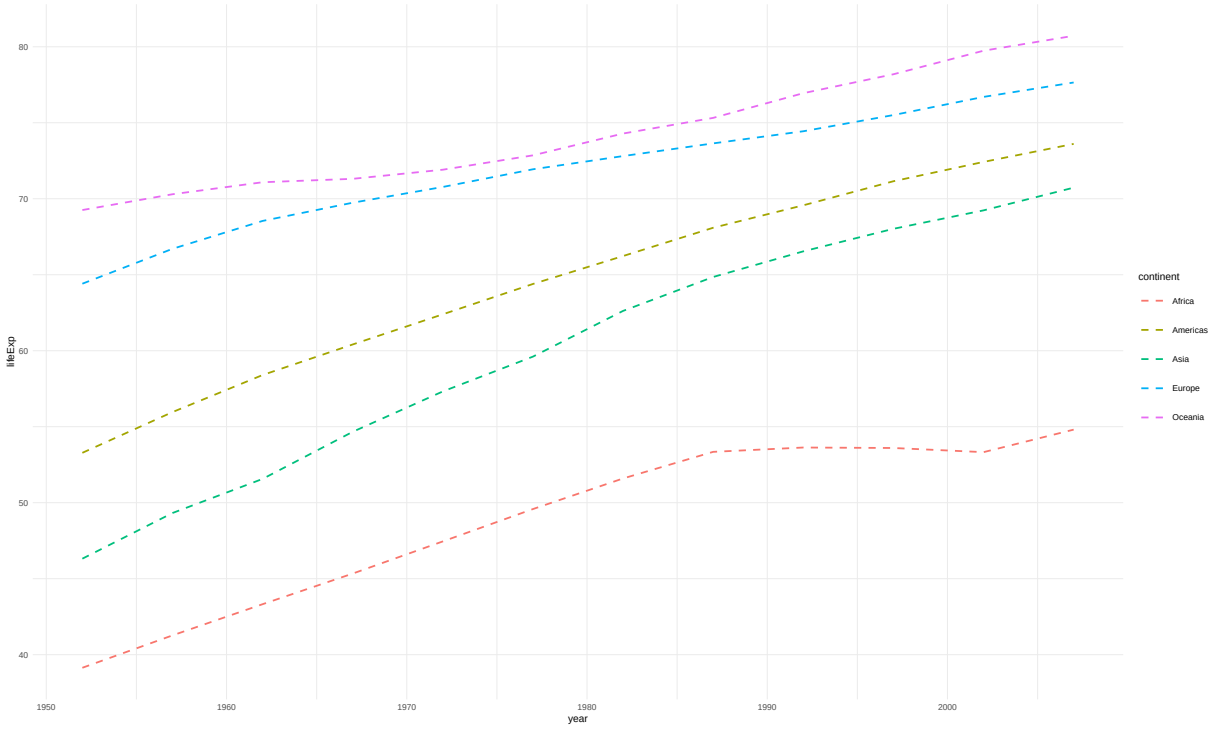
2.2.3.4 Linetype

Códigos para los distintos estilos de linea

Figure 2.3: SOURCE: http://sape.inf.usi.ch/quick-reference/ggplot2/linetype

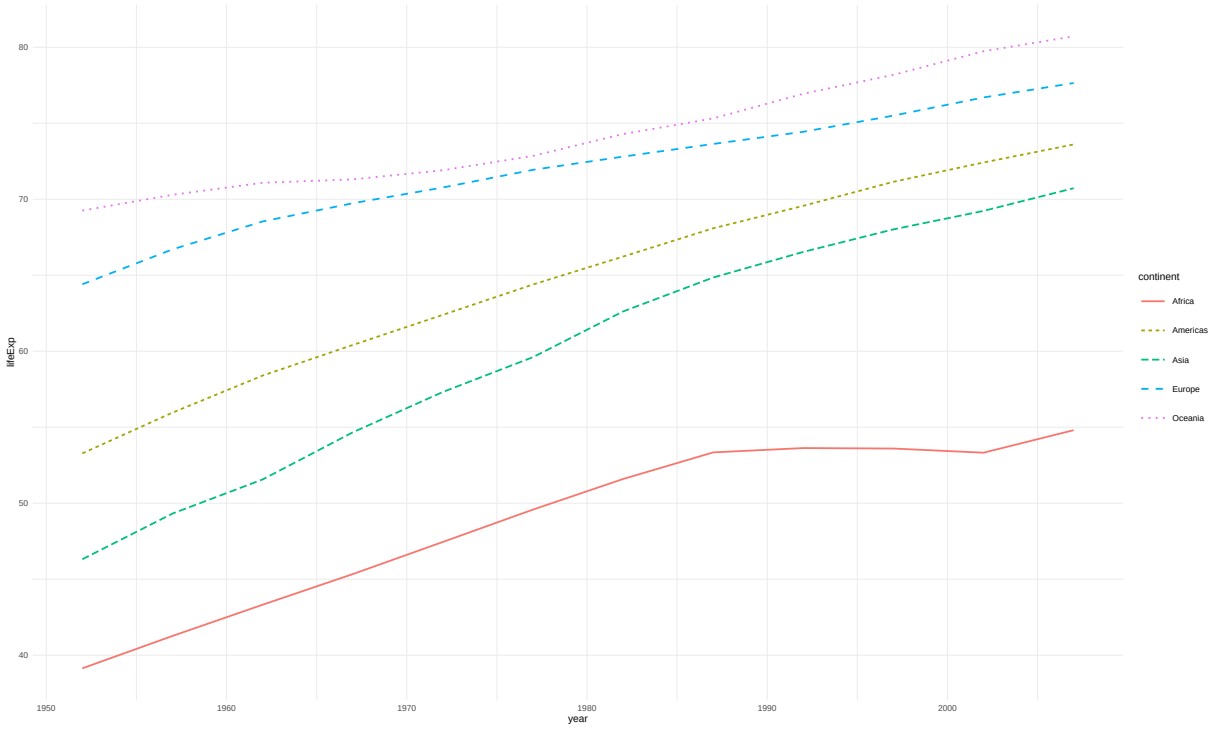
Podemos definir directamente el tipo de línea que queremos en `geom_line()`:

```
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  stat_summary(fun = mean, geom = "line", linetype = "dashed")
```



O que el tipo de línea dependa de una variable:

```
ggplot(gapminder, aes(year, lifeExp, linetype = continent, color = continent)) +
  stat_summary(fun = mean, geom = "line")
```



2.2.4 Geoms

Una de las cosas más difíciles cuando nos enfrentamos a nuevos datos es elegir el método más efectivo para visualizarlos. Hay varios recursos interesantes sobre cómo elegir una gráfica. Personalmente, para encontrar inspiración, la r graph gallery me parece un recurso fantástico.

En esta sección veremos distintos tipos de geometría, o `geoms_()`.

Algunos tipos de geoms

Para una lista exhaustiva ver el manual de ggplot2.

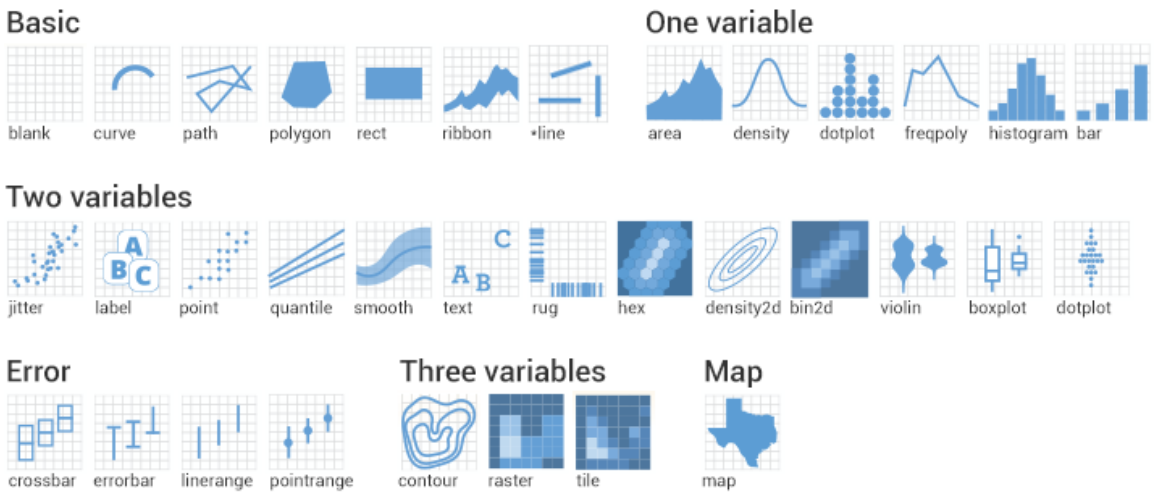
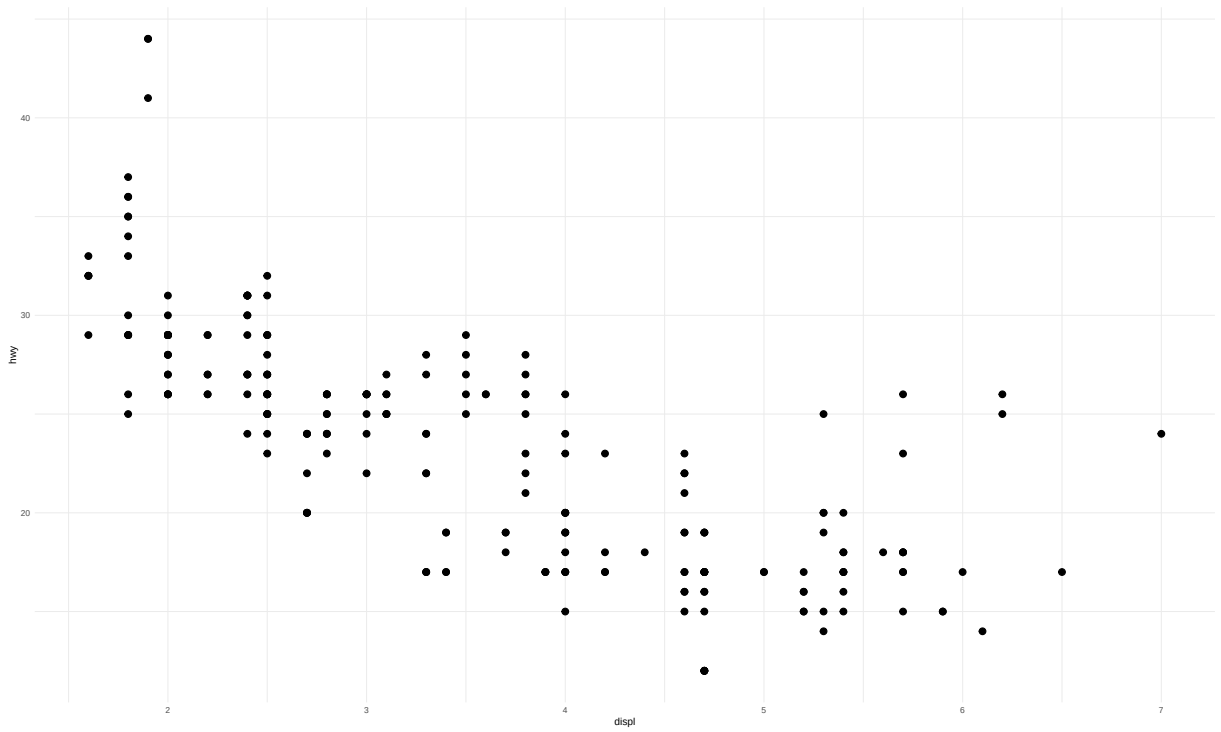


Figure 2.4: SOURCE: https://nbisweden.github.io/RaukR-2019/ggplot/presentation/ggplot_presentation_a

2.2.4.1 geom_point y geom_jitter

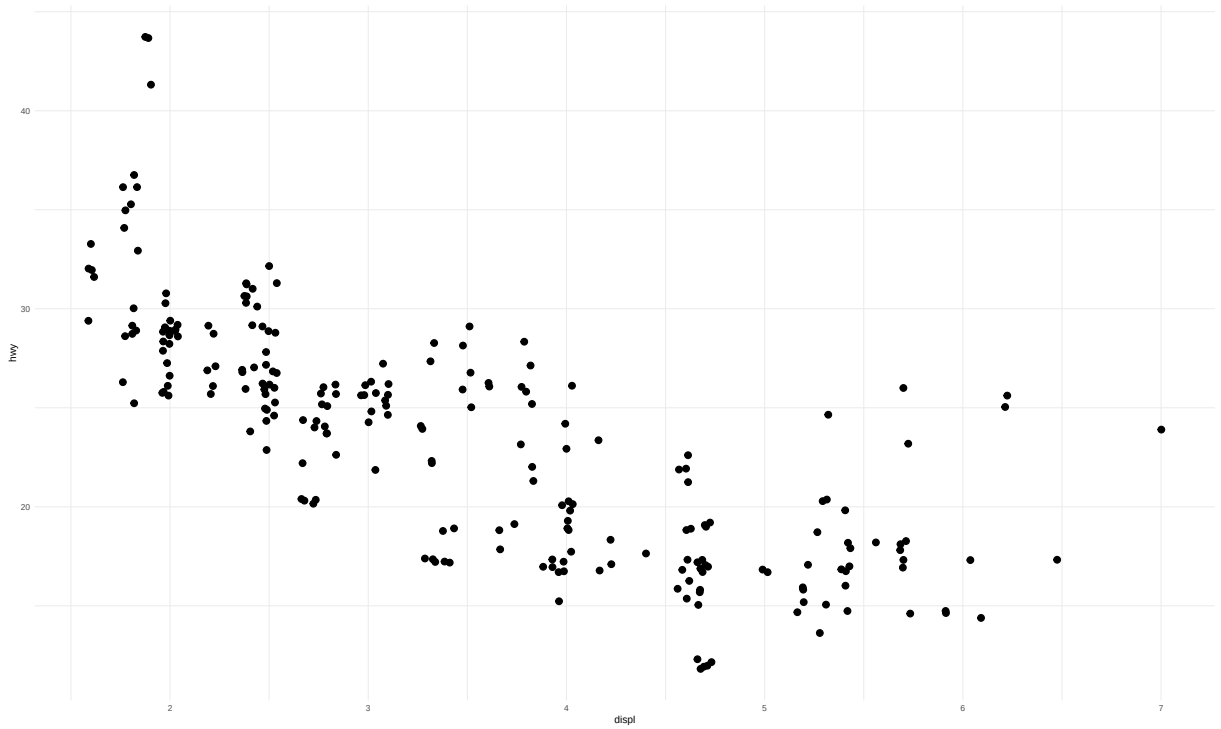
Si queremos un gráfico de dispersión o scatterplot, podemos usar el `geom_point()`

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point()
```



En algunos casos, tenemos muchos puntos que se superponen. Si usamos `geom_jitter()` la posición de los puntos cambia levemente de manera aleatoria para evitar superposiciones. Con las propiedades ´width´ y ´height´ podemos controlar el desplazamiento horizontal y vertical máximo.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_jitter()
```

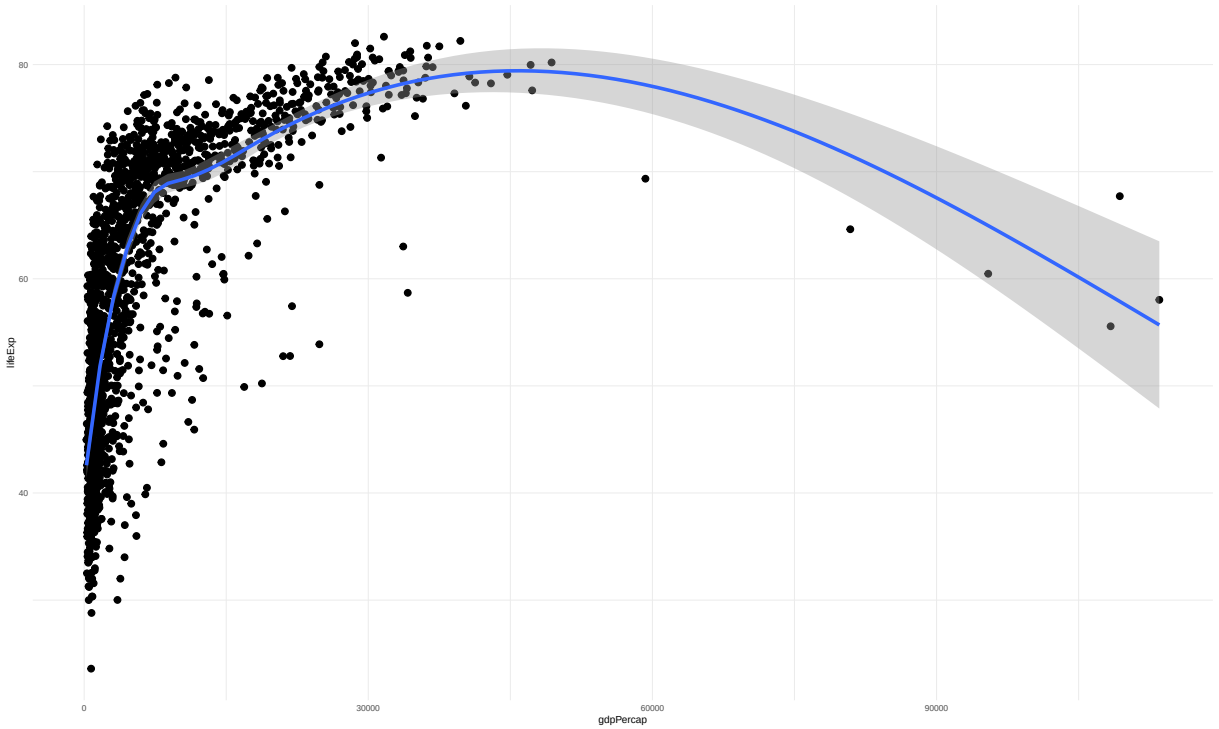


2.2.4.2 geom_smooth

Podemos añadir líneas de tendencia con `geom_smooth()`. El `method` por defecto depende de los datos. Comúnmente se usa loess, pero podemos definir explicitamente el método que queremos (e.g. `geom_smooth(method = "lm")` para usar una regresión lineal).

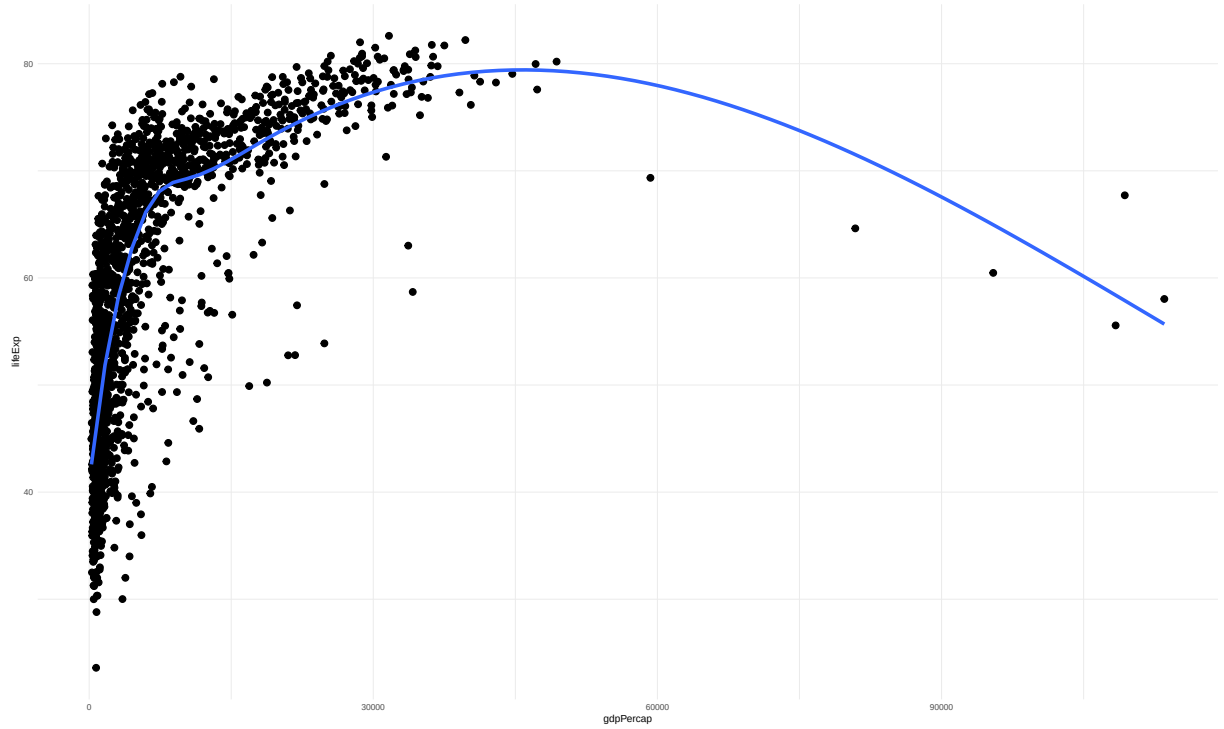
Recuerda que las funciones que usamos (todo lo que contiene `()` e.g. `geom_smooth()`) tienen parámetros, que son instrucciones adicionales que nos permiten modificar como se comportan. Para ver que opciones tenemos, podemos ver la ayuda de las funciones : `?geom_smooth()`, o poner el cursor encima y presionar F1 (ayuda).

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_smooth()
```



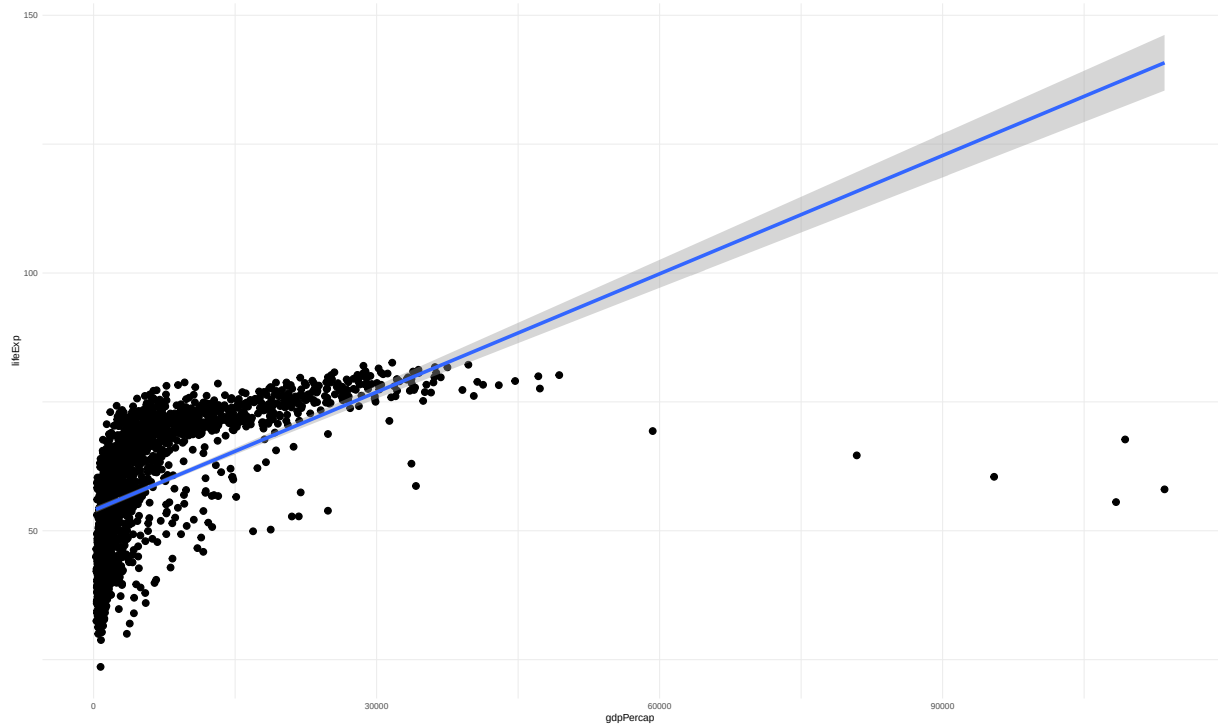
Le quitamos el intervalo de confianza con el parámetro `se`.

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_smooth(se = FALSE)
```



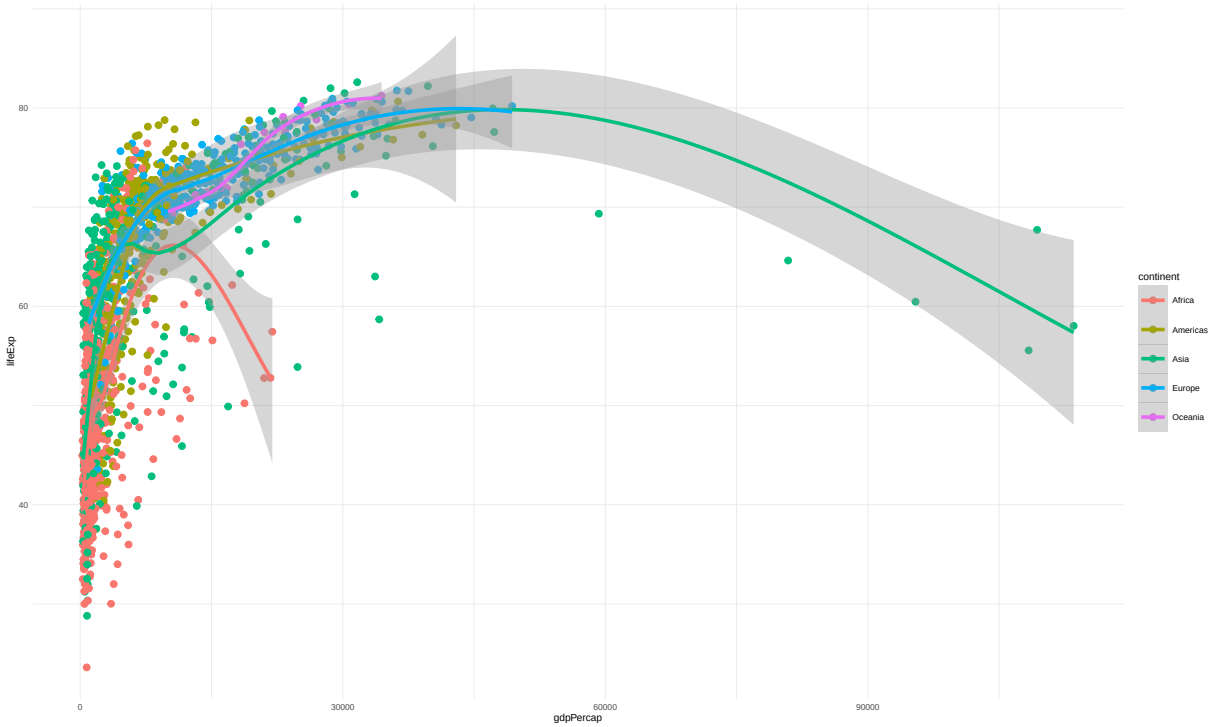
Usamos como método `lm`. En este caso, por la distribución de los datos, no parece una buena idea...

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_smooth(method = "lm")
```



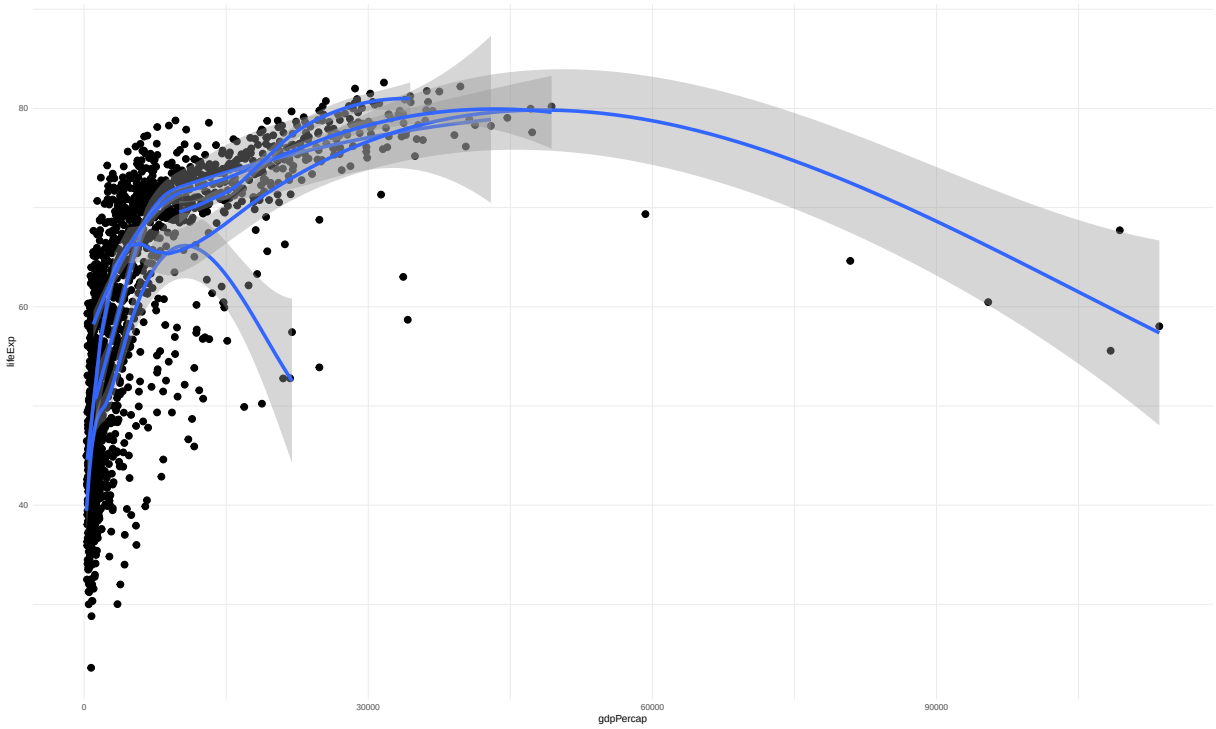
Usamos un color para cada `continent`. Al indicarlo en el `aes()` general, esto se aplica a todos los geoms (geometrías).

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point() +
  geom_smooth()
```



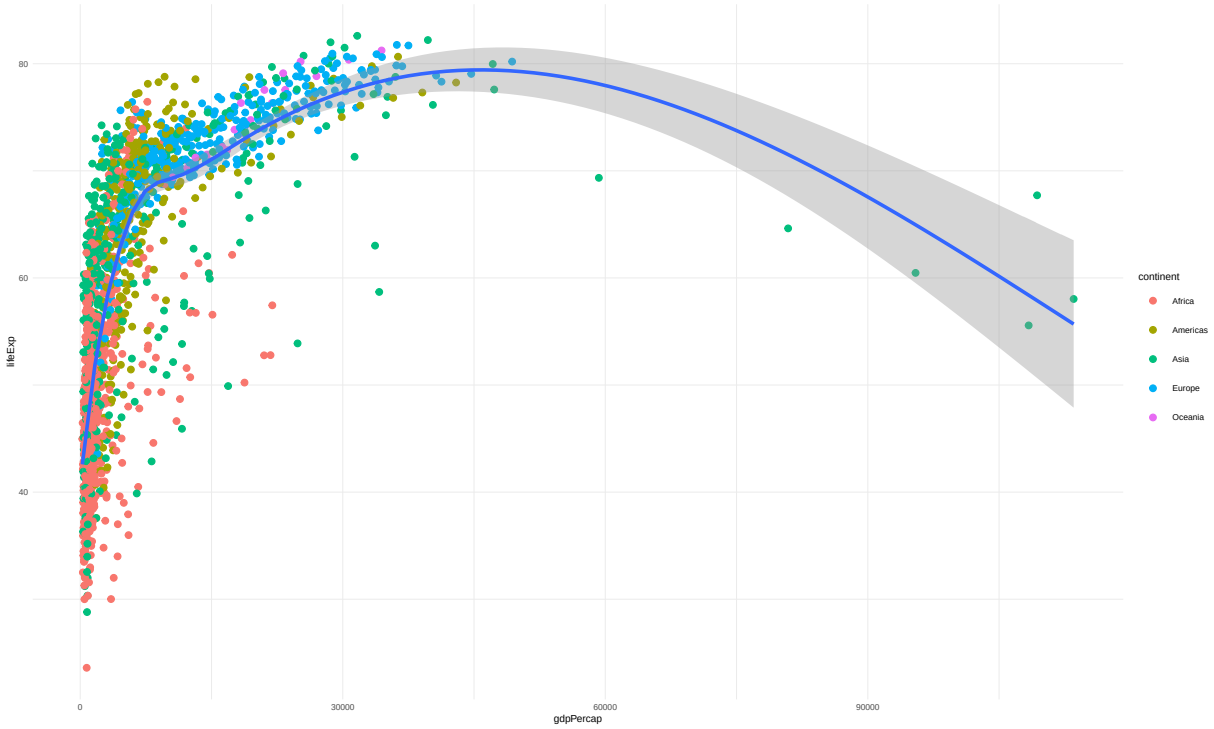
Un smooth por cada continente (group = `continent`). Dado que agrupar los puntos por continente no tiene efecto, solo se ven las múltiples líneas de tendencia.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, group = continent)) +
  geom_point() +
  geom_smooth()
```



Coloreamos puntos pero mantenemos un solo smooth introduciendo el parámetro `aes(color = continent)` dentro de `geom_point()`. El lugar donde ponemos `aes(color = continent)` determina a que geometrías afecta.

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point(aes(color = continent)) +
  geom_smooth()
```



Ejercicios

Usando como base el siguiente plot:

```
ggplot(gapminder, aes(gdpPercap, lifeExp, shape = continent)) +
  geom_point()
```

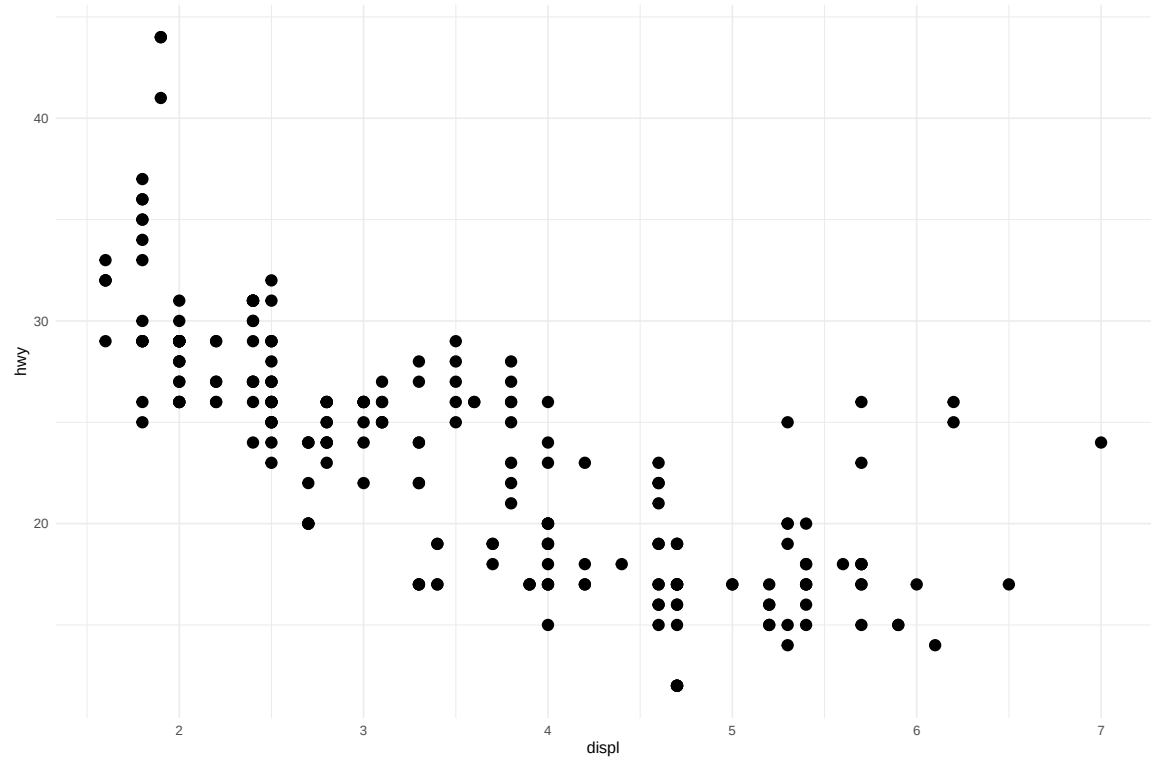
- Colorea los puntos por continente
- Muestra una línea de tendencia por continente (sin el intervalo de confianza)
- Haz que el tipo de línea cambie por continente
- Añade transparencia a los puntos para que las líneas destaquen

Solución

- parámetro `color`
- geom `geom_smooth`, parámetro `se`
- parámetro `linetype`
- parámetro `alpha` dentro de `geom_point`

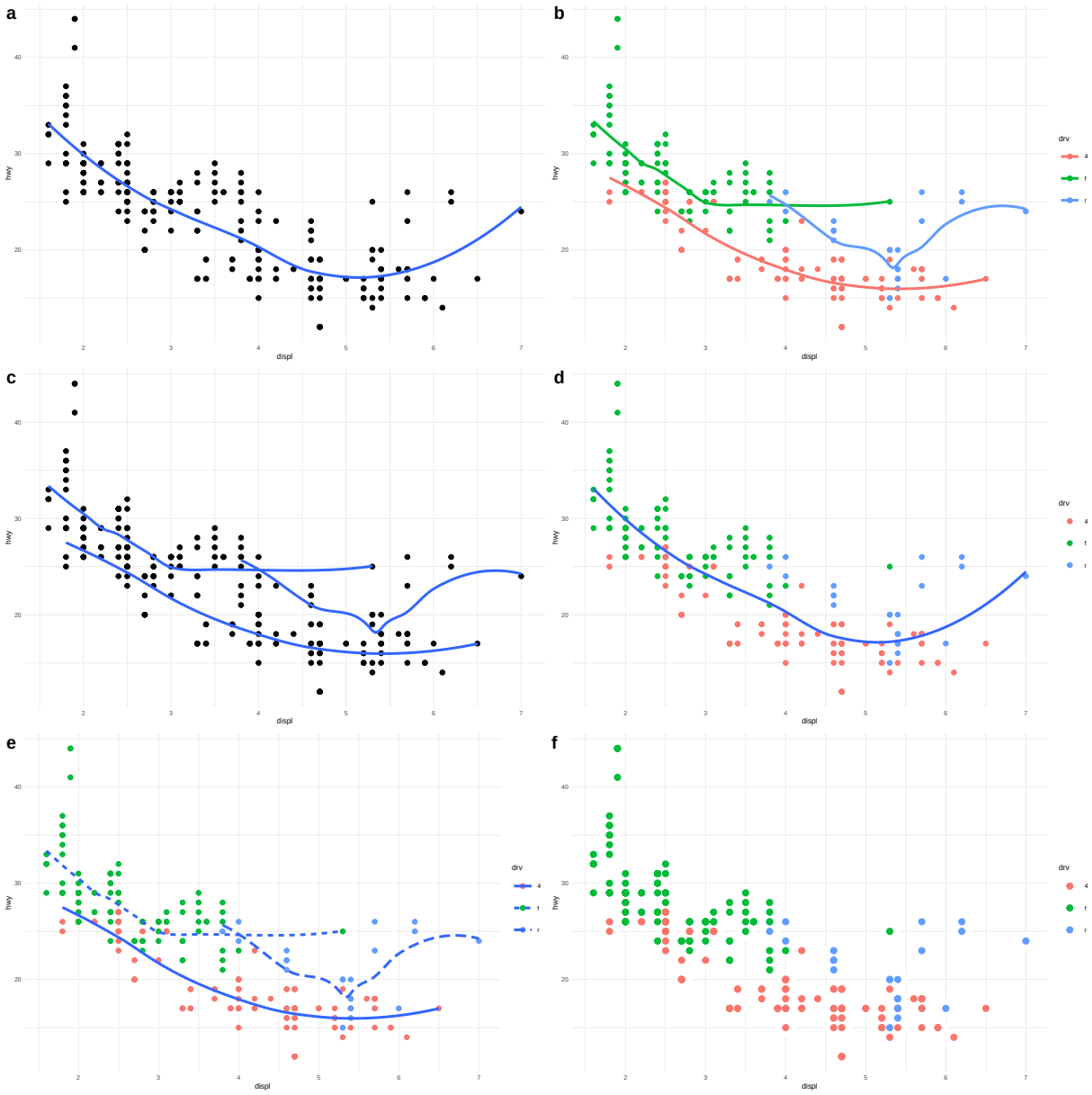
Ahora vamos a usar el data frame `mpg`. Empieza con este plot:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point()
```



Finalmente, intenta crear los siguientes 6 plots. Te recomiendo avanzar en orden alfabètico.

Para conseguirlo vamos a tener que usar parámetros como `group`, `color` o `linetype`, pensando muy bien si los tenemos que poner en el `aes()` general, o en un `aes()` dentro de geoms específicos:



Solución. No me mires hasta haberlo intentado mucho!

a) geom `geom_smooth`, parámetro `se`

b) `group = drv` dentro de `aes()`

c) parámetro `color` dentro del `aes()` general

d) parámetro `color` dentro del geom

e) parámetro `linetype`

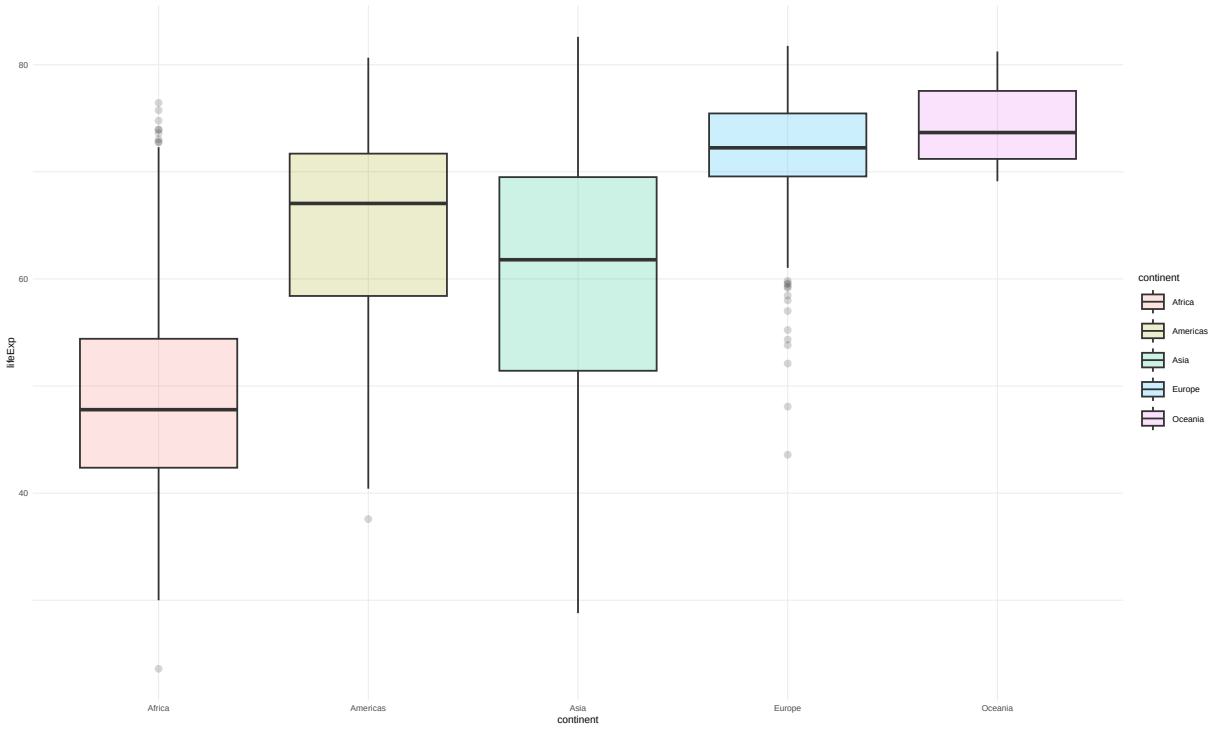
f) x2 `geom_point`

2.2.4.3 geom_boxplot y geom_violin

Podemos crear diagramas de cajas (boxplots) con `geom_boxplot` o violines con `geom_violin` para visualizar como cambian los datos por grupo.

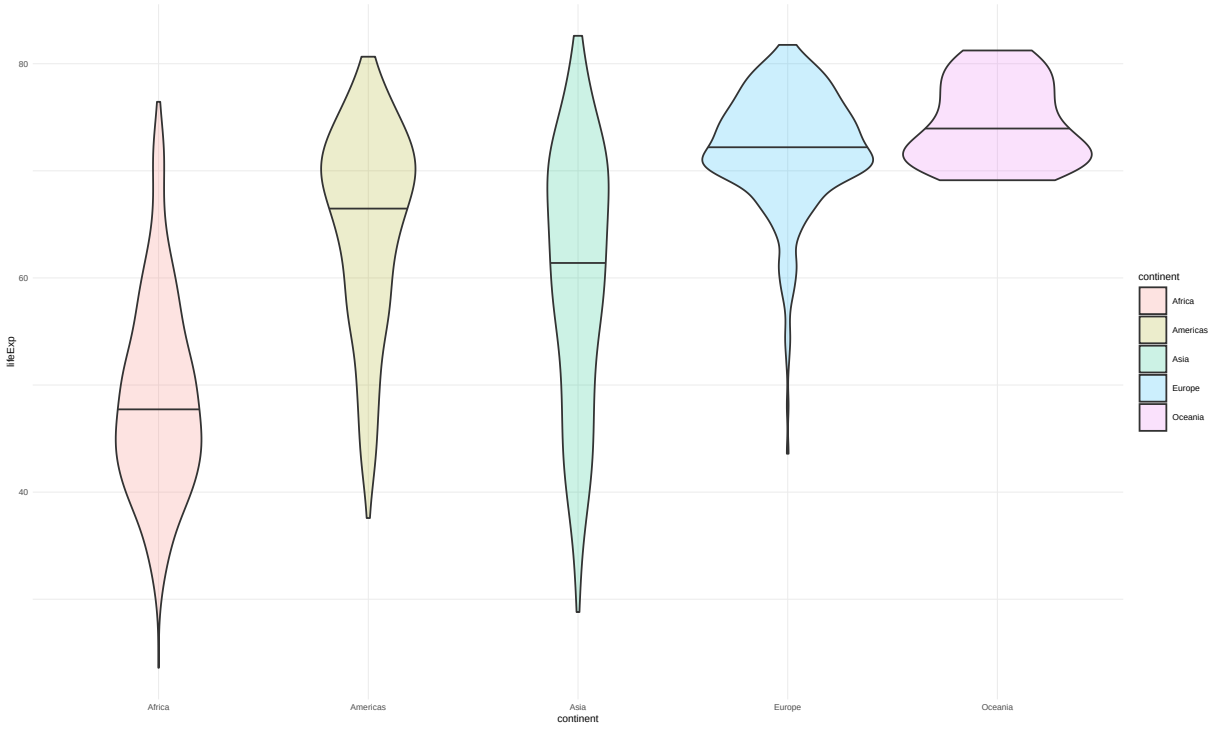
Boxplot con relleno (parámetro `fill`).

```
ggplot(gapminder, aes(continent, lifeExp, fill = continent)) +
  geom_boxplot(alpha = .2)
```



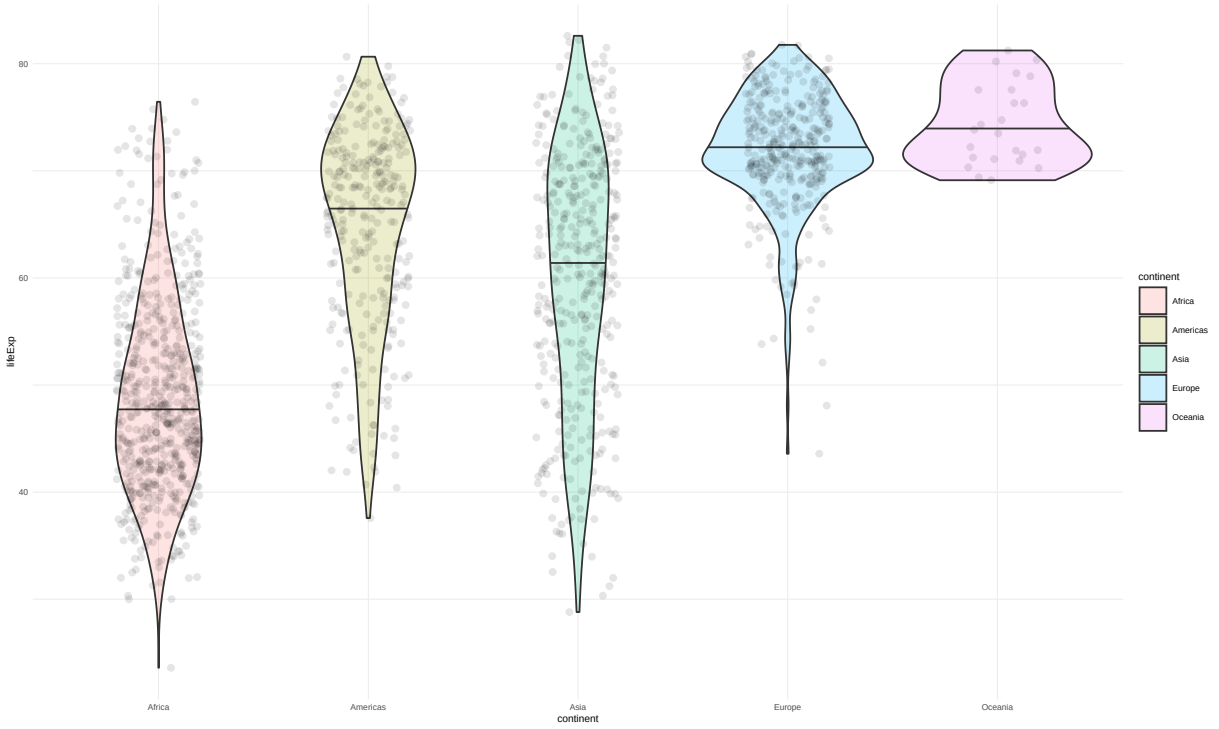
Los violin plots nos permiten ver la distribución de los datos. Podemos usar el parámetro `draw_quantiles` para dibujar quantiles, por ejemplo, el percentil 50 (la mediana). Si queremos mostrar los percentiles 25, 50 y 75, tenemos que usar `draw_quantiles = c(.25, .5, .75)`.

```
ggplot(gapminder, aes(continent, lifeExp, fill = continent)) +
  geom_violin(alpha = .2, draw_quantiles = .5)
```



Podemos combinar `geom_violin` con `geom_jitter` para mostrar las observaciones individuales. Si usamos `height = 0` y `width = .2`, los puntos mostraran el valor exacto de `lifeExp`, y se dispersarán algo en el eje horizontal.

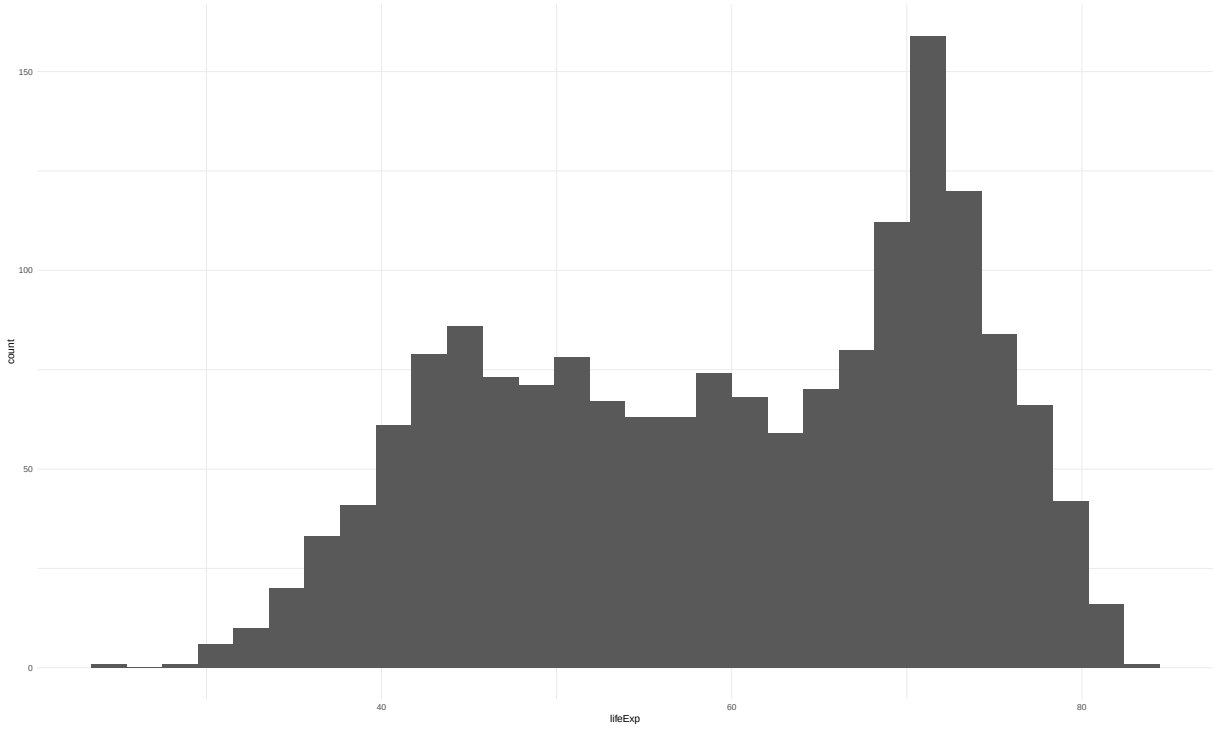
```
ggplot(gapminder, aes(continent, lifeExp)) +
  geom_violin(alpha = .2, aes(fill = continent), draw_quantiles = .5) +
  geom_jitter(alpha = .1, height = 0, width = .2)
```



2.2.4.4 geom_histogram y geom_bar

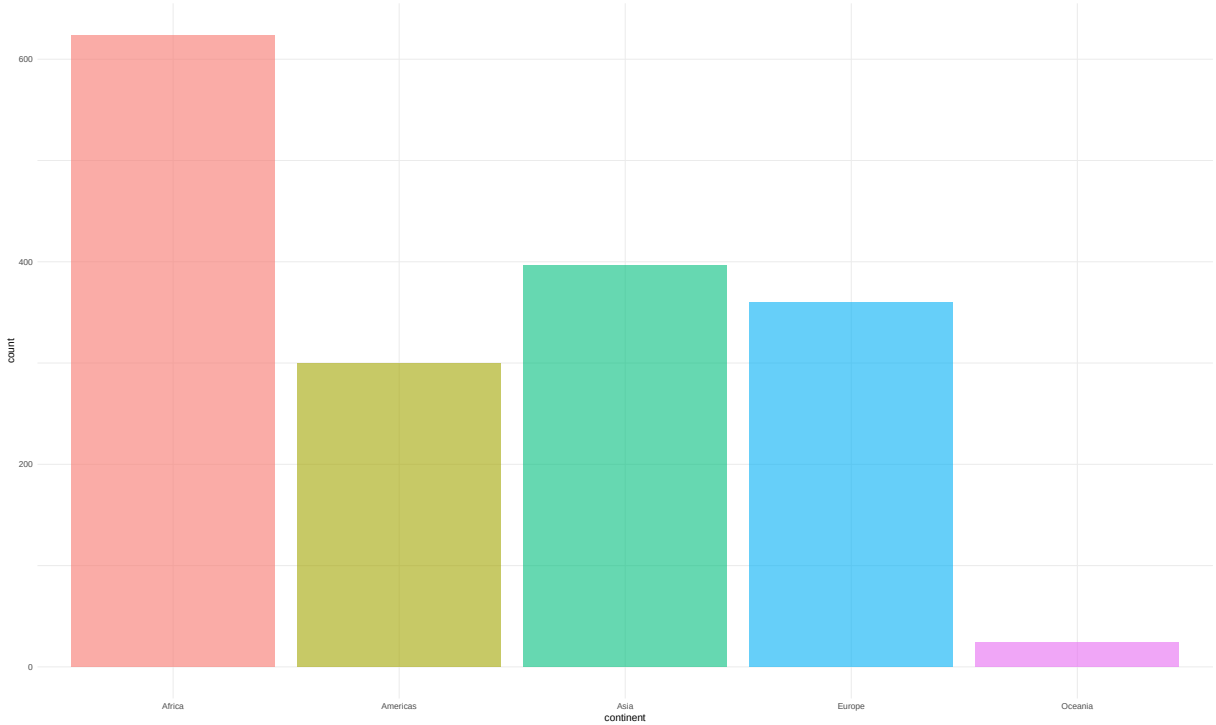
Cuando queremos visualizar la distribución de variables continuas, podemos usar histogramas (`geom_histogram()`). Como puedes ver, ahora solo le pasamos una variable a `aes()` (el eje `y` muestra el número de observaciones, y es calculado por ggplot).

```
ggplot(gapminder, aes(lifeExp)) +
  geom_histogram()
```



Si tenemos variables categóricas, usamos `geom_bar()`. Podemos usar `guides(fill = "none")` para que desaparezca la leyenda asociada al color, porque los nombres de cada categoría ya aparecen en el eje `x`:

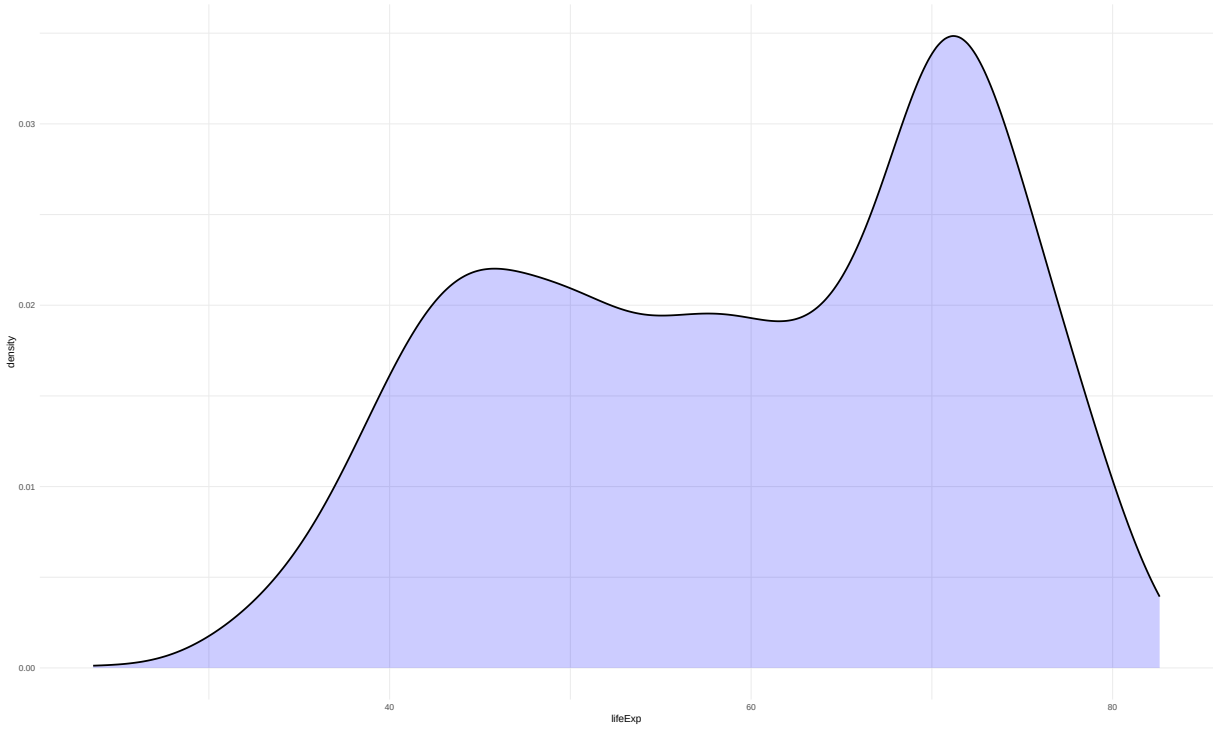
```
ggplot(gapminder, aes(continent, fill = continent)) +
  geom_bar(alpha = .6) +
  guides(fill = "none")
```



2.2.4.5 geom_density

Para visualizar distribuciones cuando tenemos muchos datos, podemos usar `geom_density()`. Eso sí, recuerda que con pocos datos, los gráficos de densidad nos dan una falsa seguridad sobre la forma de nuestra distribución.

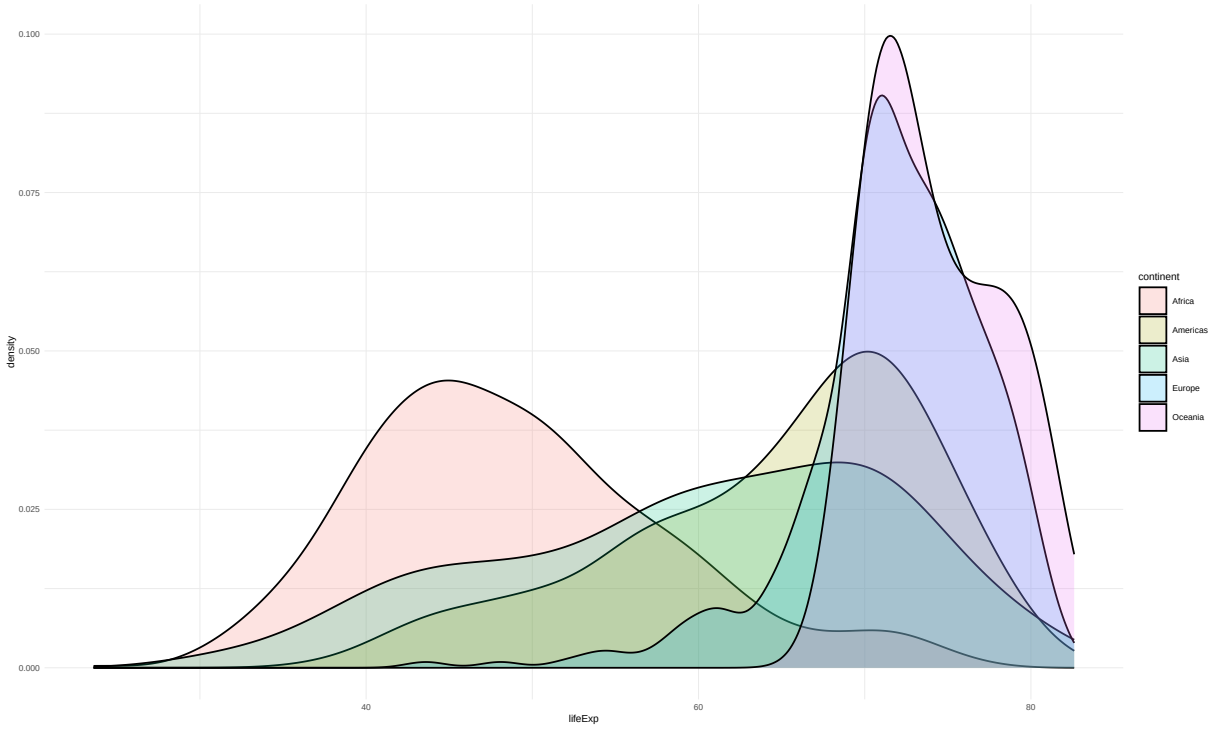
```
ggplot(gapminder, aes(lifeExp)) +
    geom_density(alpha = .2, fill = "blue")
```



Si queremos ver la distribución por continente, usamos el parámetro `fill`.

Con `alpha = .2` podemos añadir transparencia y ver todas las distribuciones. Puedes probar cambiando su valor a 1, para ver que ocurre (`alpha` puede tener valores de 0 a 1).

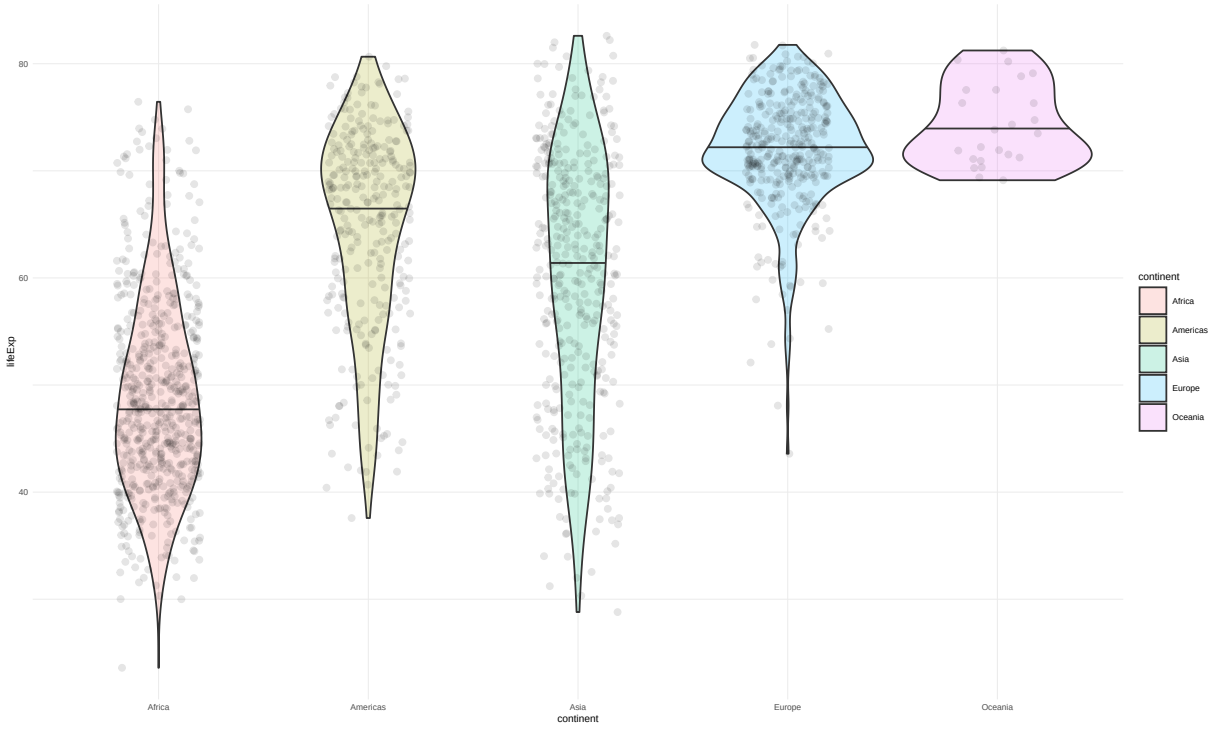
```
ggplot(gapminder, aes(lifeExp, fill = continent)) +
    geom_density(alpha = .2)
```



Ejercicios

En sección geom_boxplot y geom_violin vimos este gráfico:

```
ggplot(gapminder, aes(continent, lifeExp)) +
    geom_violin(alpha = .2, aes(fill = continent), draw_quantiles = .5) +
    geom_jitter(alpha = .1, height = 0, width = .2)
```



Como puedes ver, `aes(fill = continent)` está en `geom_violin()`, no en `ggplot()`.

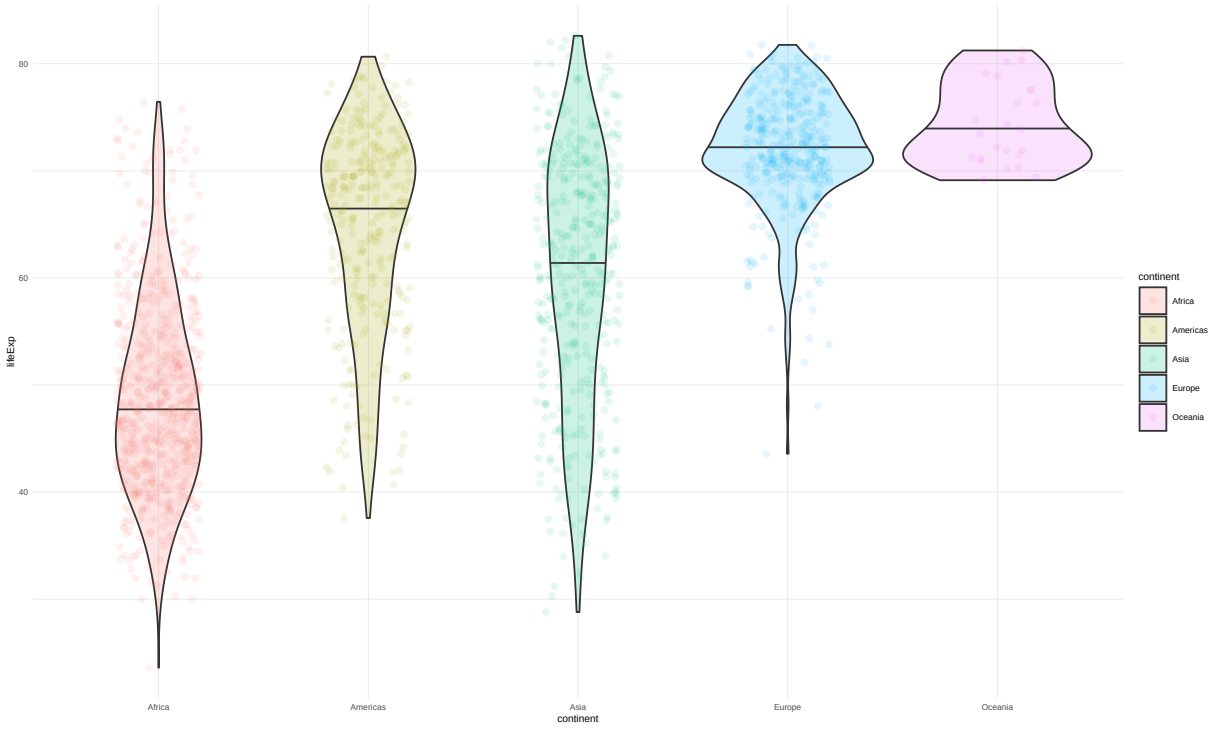
1. Que pasaría si lo pusieras en la linea `ggplot()`?

2. Afecta a los colores de `geom_jitter()`? ¿Por qué?

3. ¿Podrias reproducir el gráfico de abajo?

Solución

`aes(color = continent)` debe ir en `geom_jitter()`. Si se colorean también las lineas de los violines, es que has puesto `color = continent` dentro de `ggplot()`.



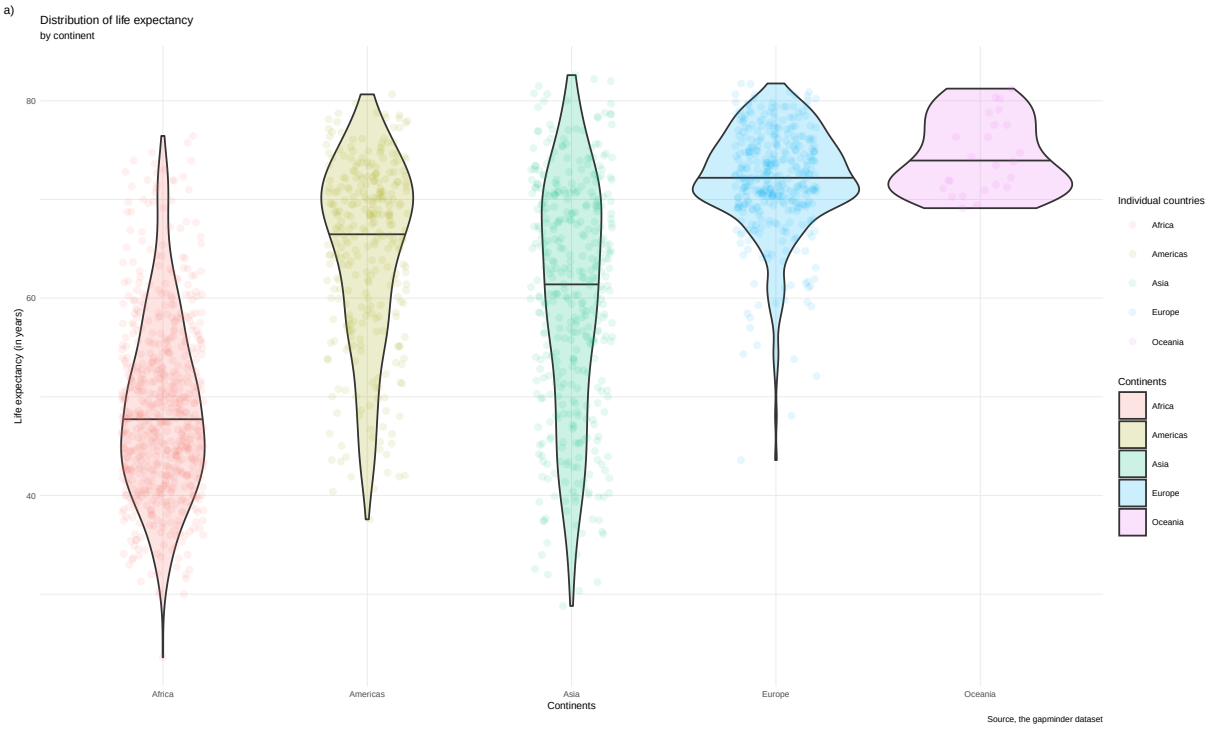
2.2.5 Personalización básica

Una gráfica necesita elementos como el título, ejes con nombres informativos, etc. Usaremos la función `labs()` para incluir o editar lo siguiente:

- `title`: título de la gráfica
- `subtitle`: subtítulo
- `caption`: pie de gráfica (abajo a la derecha)
- `tag`: etiqueta de la gráfica (arriba a la izquierda)
- `x`: eje horizontal
- `y`: eje vertical
- `fill`: título de leyenda si se usa el parámetro `fill`
- `color`: título de leyenda si se usa el parámetro `color`
- `alt`: alt-text, importante para que los lectores de pantalla usados por personas ciegas describan las gráficas

En el siguiente ejemplo, vamos a personalizar la gráfica del ejercicio anterior:

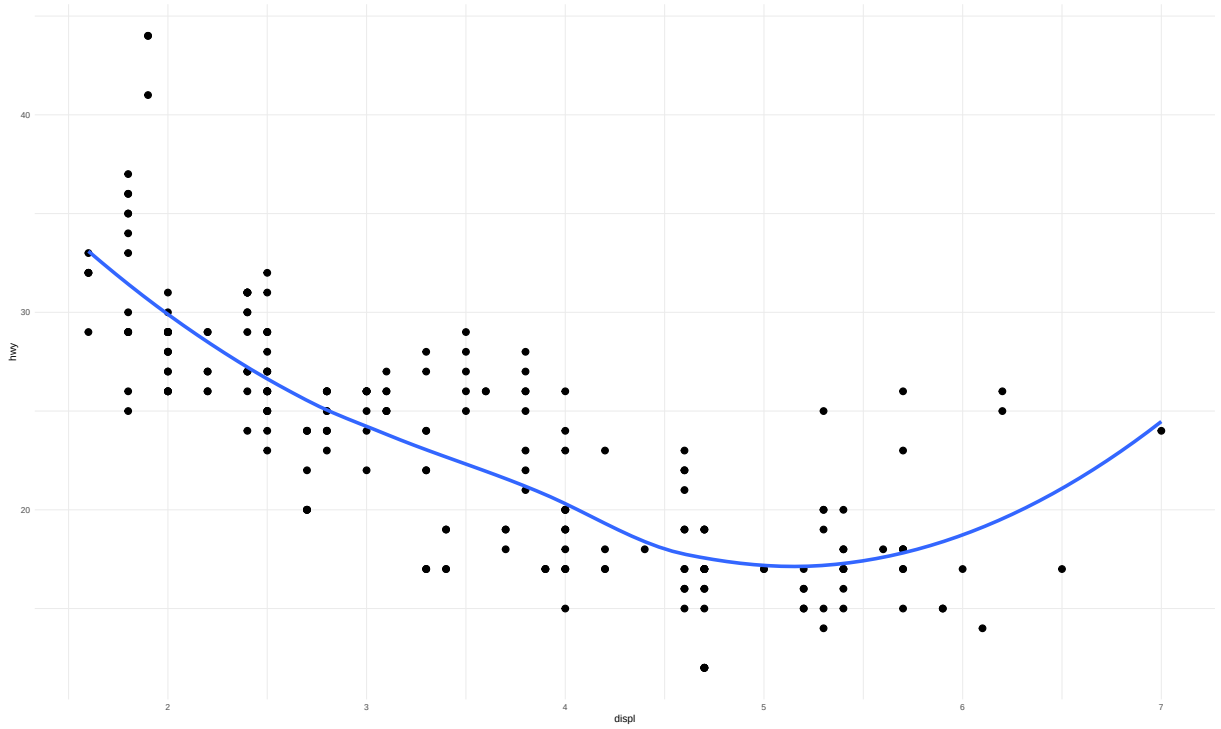
```
ggplot(gapminder, aes(continent, lifeExp)) +
  geom_violin(alpha = .2, aes(fill = continent), draw_quantiles = .5) +
  geom_jitter(alpha = .1, height = 0, width = .2, aes(color = continent)) +
  labs(title = "Distribution of life expectancy",
       subtitle = "by continent",
       caption = "Source, the gapminder dataset",
       tag = "a)",
       x = "Continents",
       y = "Life expectancy (in years)",
       fill = "Continents",
       color = "Individual countries",
       alt = "Alt text for the plot. Very useful for blind people"
       )
```



Ejercicio

Usando como base este plot:

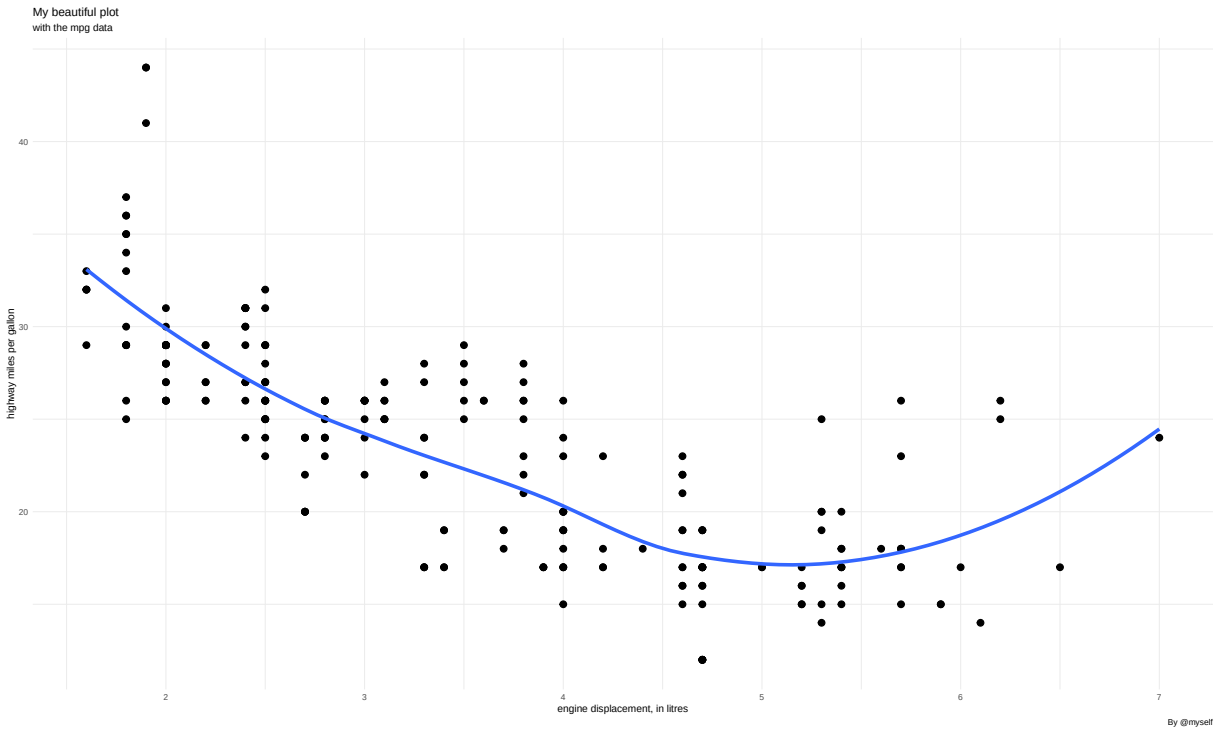
```
ggplot(mpg, aes(displ, hwy)) +
 geom_point() +
 geom_smooth(se = FALSE)
```



Personalízalo añadiendo y modificando:

- título
- subtítulo
- caption
- ejes x e y

Para conseguir esto:



Con lo que hemos visto en este capítulo, podrás crear una gran cantidad de gráficas. En el siguiente capítulo veremos algunas funcionalidades más avanzadas.

Bibliografía

- Matejka, J., & Fitzmaurice, G. (2017, May). Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (pp. 1290-1294). ACM.
- https://bbc.github.io/rcookbook/
- https://github.com/bbc/bbplot
- https://github.com/dreamRs/esquisse
- Garrick Aden-Buie. A Gentle Guide to the Grammar of Graphics with ggplot2: https://github.com/gadenbuie/gentle-ggplot2
- Michael Toth. You Need to Start Branding Your Graphs. Here's How, with ggplot!: https://michaeltoth.me/you-need-to-start-branding-your-graphs-heres-how-with-ggplot.html
- Claus Wilke: https://wilkelab.org/practicalgg/
- Thomas Lin Pedersen:

  - Part 1: https://www.youtube.com/watch?v=h29g21z0a68
  - Part 2: https://www.youtube.com/watch?v=0m4yywqNPVY

- Big Book or R : https://www.bigbookofr.com/index.html

3 Visualización avanzada

Veamos algunas funcionalidades más avanzadas con ggplot y otros paquetes.

! Paquetes para este capítulo

Igual que en capítulo anterior, para poder ejecutar en tu ordenador el código de los ejemplos y ejercicios vas a necesitar los paquetes del recuadro siguiente.
Recuerda que cuando empecemos **cada capítulo**:

1) Abre un script de R y guárdalo con el nombre del capítulo: `capitulo3.R`
2) Copia las líneas de abajo (click en el icono del cuadro gris de abajo) y pégalas en el script
3) Ejecútalas: CNTRL + ENTER para ejecutar linea a linea, o CNTRL + ALT + R para ejecutar todo

```
if (!require('cowplot')) install.packages('cowplot'); library('cowplot')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('esquisse')) install.packages('esquisse'); library('esquisse')
if (!require('gapminder')) install.packages('gapminder'); library('gapminder')
if (!require('geomtextpath')) install.packages('geomtextpath'); library('geomtextpath')
if (!require('gghighlight')) install.packages('gghighlight'); library('gghighlight')
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
if (!require('ggrain')) install.packages('ggrain'); library('ggrain')
if (!require('ggthemes')) install.packages('ggthemes'); library('ggthemes')
if (!require('ggridges')) install.packages('ggridges'); library('ggridges')
if (!require('ggtext')) install.packages('ggtext'); library('ggtext')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('plotly')) install.packages('plotly'); library('plotly')
if (!require('purrr')) install.packages('purrr'); library('purrr')
if (!require('readr')) install.packages('readr'); library('readr')
if (!require('sjPlot')) install.packages('sjPlot'); library('sjPlot')
if (!require('tidyr')) install.packages('tidyr'); library('tidyr')
```

3.1 Facets

Cuando queremos separar en gráficos independientes distintas categorías dentro de nuestros datos, podemos usar *facetas*. Hay dos funciones para esto, **facet_grid()** y **facet_wrap()**.

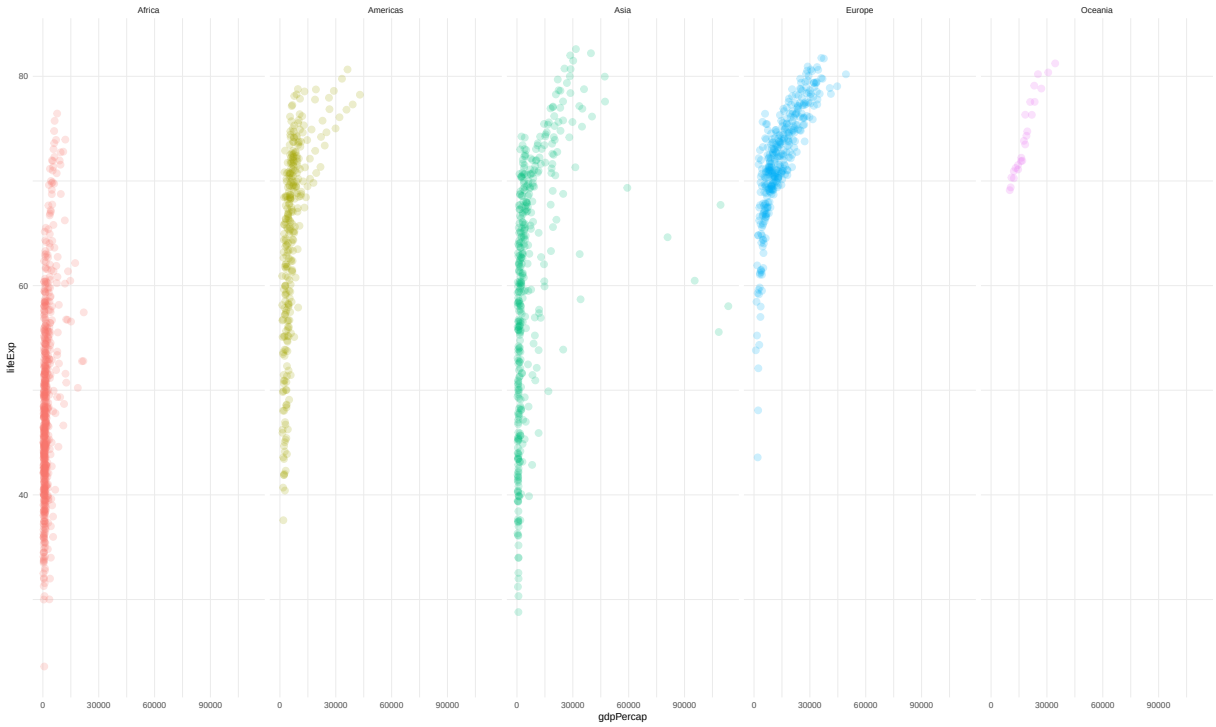
3.1.1 facet_grid

`facet_grid(~ variable)` nos devuelve una matriz simétrica de gráficas.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(alpha = .2)
```

Un gráfico para cada continente. Verás que usamos `guides(color = "none")` para que no se vea la leyenda asociada a color. Prueba ejecutar este código con y sin la última línea para ver la diferencia.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(alpha = .2) +
  facet_grid(~ continent) +
  guides(color = "none")
```



Cambiamos los ejes.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(alpha = .2) +
  facet_grid(continent ~ .) +
  guides(color = "none")
```

Podemos añadir una segunda variable (categórica) para tener un gráfico para cada combinación de categorias.

Un truco muy útil es dicotomizar variables usando condiciones lógicas (e.g. `pop > 5000000`).

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = country)) +
  geom_point(alpha = .2) +
  facet_grid(continent ~ pop > 5000000) +
  guides(color = "none")
```

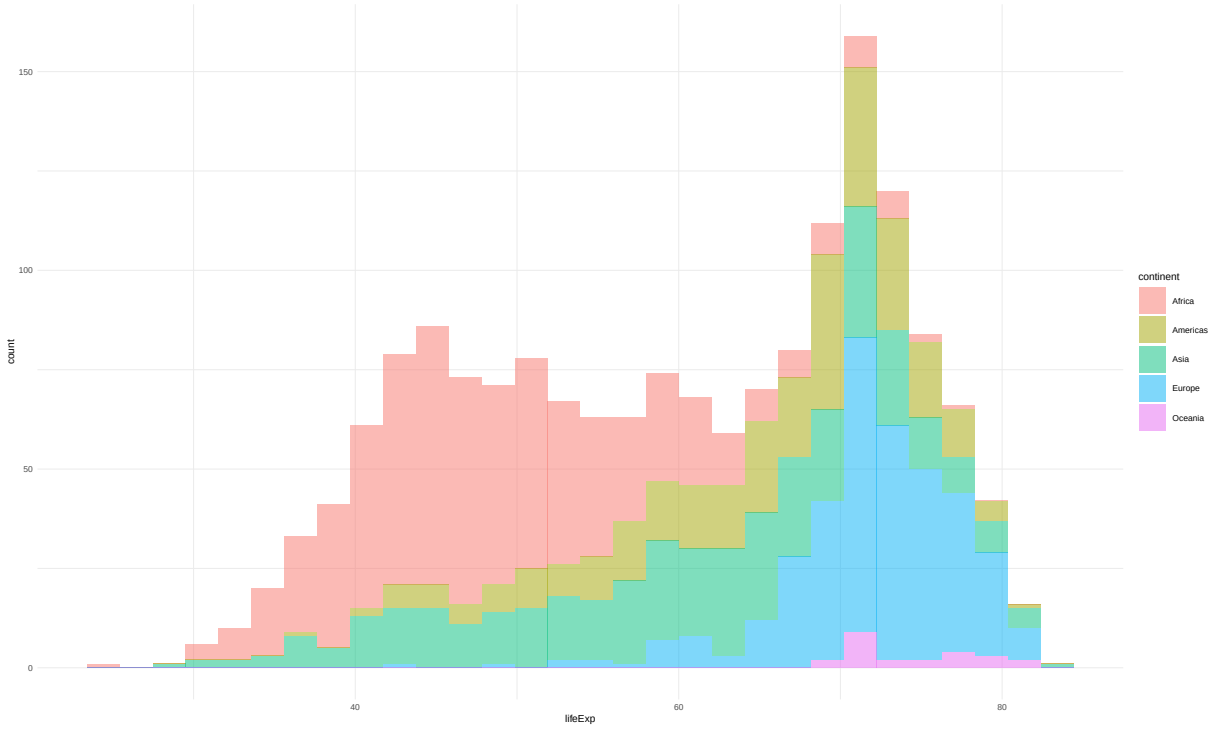
Si queremos usar nombres con significado, podemos usar la función `ifelse()`.

```
ggplot(gapminder, aes(gdpPercap, lifeExp, color = country)) +
  geom_point(alpha = .2) +
  facet_grid(continent ~ ifelse(pop > 5000000, "Big countries", "Small countries")) +
  guides(color = "none")
```

3.1.2 facet_wrap

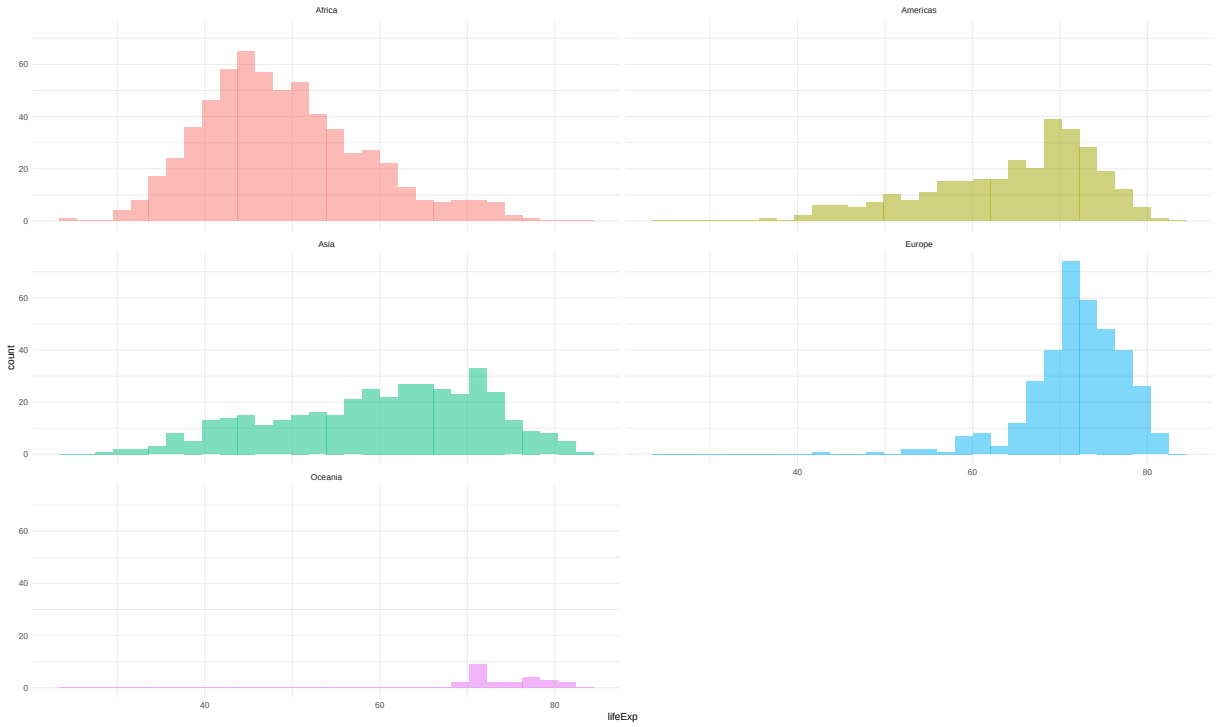
`facet_wrap(~ variable)` nos devuelve tantas facetas como niveles de la variable, pudiendo definir el número de filas y columnas que queremos.

```
ggplot(gapminder, aes(lifeExp, fill = continent)) +
  geom_histogram(alpha = .5)
```



Facetas por continente (en 2 columnas con `ncol = 2`).

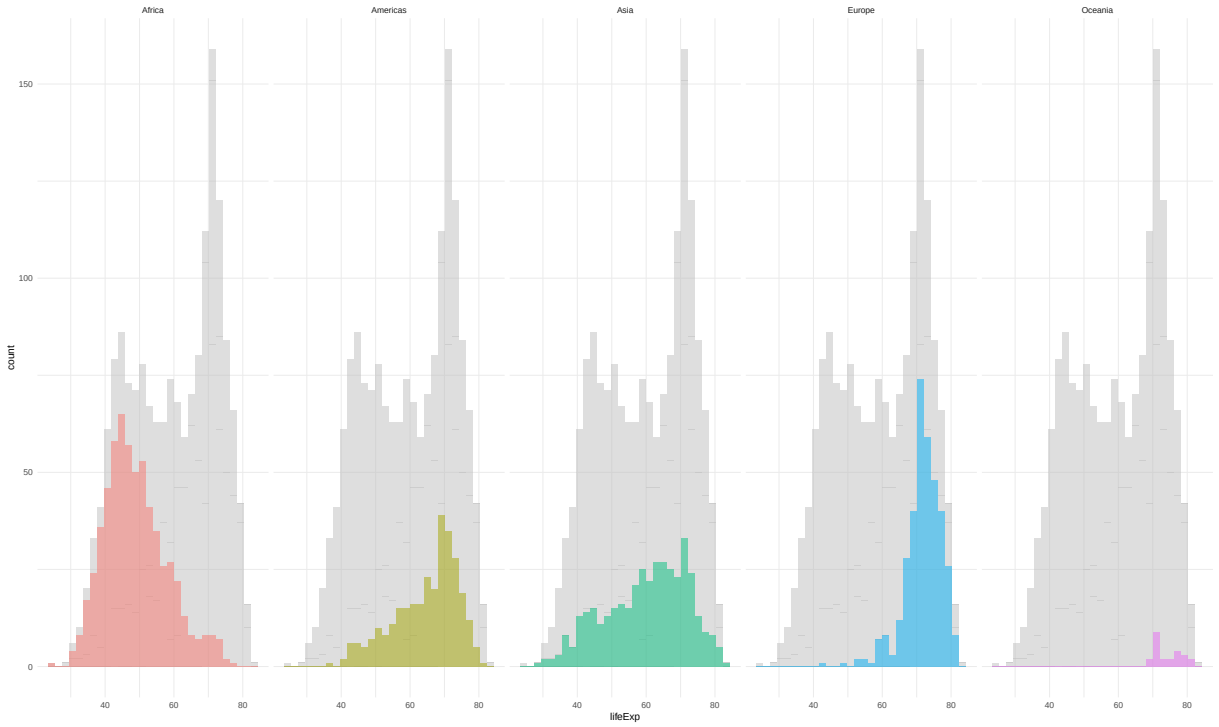
```
ggplot(gapminder, aes(lifeExp, fill = continent)) +
  geom_histogram(alpha = .5) +
  facet_wrap( ~ continent, ncol = 2) +
  guides(fill = "none")
```



3.1.3 gghighlight con facetas

Con la función `gghighlight()` del paquete gghighlight podemos añadir una capa para facilitar la comparación de cada faceta con los datos completos.

```
ggplot(gapminder, aes(lifeExp, fill = continent)) +
  geom_histogram(alpha = .5) +
  facet_wrap( ~ continent, nrow = 1) +
  guides(color = "none") +
  gghighlight::gghighlight()
```



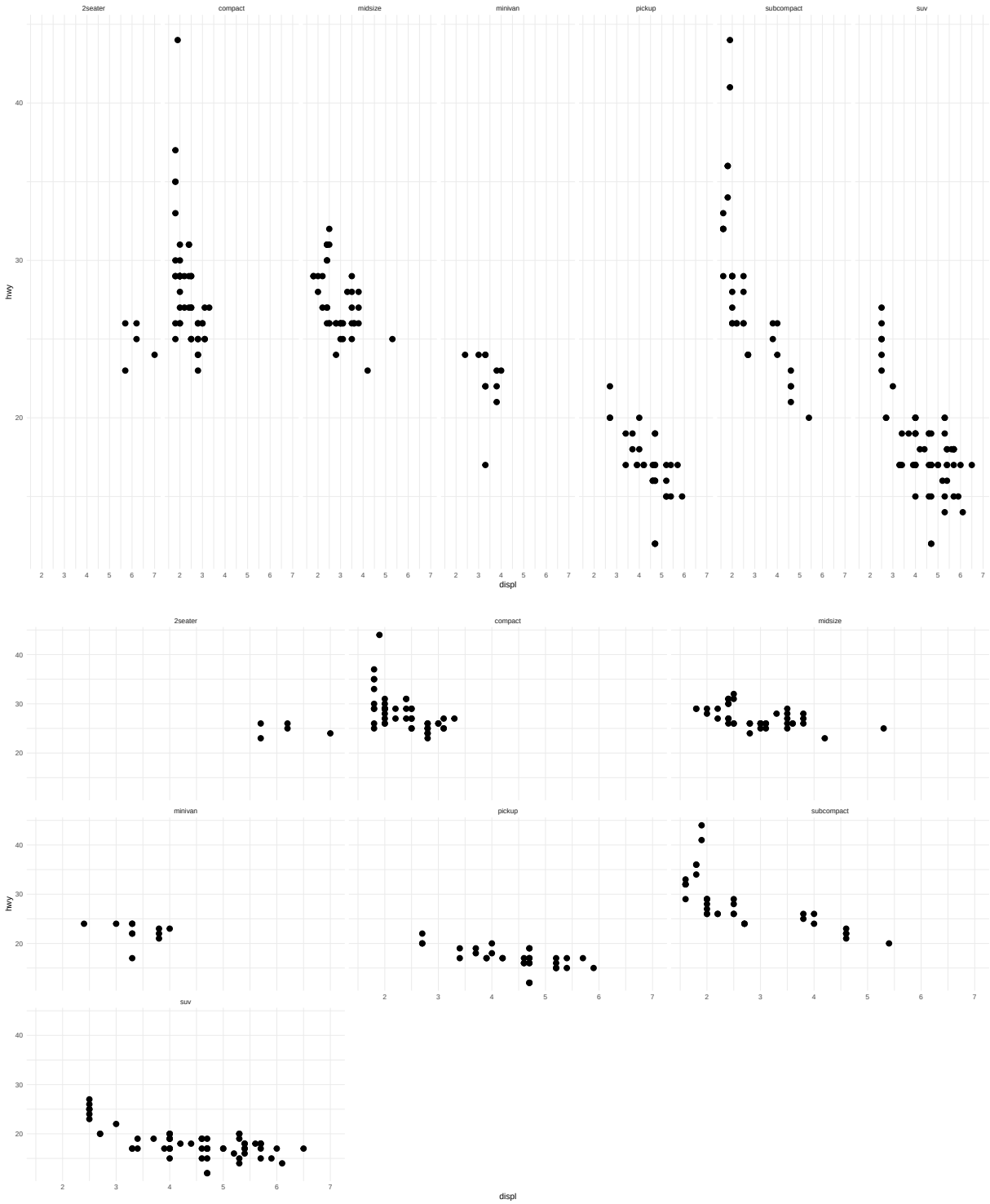
Ejercicios

Usando como base el plot siguiente:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point()
```

- Crea un panel para cada tipo de coche (`class`) en una rejilla simétrica
- Crea un panel para cada tipo de coche (`class`), mostrando paneles en 3 filas

> Solución
>
> `facet_grid()` permite crear rejillas simétricas de paneles, y el parámetro `nrow` de `facet_wrap()` nos ayuda con paneles con números de filas definidos.


2seater
compact
midsize
minivan
pickup
subcompact
suv
hwy
displ

3.2 Transformaciones estadísticas

ggplot2 nos permite hacer transformaciones estadísticas al crear los gráficos. Vamos a ver algunos ejemplos aquí, pero se pueden hacer muchas más cosas. Para más detalles, ver r4ds.

3.2.1 Computaciones con ggplot: stat_summary()

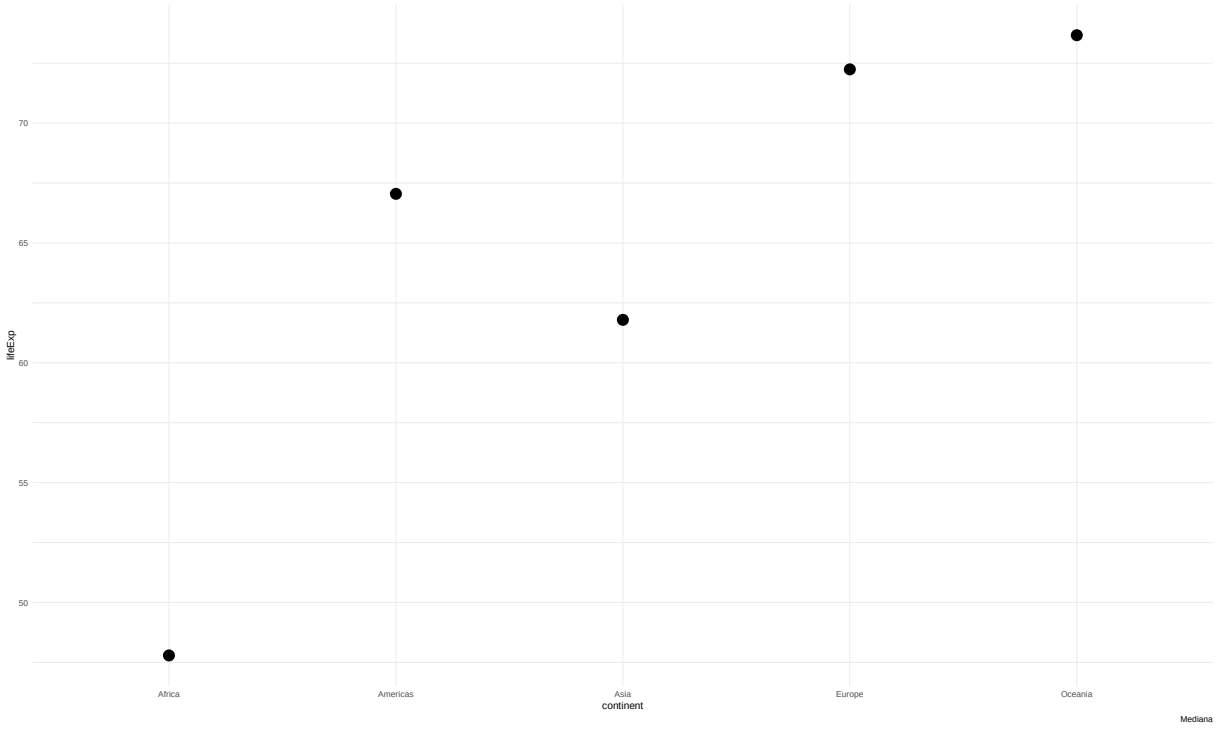
En ocasiones queremos visualizar estadísticas descriptivas asociadas a los datos (e.g. promedio, mínimo y máximo por condición), pero como generalmente trabajaremos con data frames en formato long (una observación por fila), no podremos usar los geoms que hemos visto hasta ahora.

Tenemos dos opciones, la primera es preparar nuevos data frames antes de pasar a la visualización. La segunda, realizar la computación directamente con ggplot, usando `stat_summary()` junto con alguna de las funciones tradicionales para extraer estadísticas descriptivas.

> **i** Ejemplos de funciones que podemos usar en los gráficos
>
> - **min()**: mínimo
> - **max()**: máximo
> - **mean()**: media
> - **median()**: mediana
> - **sd()**: desviación estándar

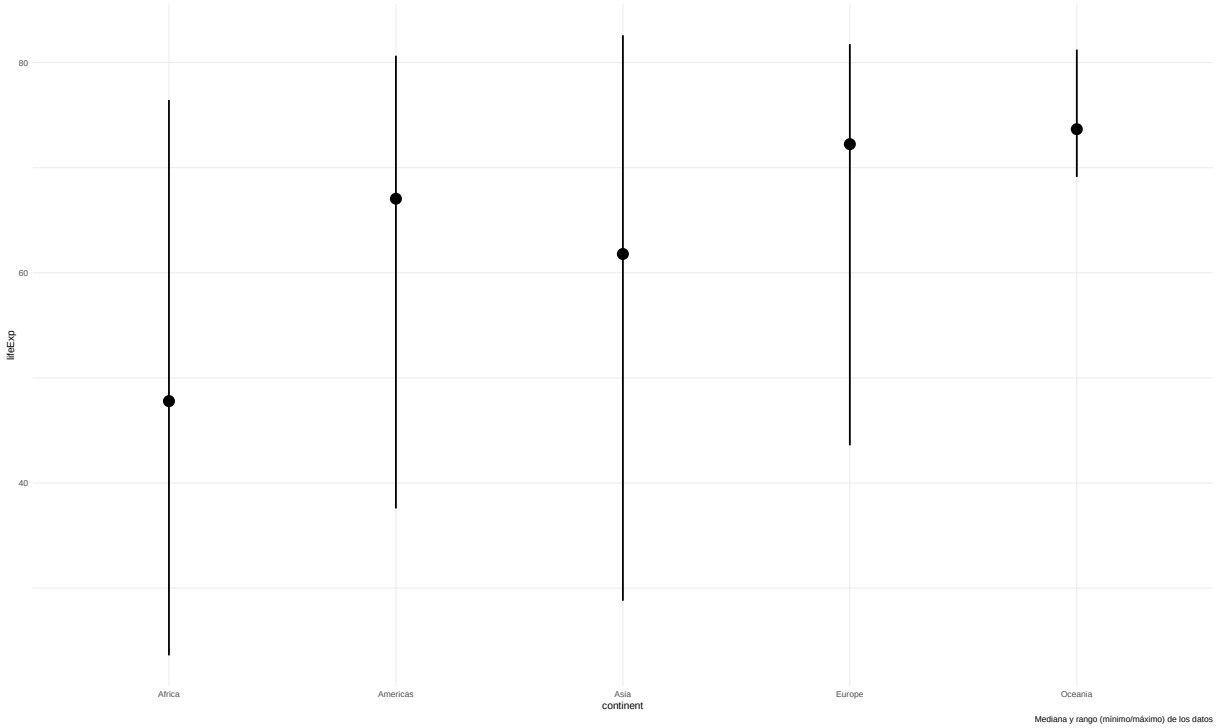
Con `stat_summary()` podemos usar funciones simples de manera directa. Por ejemplo, si queremos visualizar la mediana de `lifeExp` para cada continente, podemos hacer lo siguiente:

```
ggplot(gapminder, aes(continent, lifeExp)) +
  stat_summary(fun = median) +
  labs(caption = "Mediana")
```



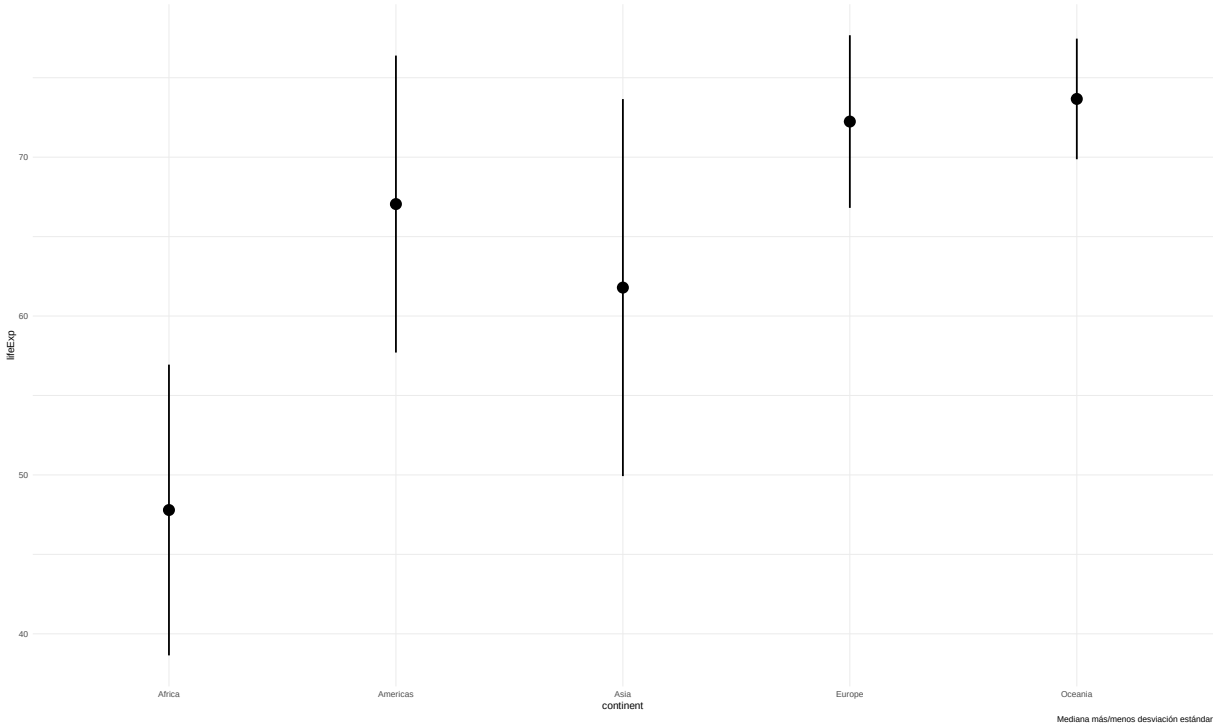
`stat_summary()` tiene un buen número de parámetros (F1 sobre la función para ver la ayuda). Por ejemplo, `fun.min` y `fun.max` nos permitirán añadir a la gráfica anterior el rango completo de los datos:

```
ggplot(gapminder, aes(continent, lifeExp)) +
  stat_summary(
    fun = median,
    fun.min = min,
    fun.max = max
    ) +
  labs(caption = "Mediana y rango (mínimo/máximo) de los datos")
```



Si queremos usar funciones algo más complejas, la sintaxis es diferente. En este caso mostramos media ± desviación estándar:

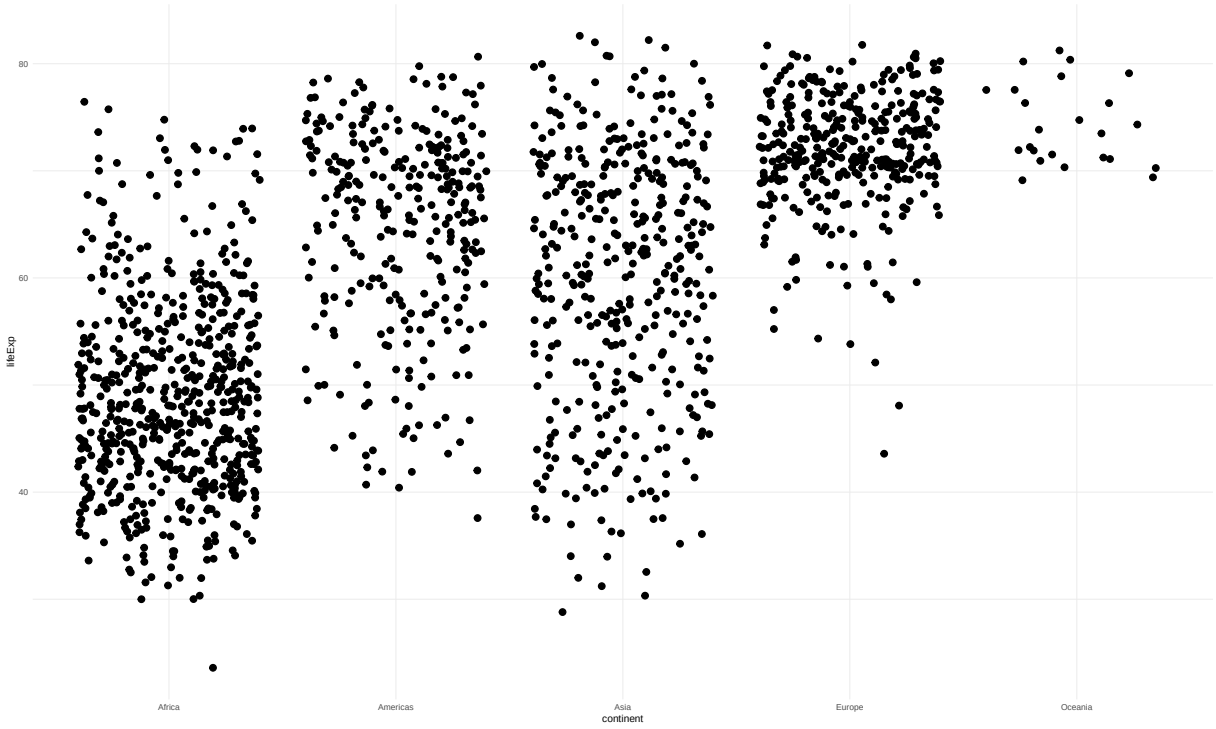
```
ggplot(gapminder, aes(continent, lifeExp)) +
  stat_summary(
    fun = median,
    fun.min = function(x) median(x) - sd(x),
    fun.max = function(x) median(x) + sd(x)
    ) +
labs(caption = "Mediana más/menos desviación estándar")
```



3.2.2 Promedios por grupo

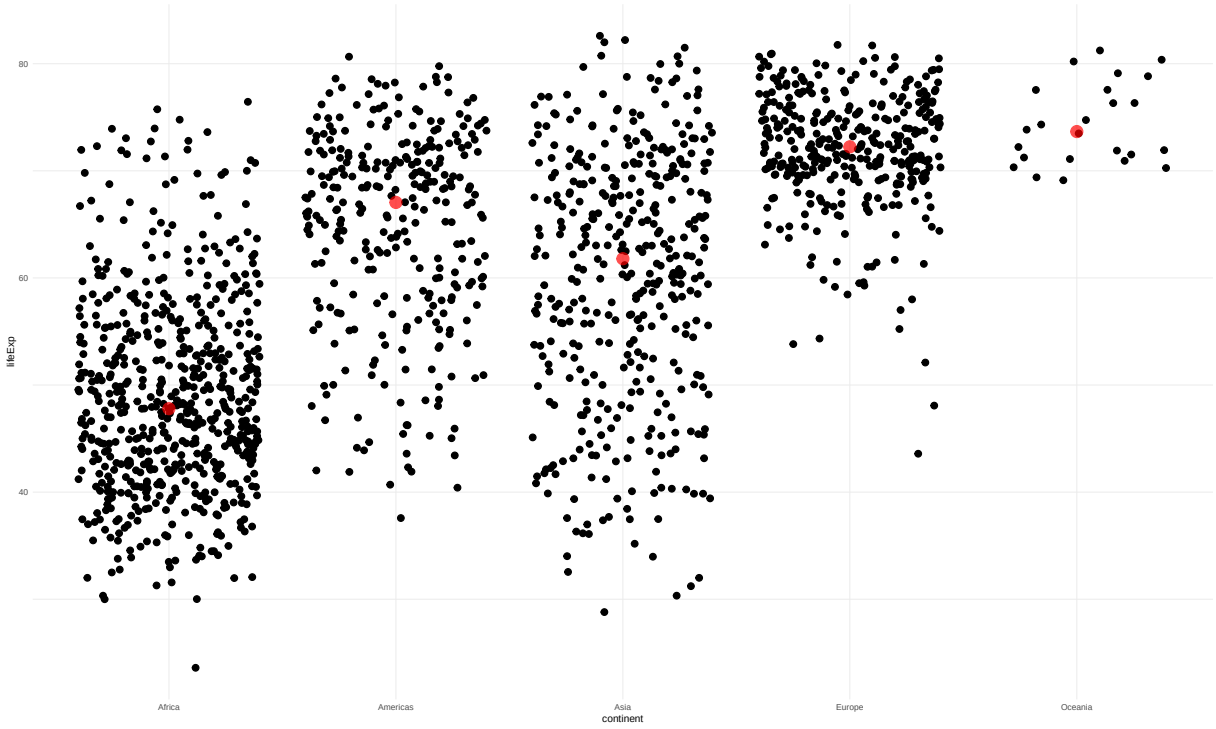
Lo interesante es que podemos añadir estas transformaciones estadísticas como una capa más en los gráficos. Esto es ideal para mostrar los puntos individuales de nuestros datos, algo crítico como vimos en el tema anterior. Así que, a este gráfico inicial…

```
ggplot(gapminder, aes(continent, lifeExp)) +
geom_jitter()
```



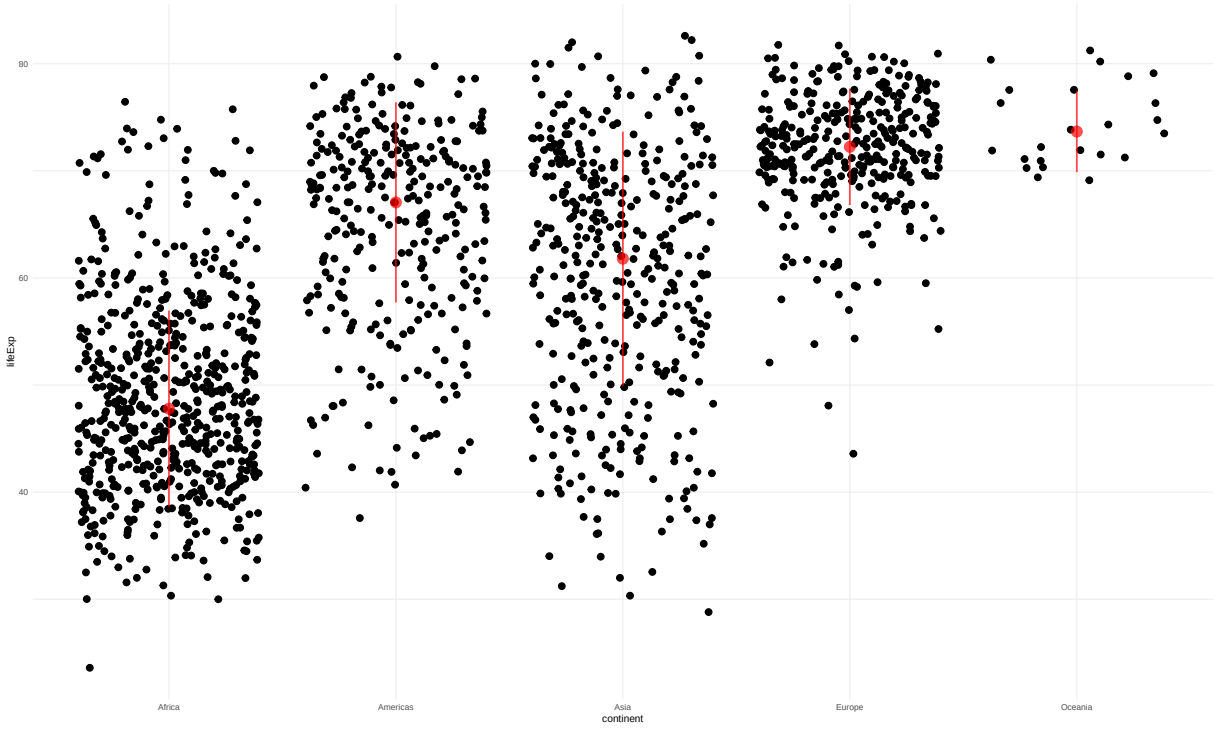
Le podemos añadir un punto mostrando la mediana por grupo:

```
ggplot(gapminder, aes(continent, lifeExp)) +
geom_jitter() +
stat_summary(fun = median,
             geom = "point",
             color = "red", size = 3, alpha = .7)
```



O la mediana más la desviación estándar:

```
ggplot(gapminder, aes(continent, lifeExp)) +
geom_jitter() +
stat_summary(fun = median,
             color = "red", size = .5, alpha = .7,
             fun.min = function(x) median(x) - sd(x),
             fun.max = function(x) median(x) + sd(x))
```



Ejercicios

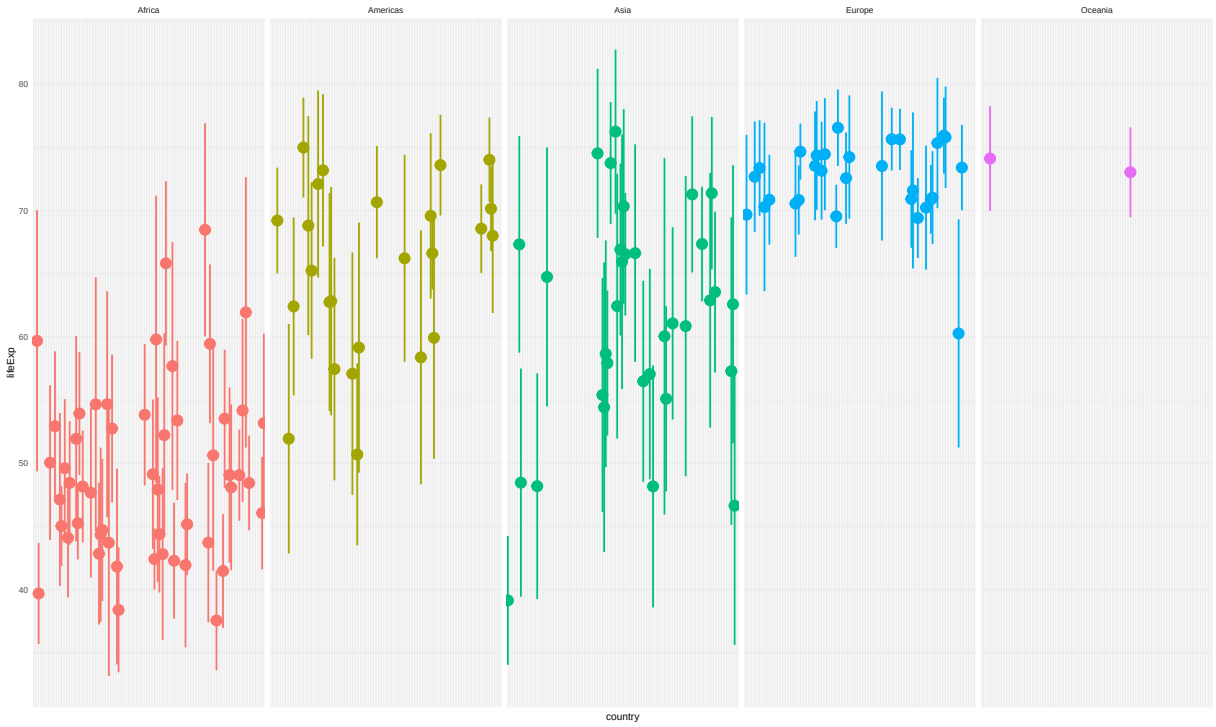
Ejercicio básico

Usando como base:

```
ggplot(gapminder, aes(country, lifeExp, color = continent)) +
  stat_summary(...) +
  facet_grid(...) +
  theme(axis.text.x = element_blank(), # Eliminamos etiquetas de nombres de paises del eje x
        legend.position = "none") # Elimina la leyenda
```

¿Podrías crear este gráfico? Mostramos mediana ± sd para cada país, organizado por continente.

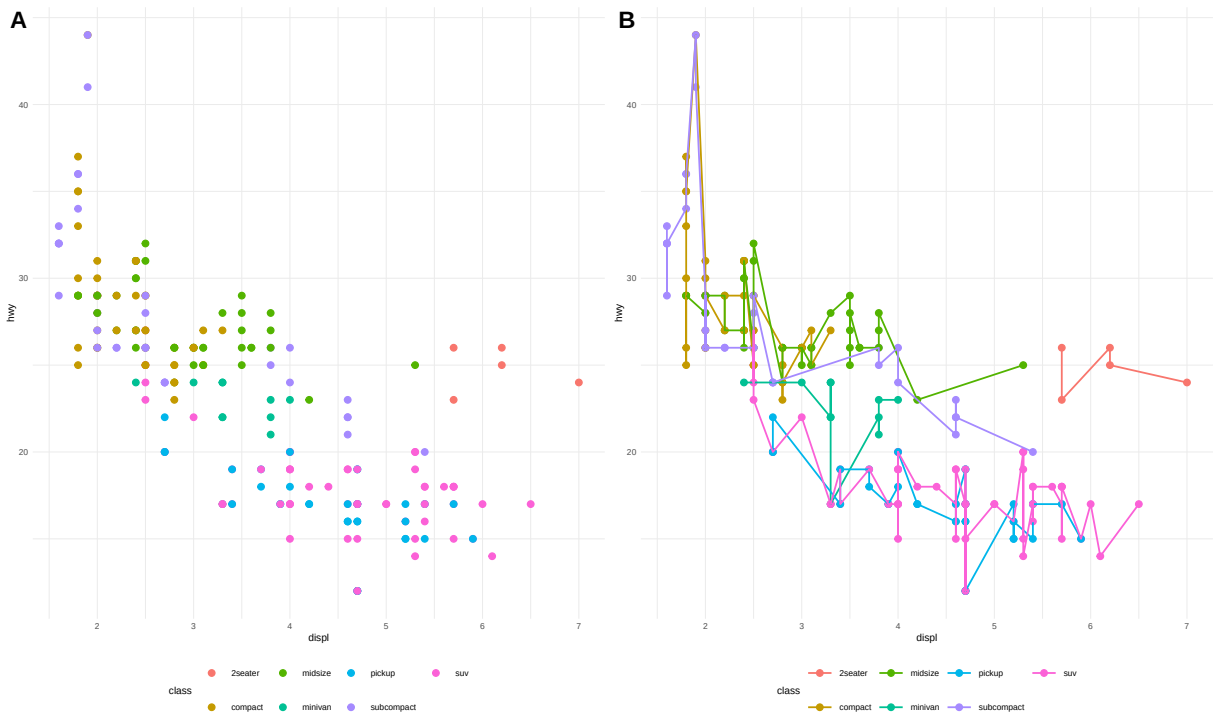
> **Pista**
>
> Tienes que encontrar los parámetros adecuados para `stat_summary()` y `facet_grid()`. Puedes ver ejemplos en: - computaciones con ggplot: stat_summary() - Facet_grid.



Ejercicio avanzado

Cuando al plot **A** trato de añadirle líneas para cada `class`, me aparece algo como lo de **B**, porque tenemos varios puntos en cada nivel de `displ`.

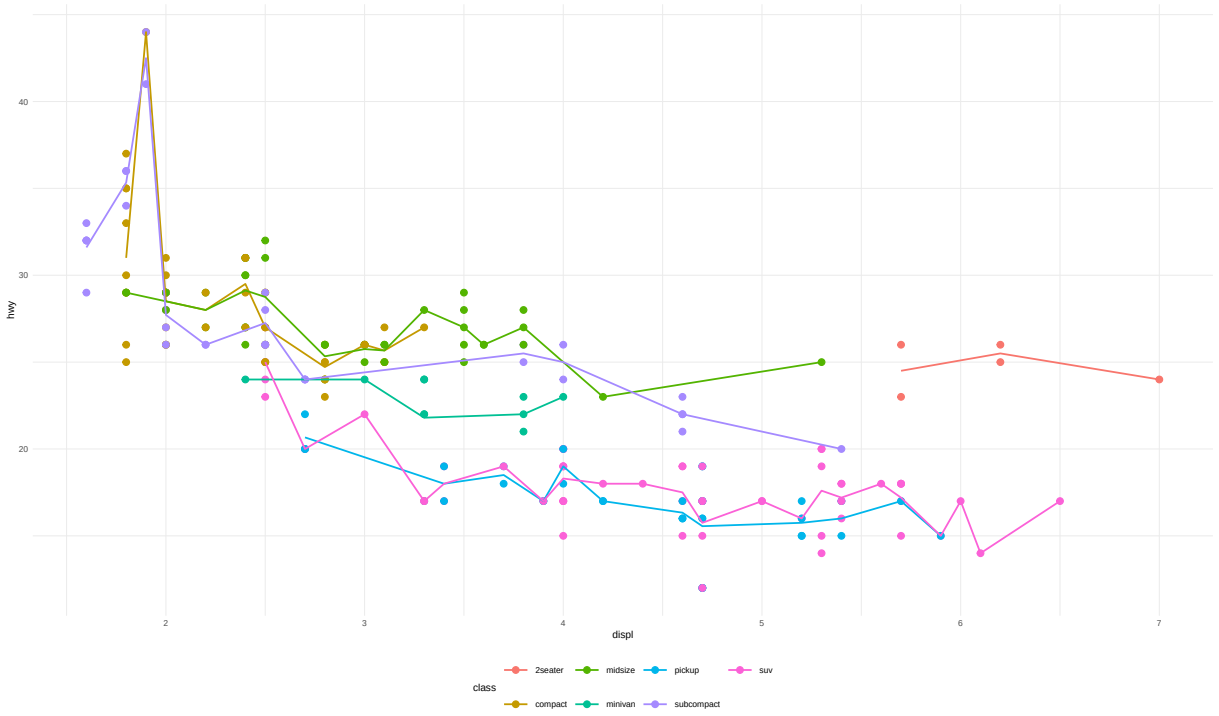
```
plotA = ggplot(mpg, aes(displ, hwy, color = class)) +
  geom_point() +
  theme(legend.position = "bottom")

plotB = ggplot(mpg, aes(displ, hwy, color = class)) +
  geom_point() +
  geom_line() +
  theme(legend.position = "bottom")

cowplot::plot_grid(plotA, plotB, labels = c("A", "B"))
```



Pero en realidad no quiero que las líneas pasen por todos los puntos, sino que muestren el promedio en cada nivel de `displ` para cada `class` de vehículo.

1) ¿Podrías reproducir el gráfico de abajo?

> **Pista**
>
> Tendrás que reemplazar `geom_line()` por `stat_summary()`, usando el parámetro `geom = "line"` para indicarle que quieres usar lineas en lugar de puntos.



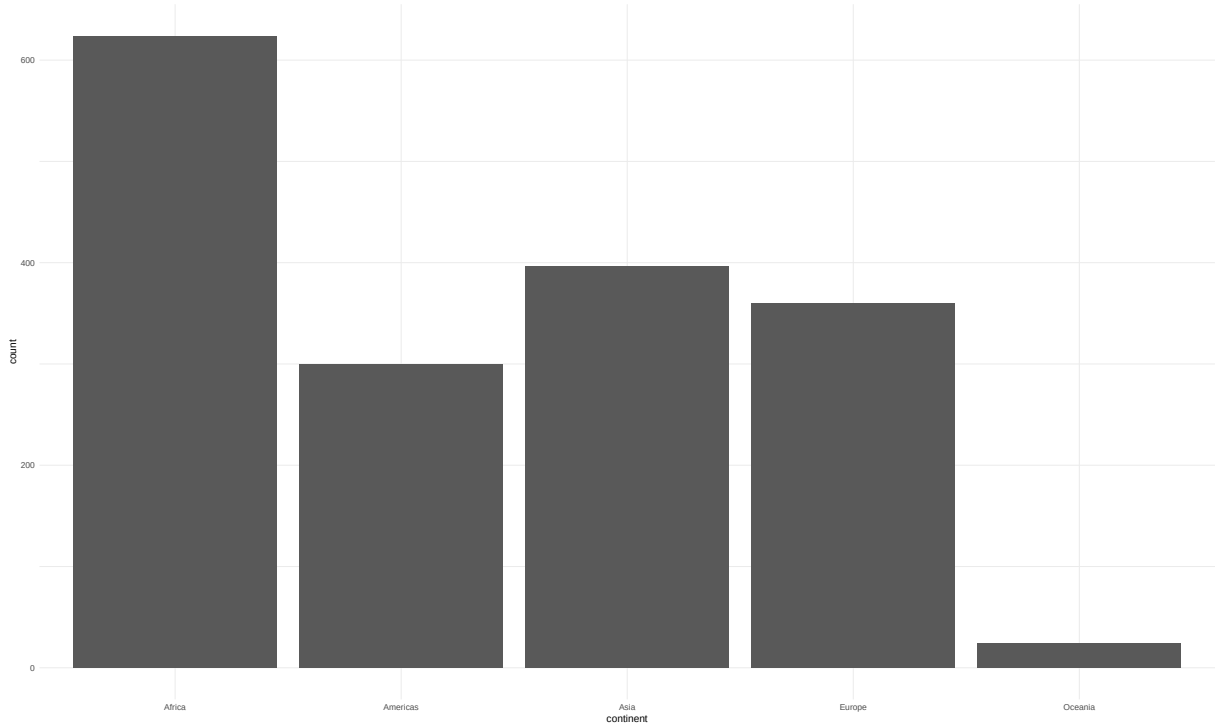
3.3 Personalización avanzada de gráficas

Habitualmente, un vez hemos creado la gráfica, querremos personalizar varias cosas, como las escalas, colores, estilos, título, etc.

3.3.1 Coordenadas

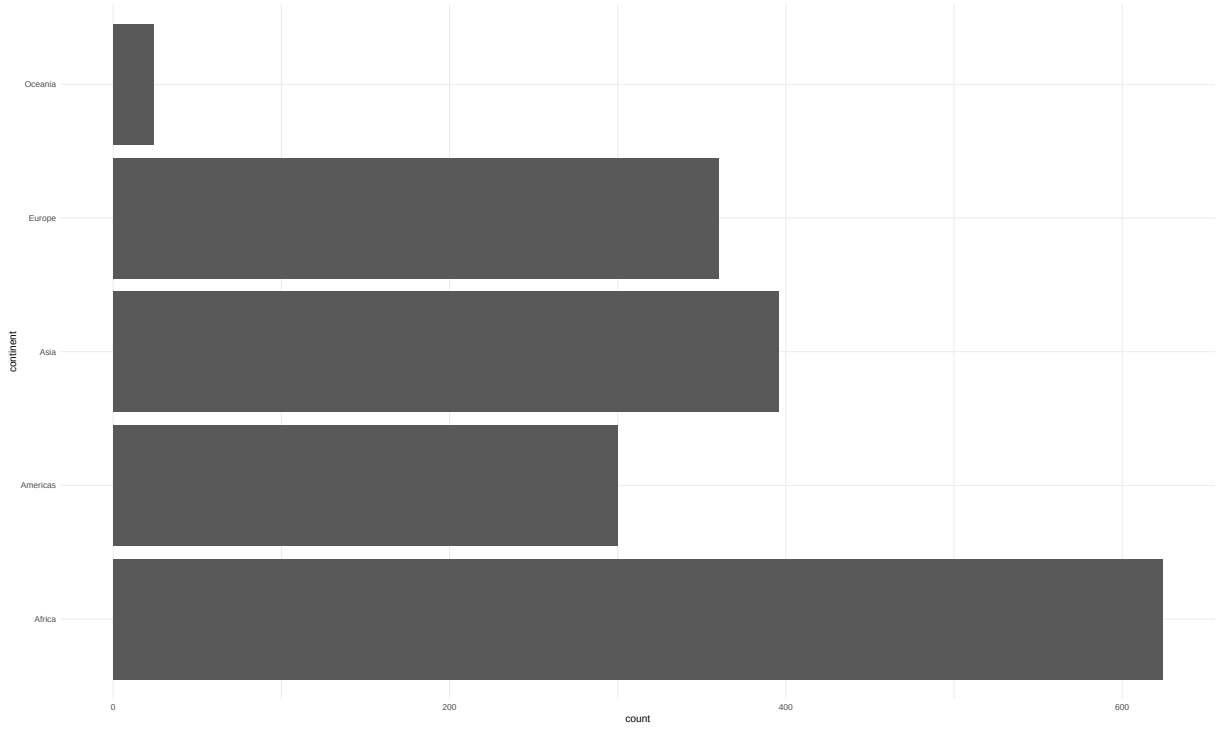
Gráfico inicial:

```
ggplot(gapminder, aes(continent)) +
    geom_bar()
```



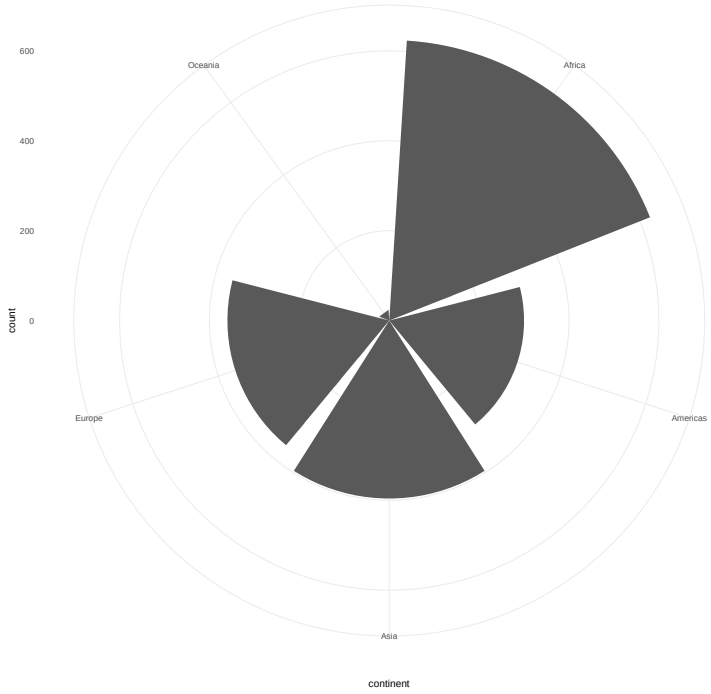
Usamos `coord_flip()` para rotar las coordenadas:

```
ggplot(gapminder, aes(continent)) +
  geom_bar() +
  coord_flip()
```



O `coord_polar()` para usar el sistema de coordenadas polar (360º):

```
ggplot(gapminder, aes(continent)) +
  geom_bar() +
  coord_polar()
```



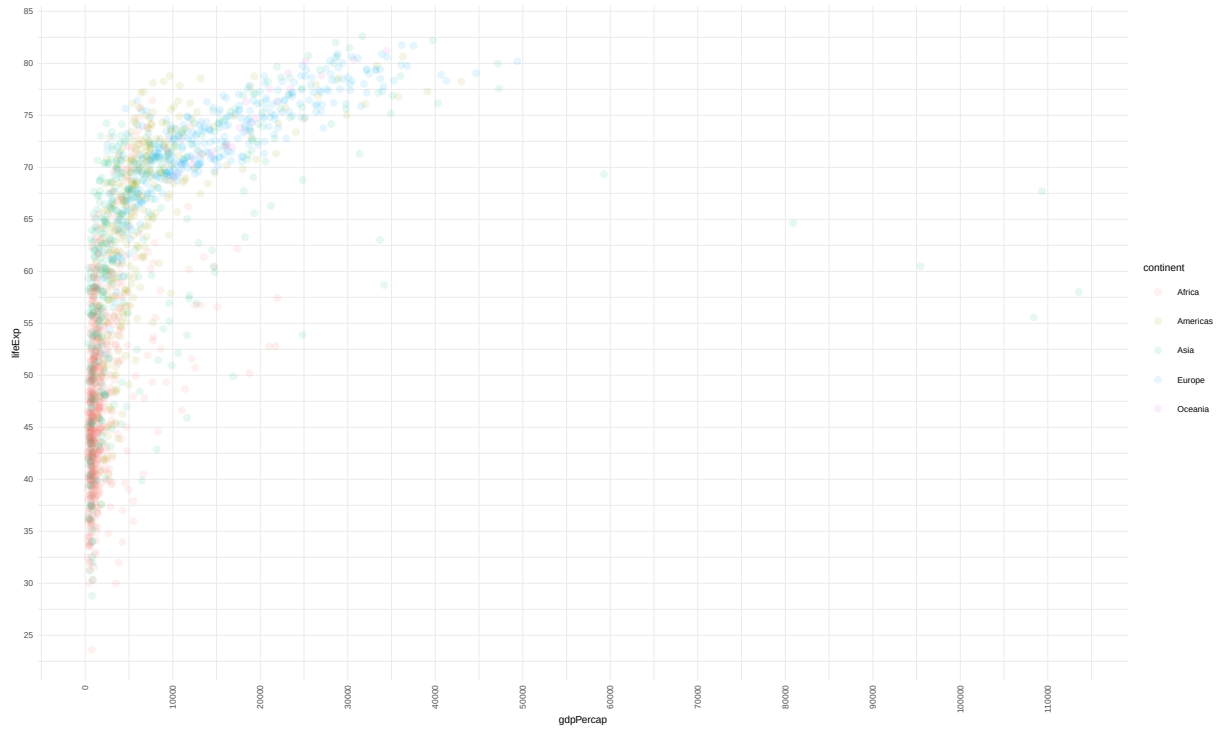
3.3.2 Scales

Usaremos las funciones que empiezan por `scale_` para multitud de cosas, por ejemplo, cambiar las etiquetas de los ejes x o y:

```
# Gráfico inicial
plot_base = ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(alpha = .1)
plot_base
```

Definimos cuantos breaks queremos en cada eje (`n.breaks`), y rotamos las etiquetas del eje x (`guide = guide_axis(angle = 90)`):

```
plot_base +
  scale_x_continuous(n.breaks = 15, guide = guide_axis(angle = 90)) +
  scale_y_continuous(n.breaks = 15)
```



Separador de miles (`labels = scales::comma`) y breaks en x (`n.breaks`):

```
plot_base +
  scale_y_continuous(n.breaks = 15) +
  scale_x_continuous(n.breaks = 6, labels = scales::comma)
```

Con `scales::dollar_format()` le damos formato de $ ($M)

```
plot_base +
  scale_y_continuous(n.breaks = 15) +
  scale_x_continuous(n.breaks = 6,
                     labels = scales::dollar_format(
                       prefix = "$",
                       suffix = "M")
                     )
```

Escala logarítmica. Muy útil para mostrar crecimiento exponencial:

```
plot_base +
  scale_y_continuous(n.breaks = 15) +
  scale_x_log10(n.breaks = 4,
                labels = scales::dollar_format(
                  prefix = "$",
                  suffix = "M")
                )
```

Invertimos escala:

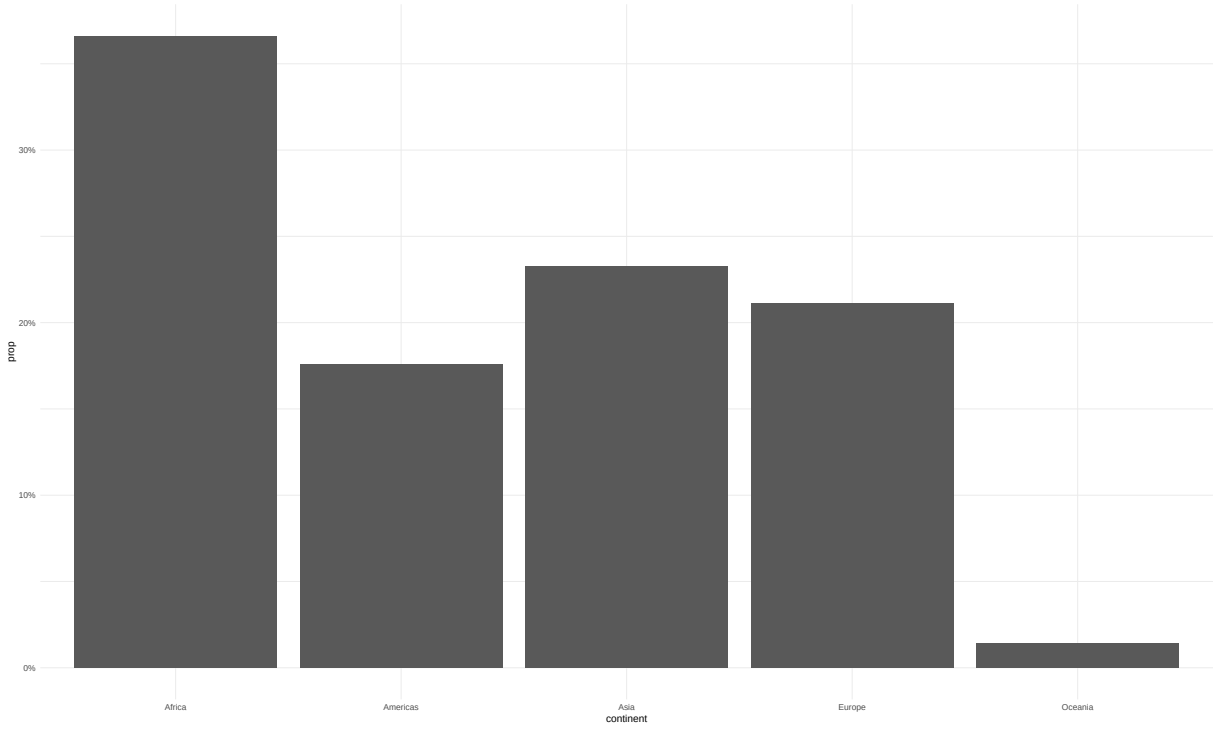
```
plot_base +
  scale_y_reverse()
```

No mostramos el texto ni los ticks de los breaks de x:

```
plot_base +
  scale_y_reverse() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```

Porcentaje:

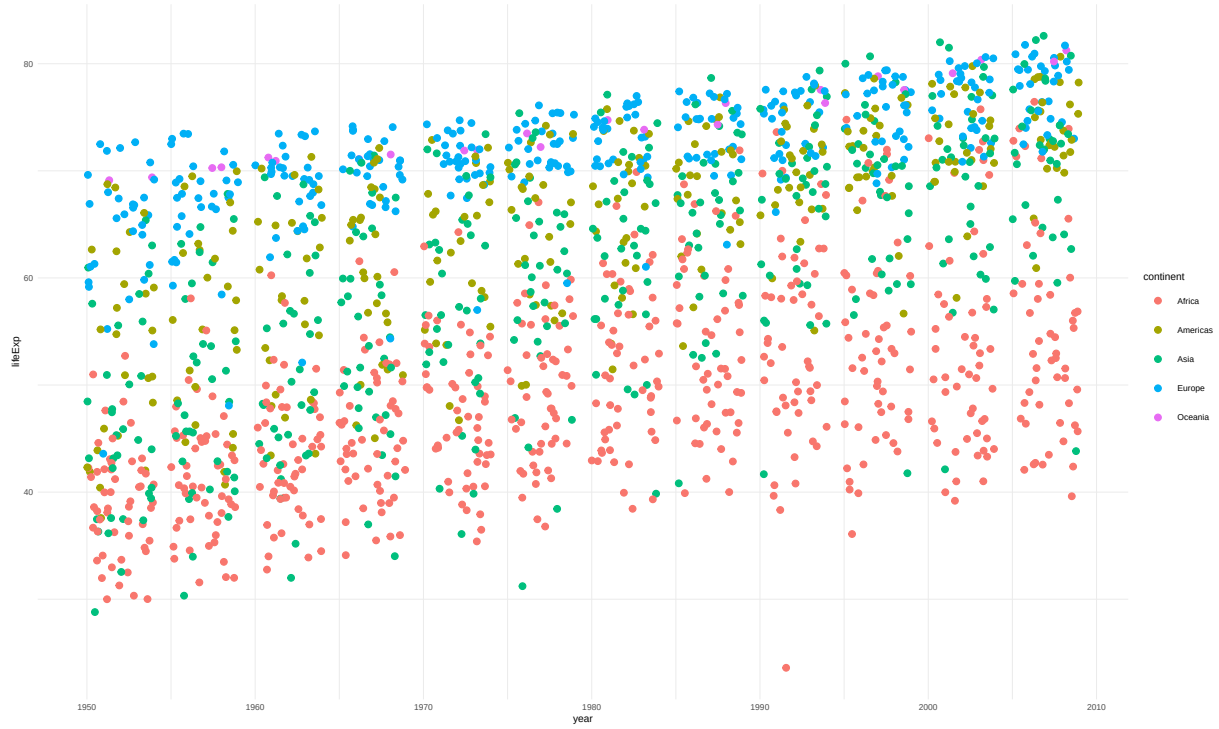
```
ggplot(gapminder, aes(continent, after_stat(prop), group = 1)) +
  geom_bar() +
  scale_y_continuous(labels = scales::percent)
```



3.3.3 Legends

La leyenda de las gráficas nos muestra, por defecto, los colores, rellenos, tipos de linea, etc. que hayamos usado. Por ejemplo, abajo nos muestra la leyenda asociada al color.

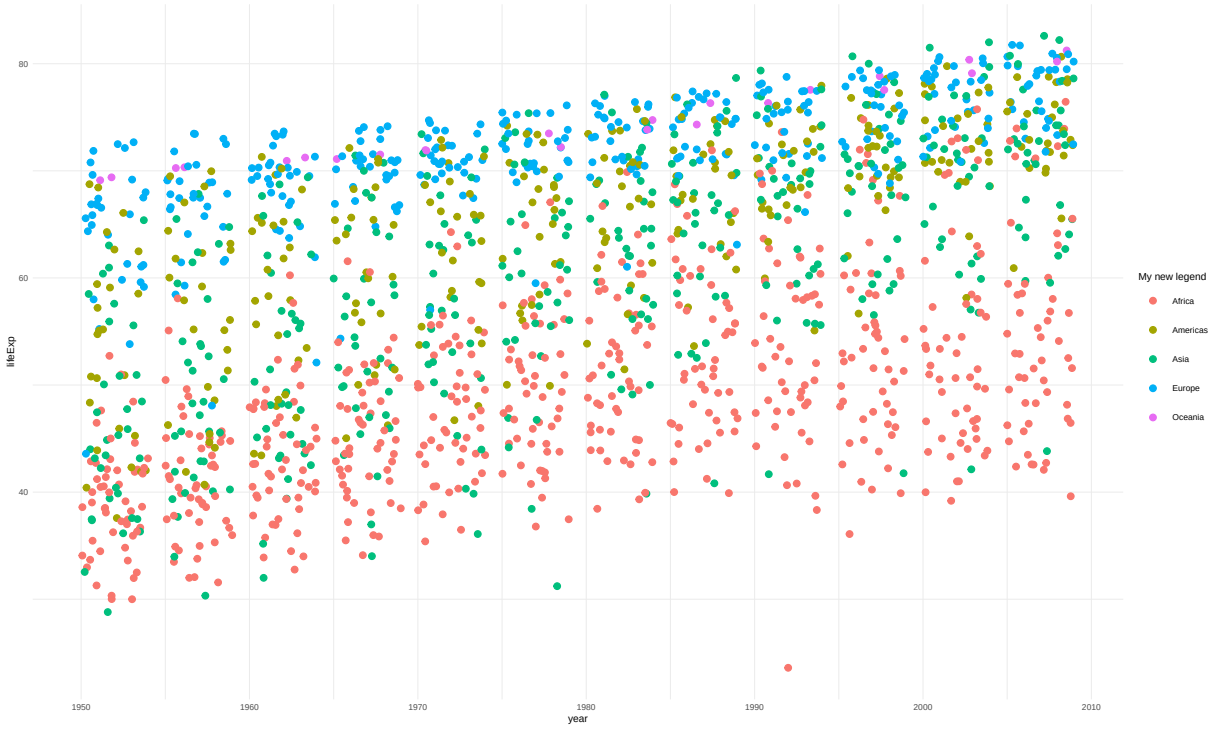
```
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  geom_jitter()
```



Podemos hacer algunas cosas básicas como cambiar el nombre de la leyenda, o no mostrarla.

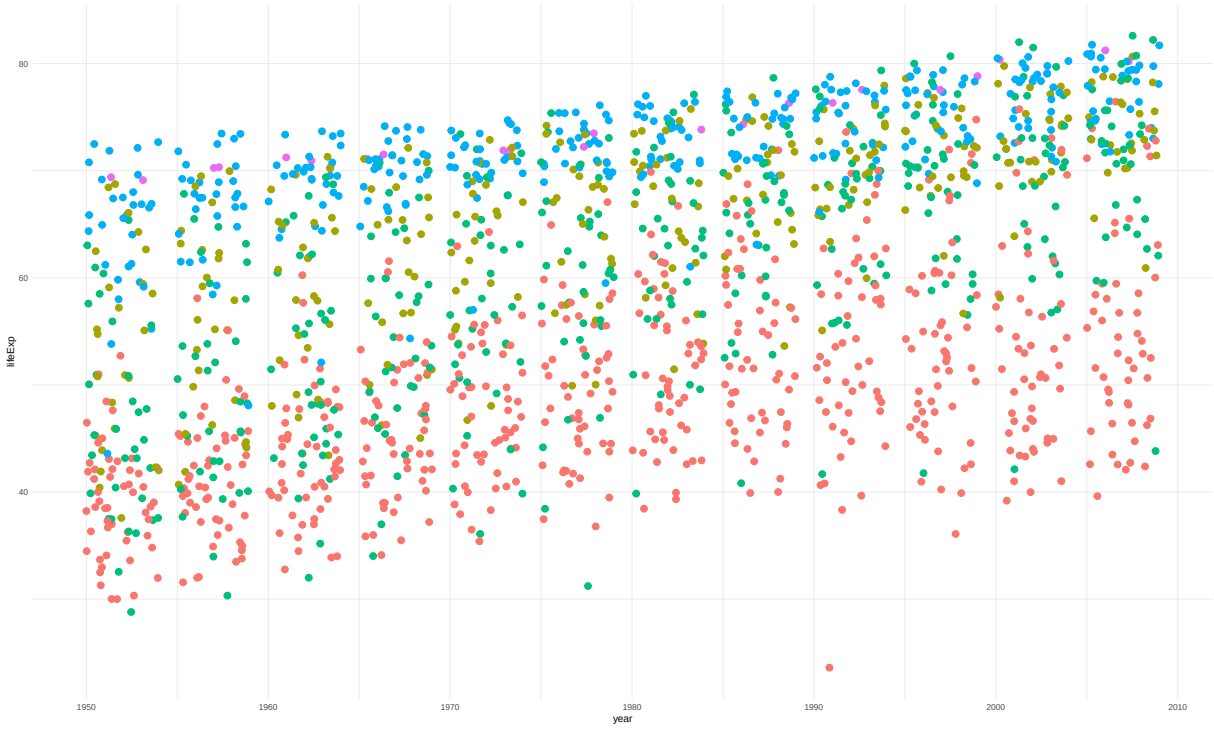
Por ejemplo, usamos `labs(color = "")` o `labs(fill = "")` para cambiar el título de las leyendas asociadas a colores o rellenos:

```
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  geom_jitter() +
  labs(color = "My new legend")
```



Por otro lado, con `guides(color = "none")` hacemos desaparecer la leyenda asociada al color:

```
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  geom_jitter() +
  guides(color = "none")
```



Con `theme(legend.position = "none")` hacemos desaparecer la leyenda completa:

```r
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  geom_jitter() +
  theme(legend.position = "none")
```

O también podemos definir una nueva ubicación para nuestra leyenda. Con `theme(legend.position = "bottom")` movemos la leyenda abajo:

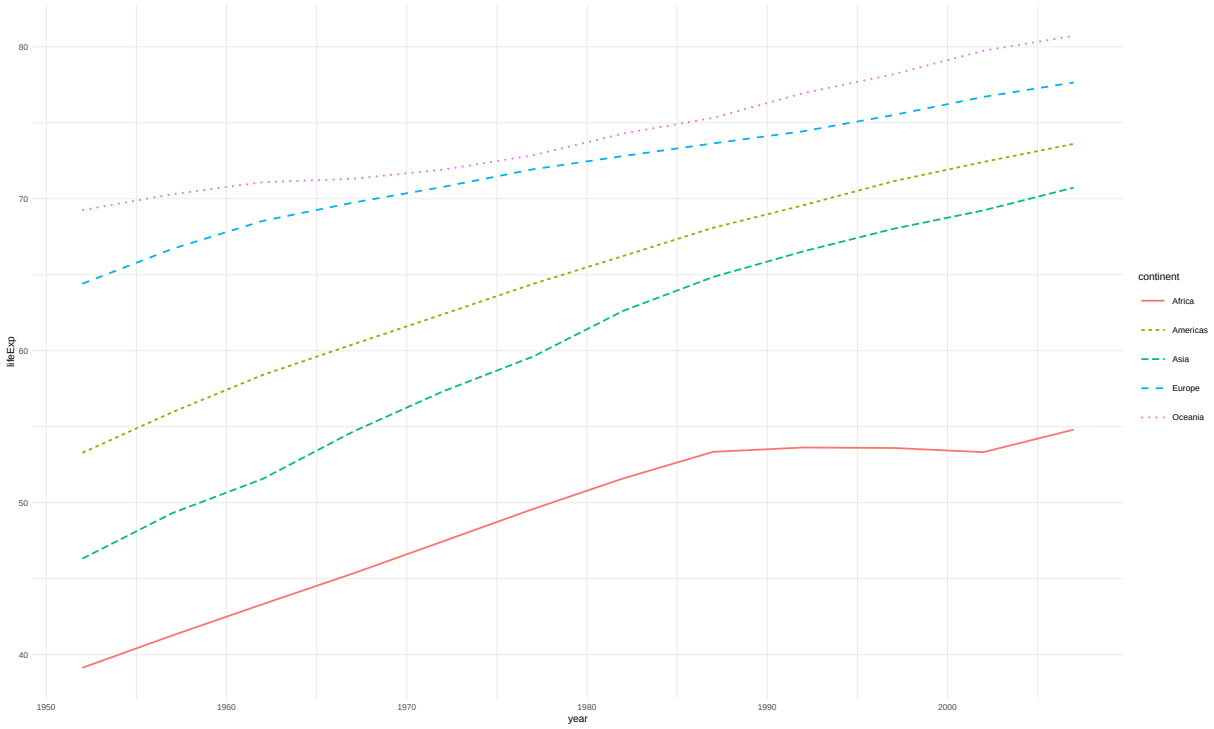
```r
ggplot(gapminder, aes(year, lifeExp, color = continent)) +
  geom_jitter() +
  theme(legend.position = "bottom")
```

3.3.3.1 Fancy pants legends

En ocasiones podemos simplificar notablemente las gráficas reemplazando la leyenda clásica por algo más moderno.

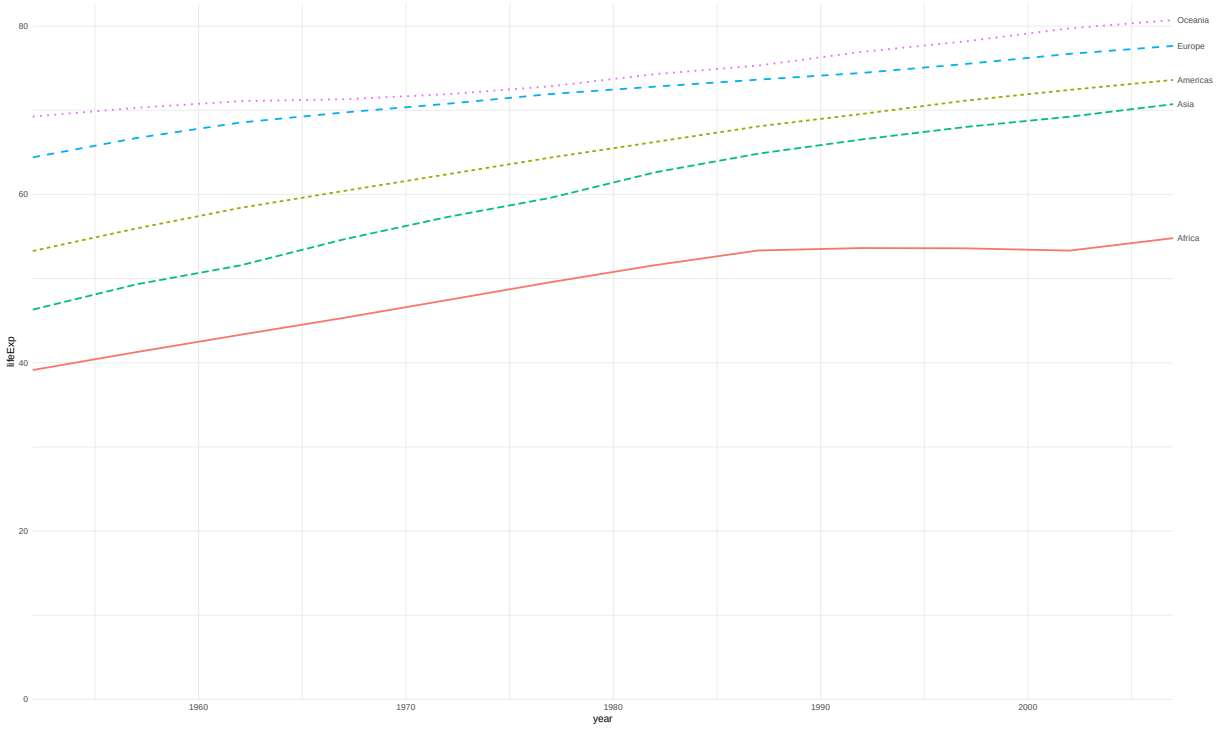
Podemos usar el eje secundario (derecho) para mostrar etiquetas. Partimos del gráfico siguiente:

```
ggplot(gapminder, aes(year, lifeExp, linetype = continent, color = continent)) +
  stat_summary(fun = mean, geom = "line")
```



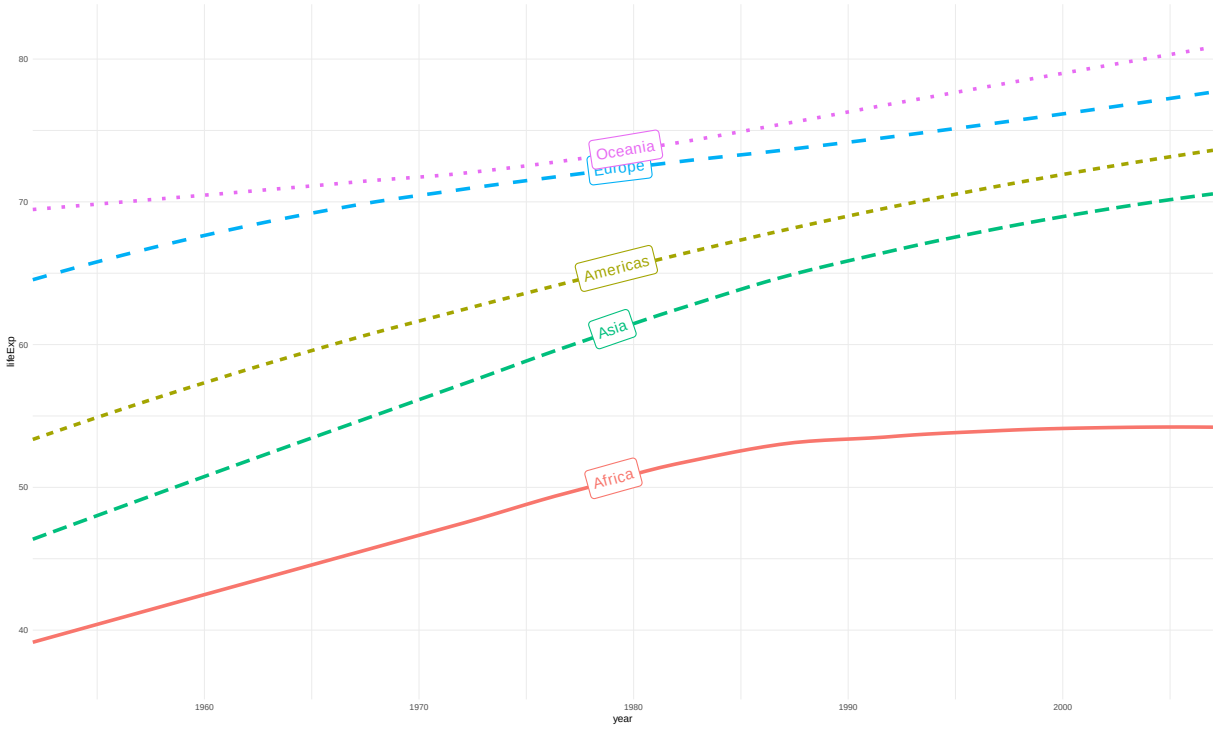
Usando un poco de vodoo, podemos convertirlo en esto:

```
gapminder_last = gapminder |>
  group_by(continent) |>
  filter(year == max(year)) |>
  summarize(lifeExp = mean(lifeExp))

ggplot(gapminder, aes(year, lifeExp, linetype = continent, color = continent)) +
  stat_summary(fun = mean, geom = "line") +
  scale_y_continuous(
    limits = c(0, max(gapminder$lifeExp)),
    expand = c(0,0),
    sec.axis = dup_axis(
      breaks = gapminder_last$lifeExp,
      labels = gapminder_last$continent,
      name = NULL)) +
  scale_x_continuous(expand = c(0,0)) +
  guides(color = "none",
         linetype = "none")
```



Otra estrategia interesante es colocar las etiquetas en el camino de las líneas. Para ello, necesitaremos la función `geom_labelsmooth()` del paquete {geomtextpath}:

```
ggplot(gapminder,
       aes(year, lifeExp, linetype = continent, color = continent)) +

  geomtextpath::geom_labelsmooth(
    aes(label = continent),
    text_smoothing = 30,
    method = "loess",
    formula = y ~ x,
    size = 3,
    linewidth = 1,
    boxlinewidth = 0.3
  ) +

  scale_x_continuous(expand = c(0, 0)) +
  guides(color = "none",
         linetype = "none")
```



Ejercicios

Ejercicio básico

El plot del panel (A) tiene varios problemas:

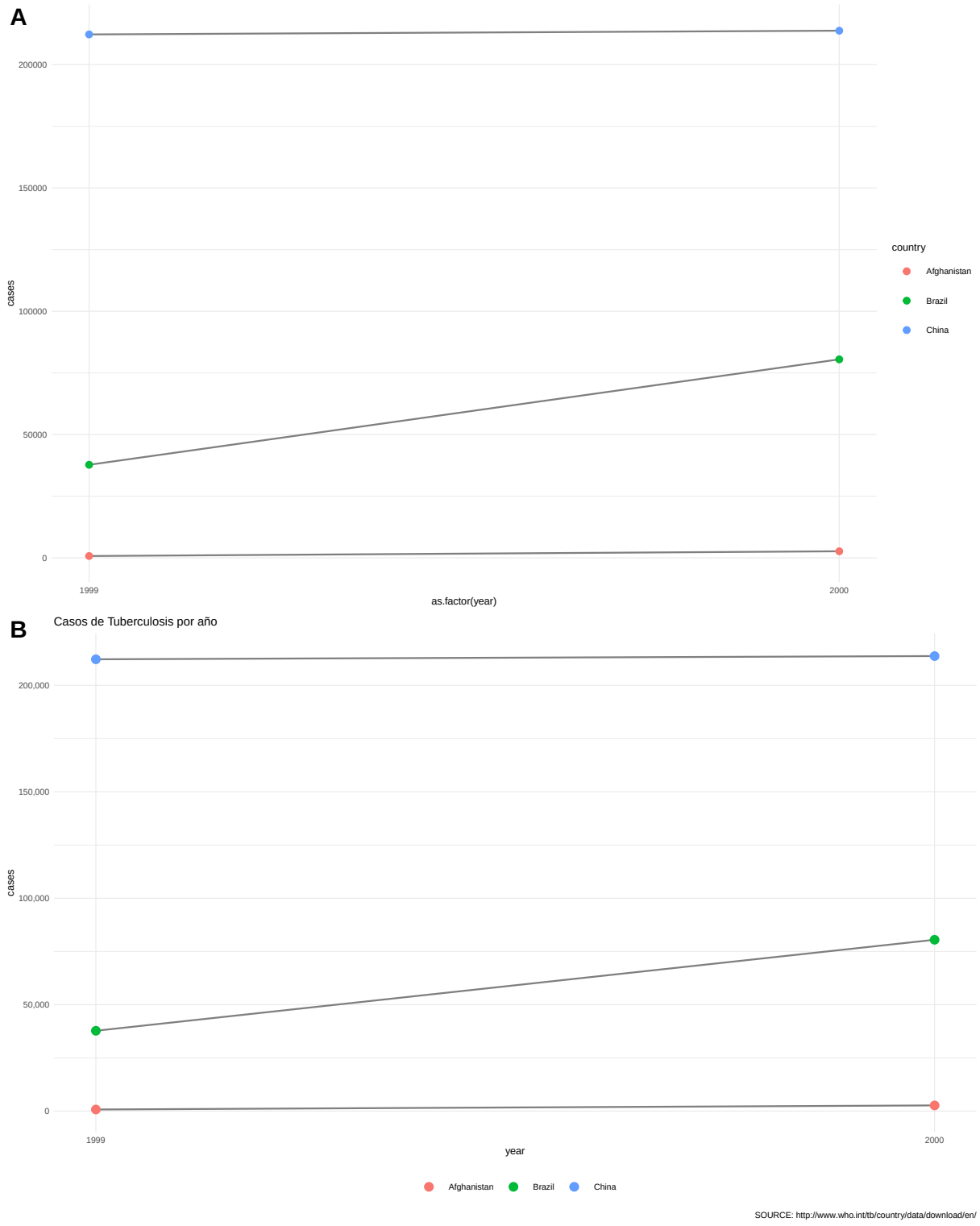
- los casos no se muestran con un separador de miles
- la leyenda esta a la derecha ocupado un espacio precioso, debería estar abajo
- al gráfico le falta el título, y caption
- la etiqueta del eje x debería ser `year` en lugar de `as.factor(year)`

Usando el plot base (A):

```
ggplot(table1, aes(as.factor(year), cases)) + # Usamos as.factor(year) para evitar que se mu
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country)) +
  scale_x_discrete(expand = c(.05, 0)) # Movemos las etiquetas del eje x hacia los extremos
```

Trata de resolver los problemas e intenta llegar al resultado que se ve en el panel (B).

> Soluciones
>
> - Recuerda la función`scales::comma()` que vimos más arriba - `theme(legend.position = "ALGO AQUI")` nos permite mover la leyenda. Si vas la ayuda de `theme()` y buscas `legend.position`, encontrarás sus opciones. - Los parámetros de `labs()` nos permiten añadir títulos, subtítulos, editar los valores de las etiquetas de `x` e `y`, añadir `caption`, etc.

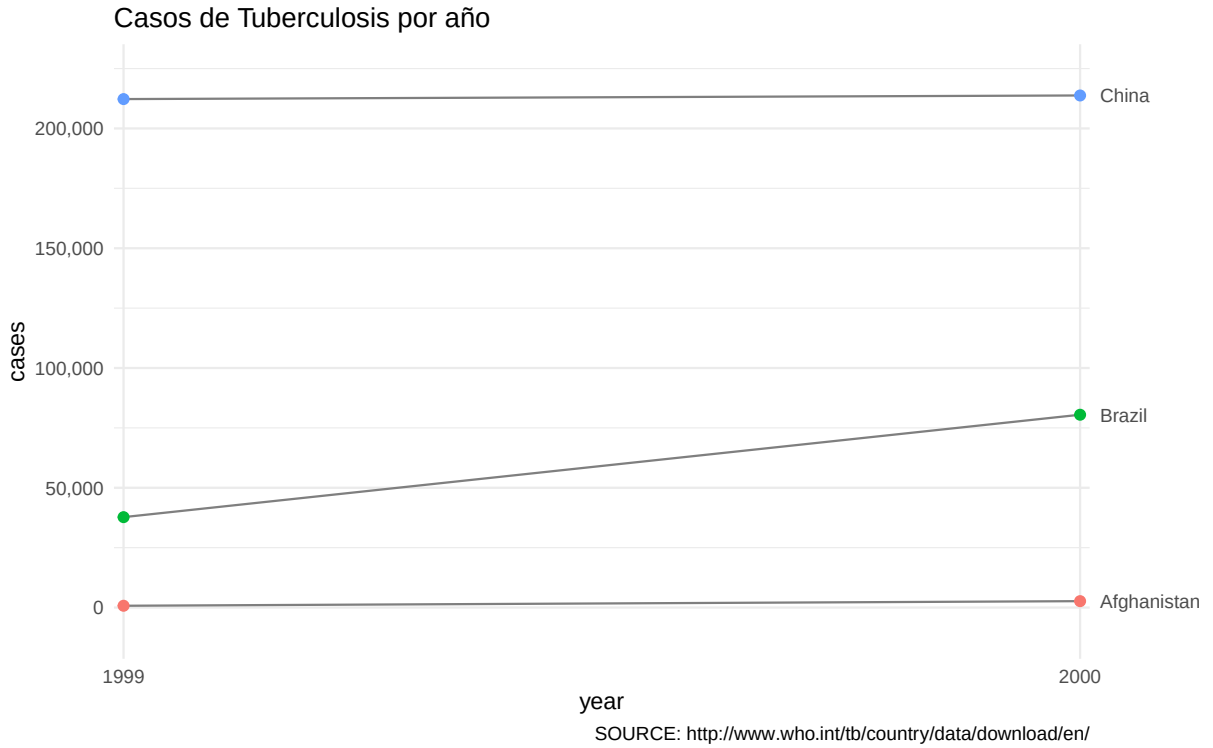


Ejercicio avanzado

Si te sobra tiempo, puedes tratar de reproducir la siguiente versión mejorada...

Soluciones

Hemos visto como hacer esto en el primer ejemplo de fancy pants legends

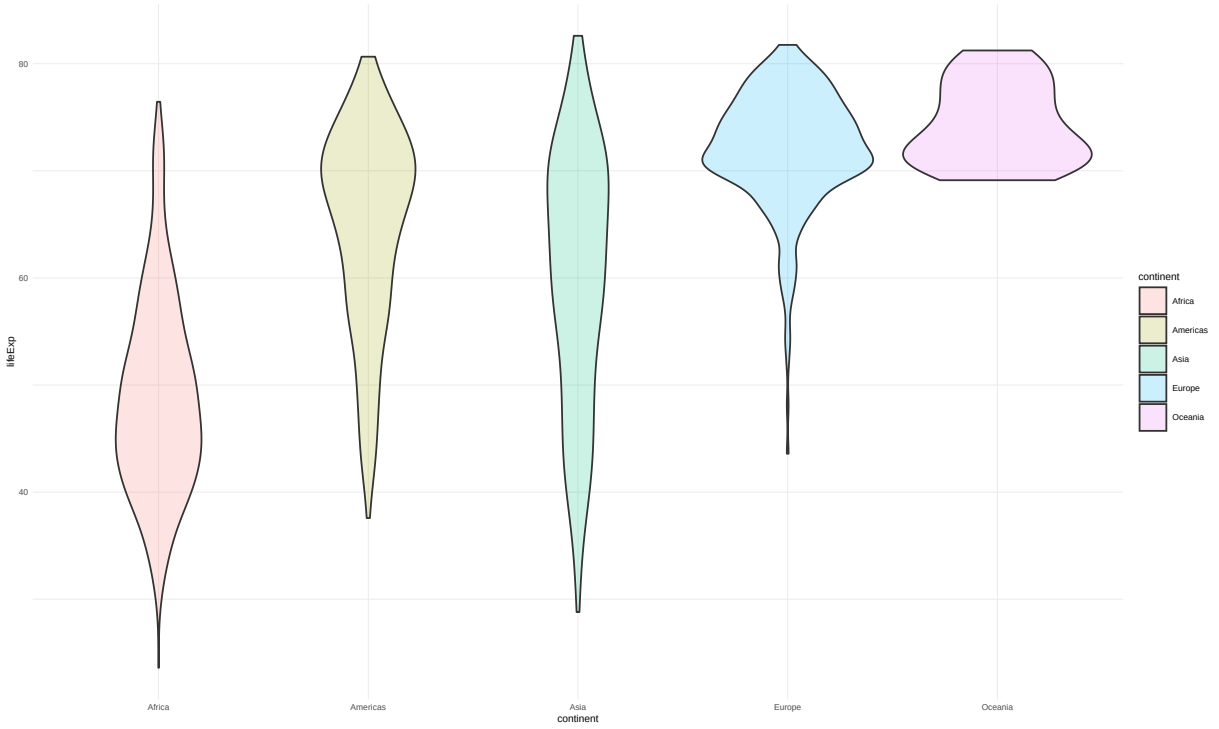


3.3.4 Colors and fill scales

Las funciones `scale_color_`, `scale_fill_` nos sirven para hacer cambios globales en los colores o rellenos de las gráficas. Algunos ejemplos:

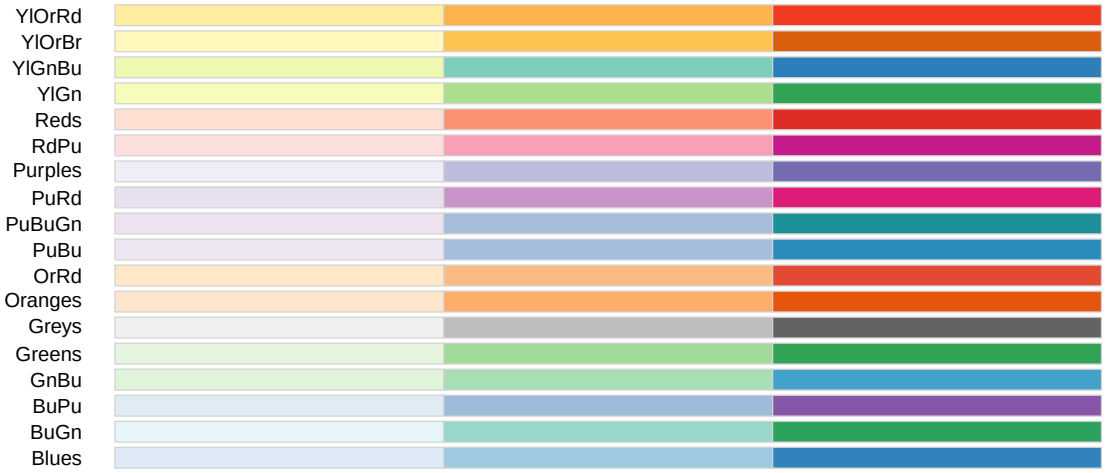
Plot inicial:

```
# Plot inicial
ggplot(gapminder, aes(continent, lifeExp, fill = continent)) +
  geom_violin(alpha = .2)
```



Podemos usar diferentes paletas de colores preexistentes. Una manera de consultar las paletas disponibles es con `RColorBrewer::display.brewer.all()`:

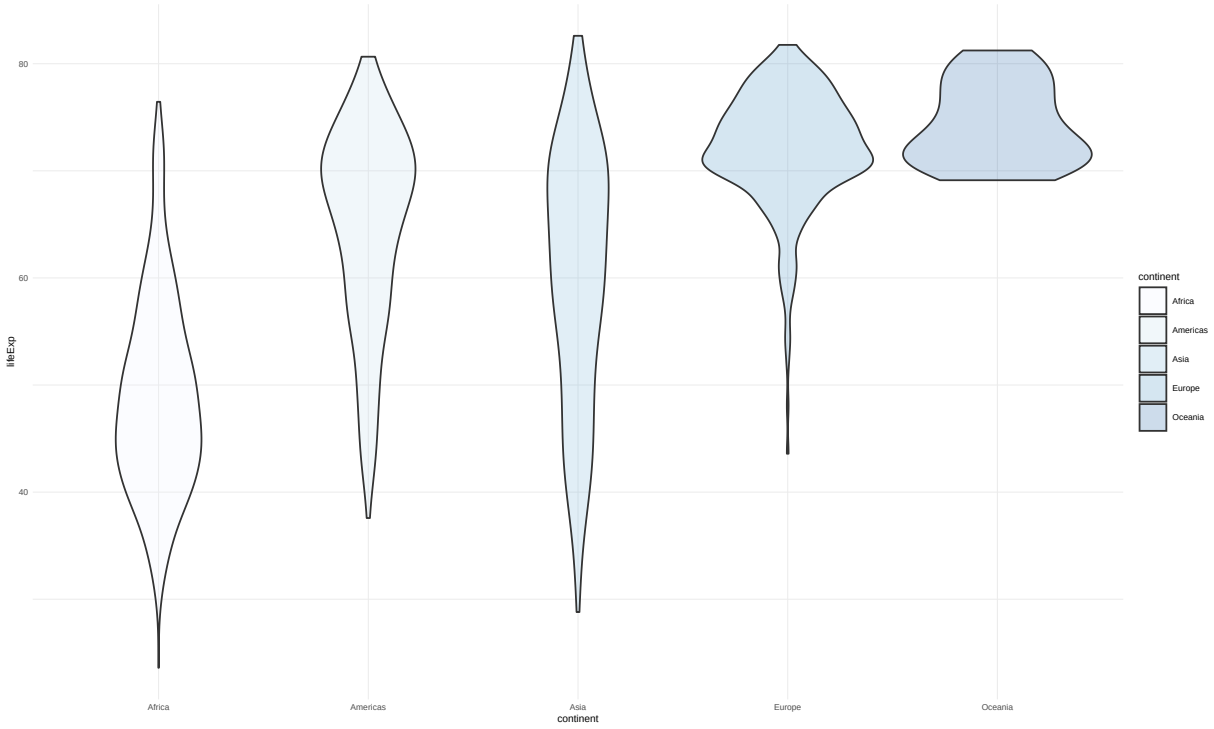
```
RColorBrewer::display.brewer.all(n = 3,
                                 type = "seq", # Colores secuenciales
                                 exact.n = FALSE,
                                 colorblindFriendly = TRUE)
```



Podemos ver que el parámetro `type` nos permite seleccionar paletas secuenciales, divergentes y cualitativas. Por ejemplo, `type = "qual"` nos mostrará colores de paletas cualitativas (¡pruébalo!).

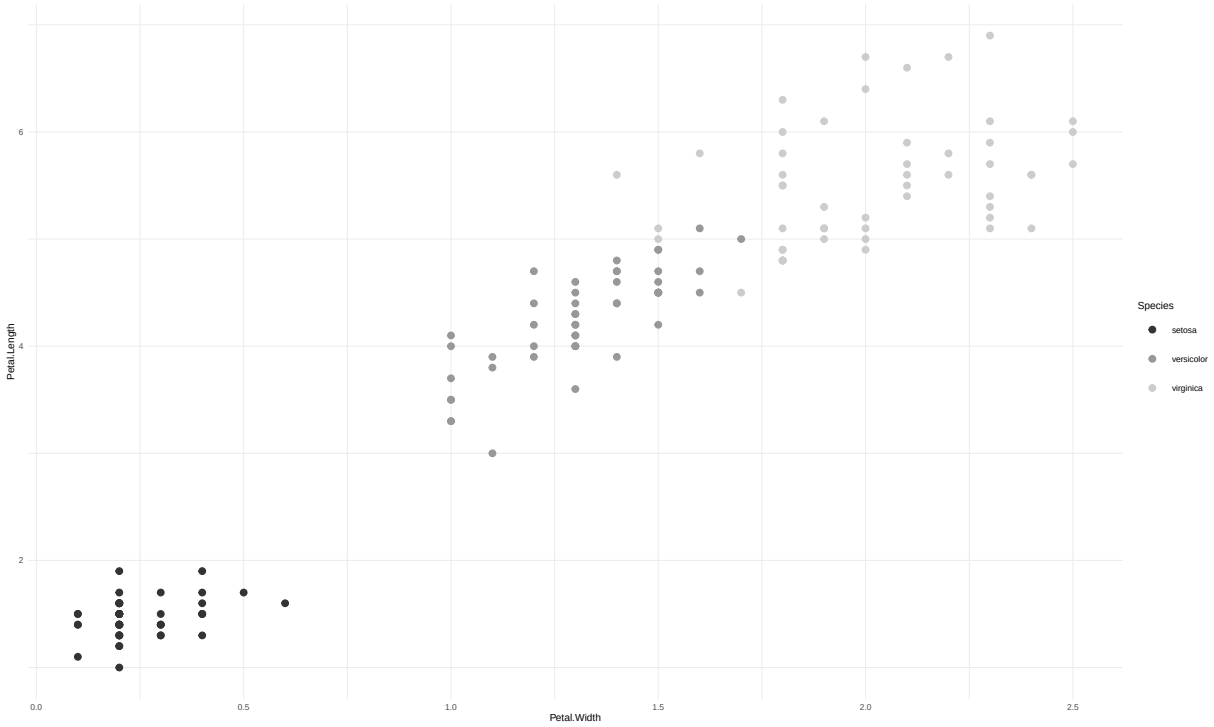
Una vez elegía nuestra paleta, la podemos aplicar con `scale_fill_brewer()` para `fill` o con `scale_color_brewer_brewer()` para `color`. En este caso usamos la `palette = "Blues"`:

```
ggplot(gapminder, aes(continent, lifeExp, fill = continent)) +
  geom_violin(alpha = .2) +
  scale_fill_brewer(palette = "Blues")
```



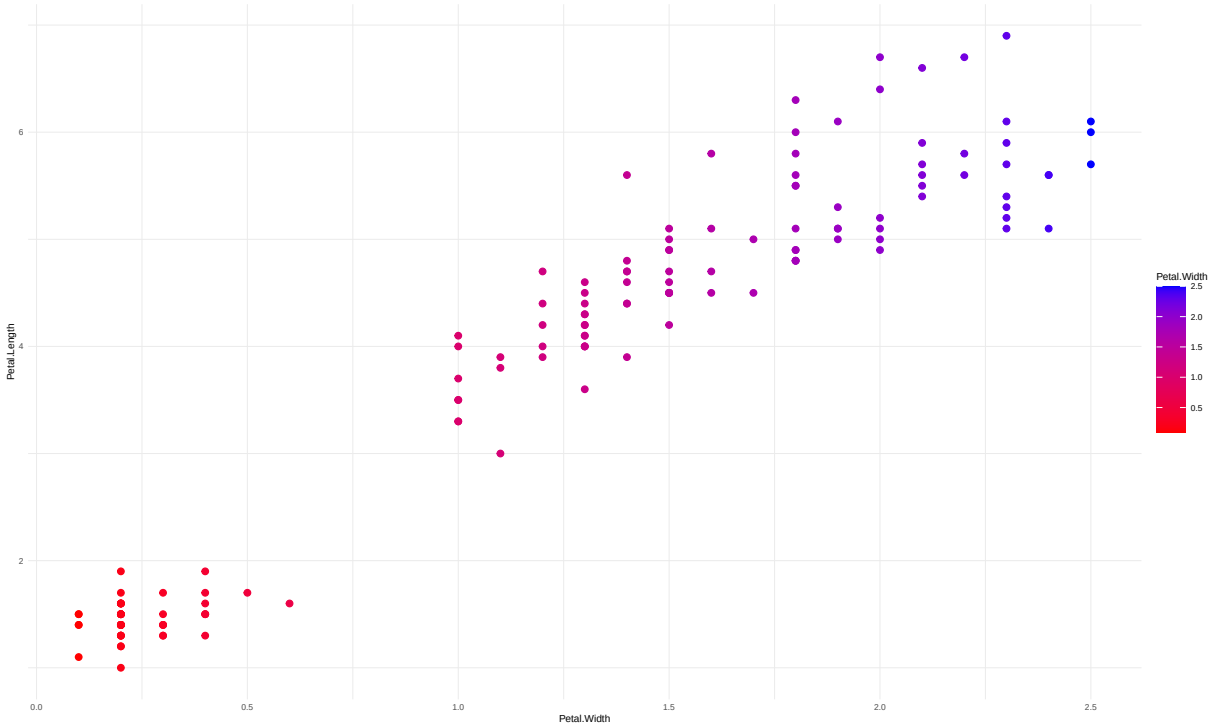
Usamos `scale_color_grey()` para estala de grises:

```
ggplot(iris, aes(Petal.Width, Petal.Length, color = Species)) +
  geom_point() +
  scale_color_grey(start = 0.2, end = 0.8, na.value = "red")
```



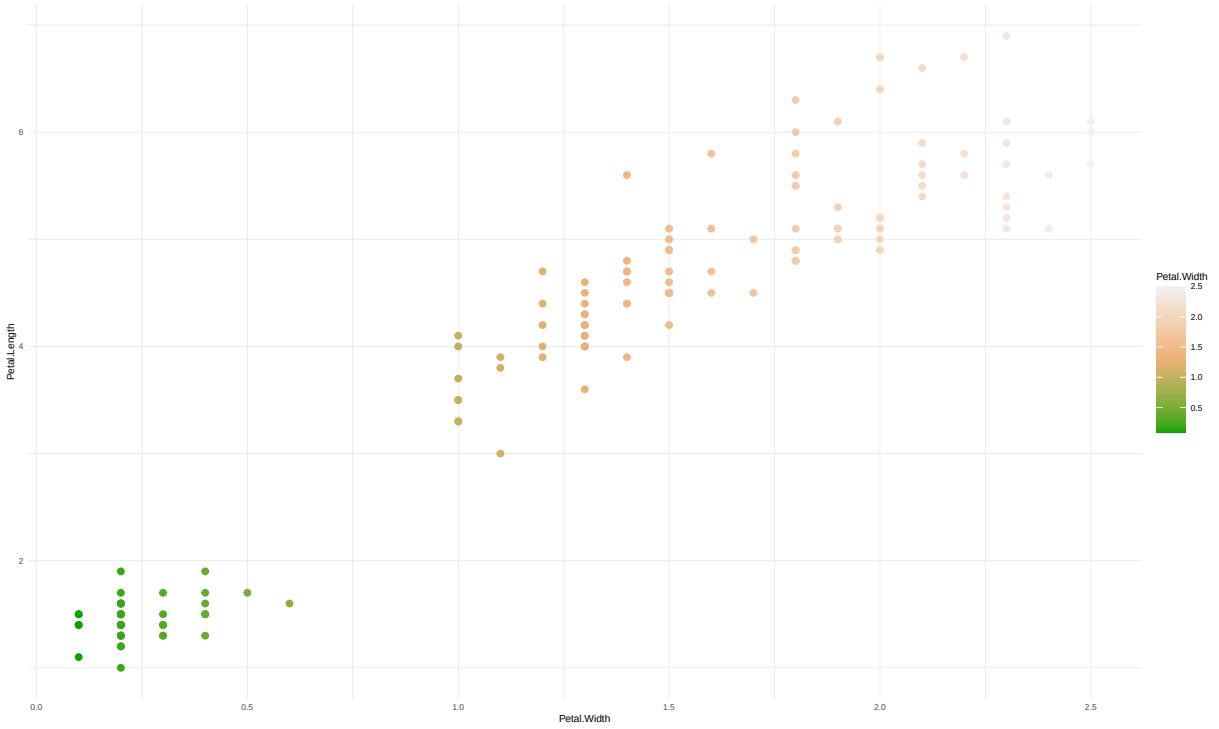
`scale_color_gradient()` para gradientes:

```r
ggplot(iris, aes(Petal.Width, Petal.Length, color = Petal.Width)) +
  geom_point() +
  scale_color_gradient(low = "red", high = "blue")
```



O `scale_colour_gradientn()` con un número predeterminado de colores definido por la paleta `terrain.colors()`:

```r
# Gradient con un numero predefinidos de una paleta
ggplot(iris, aes(Petal.Width, Petal.Length, color = Petal.Width)) +
  geom_point() +
  scale_colour_gradientn(colours = terrain.colors(3))
```



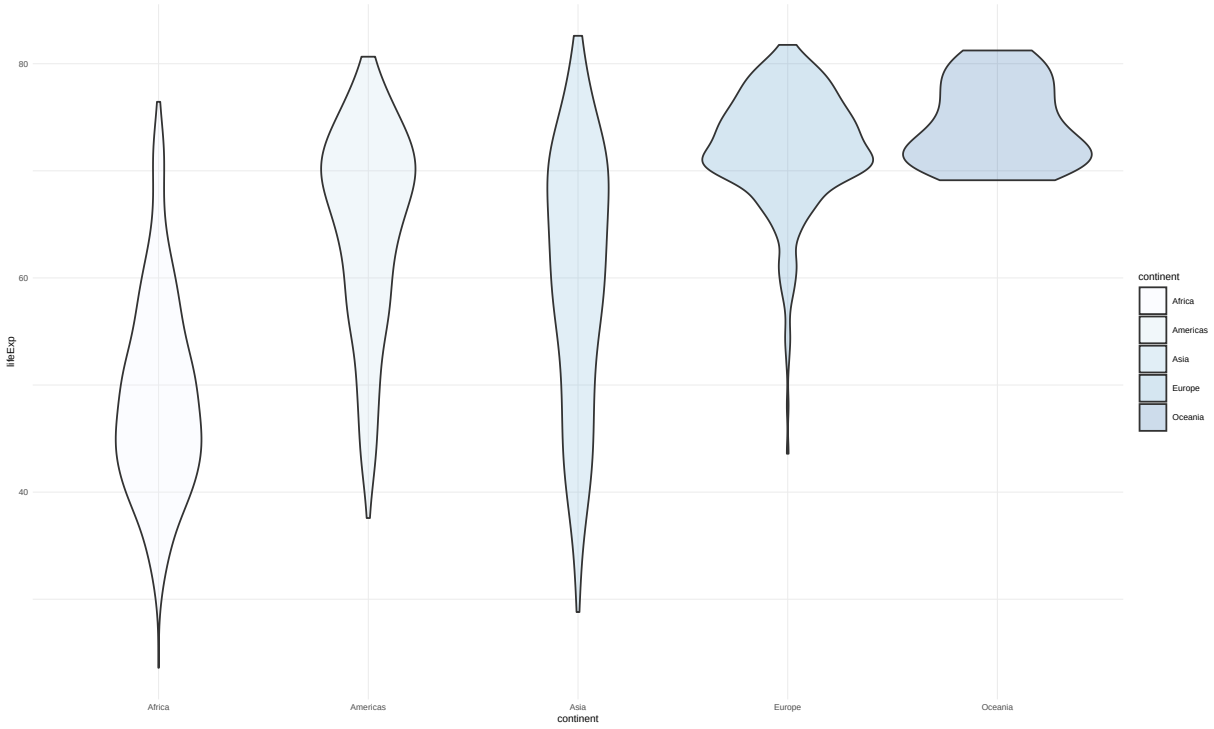
Ejercicios

Ejercicio básico

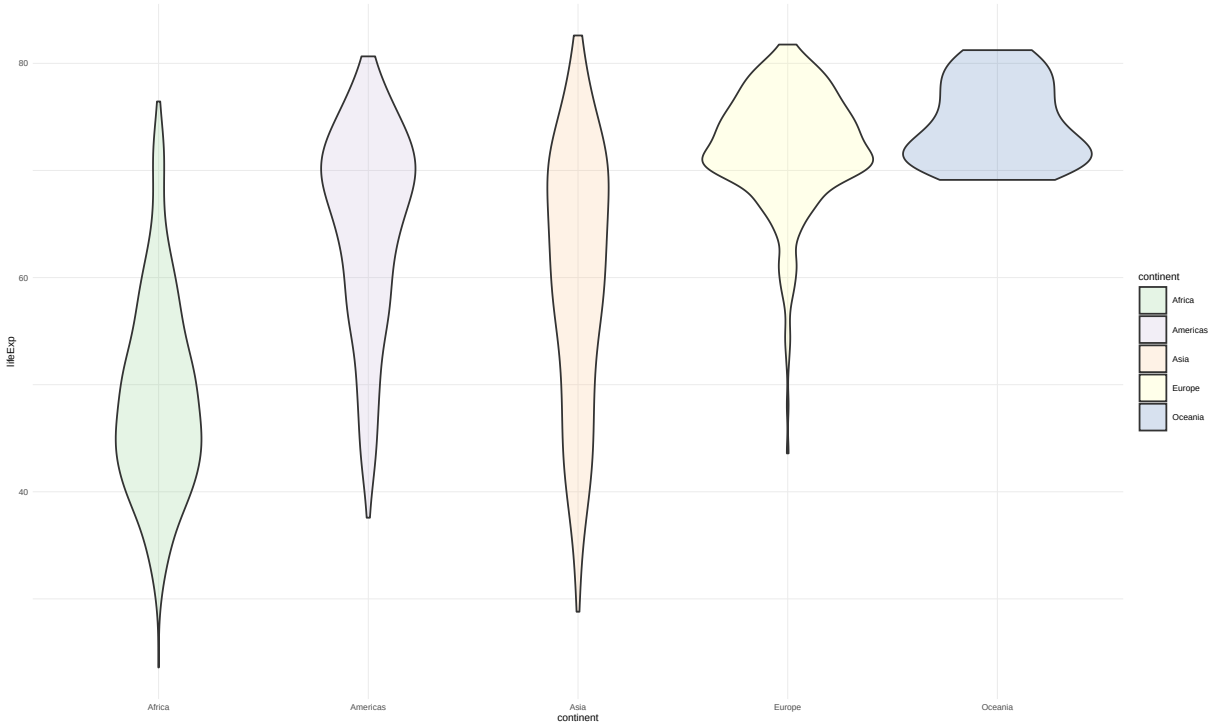
Usando como base este plot, podrías cambiarle la paleta de color para usar una de las paletas cualitativas?

> Soluciones
>
> Ver parámetro `type` de la ayuda de `scale_fill_brewer()`

```
ggplot(gapminder, aes(continent, lifeExp, fill = continent)) +
  geom_violin(alpha = .2) +
  scale_fill_brewer(palette = "Blues")
```



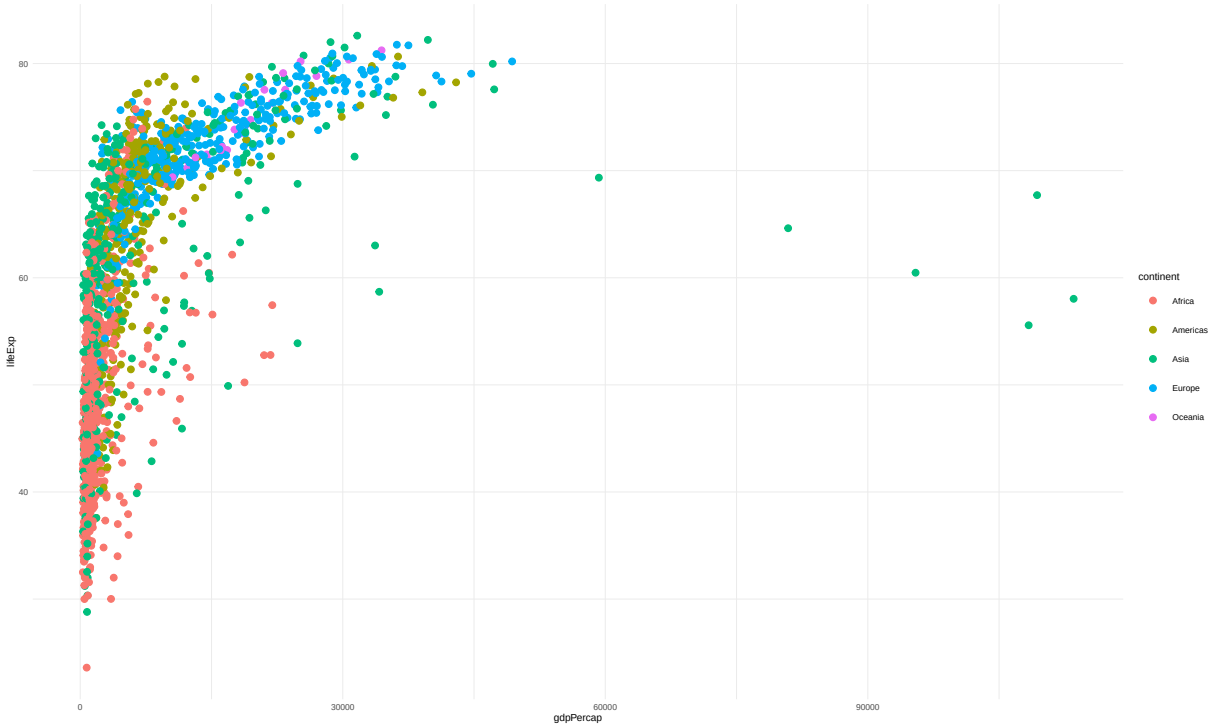
El gráfico final debería verse así:



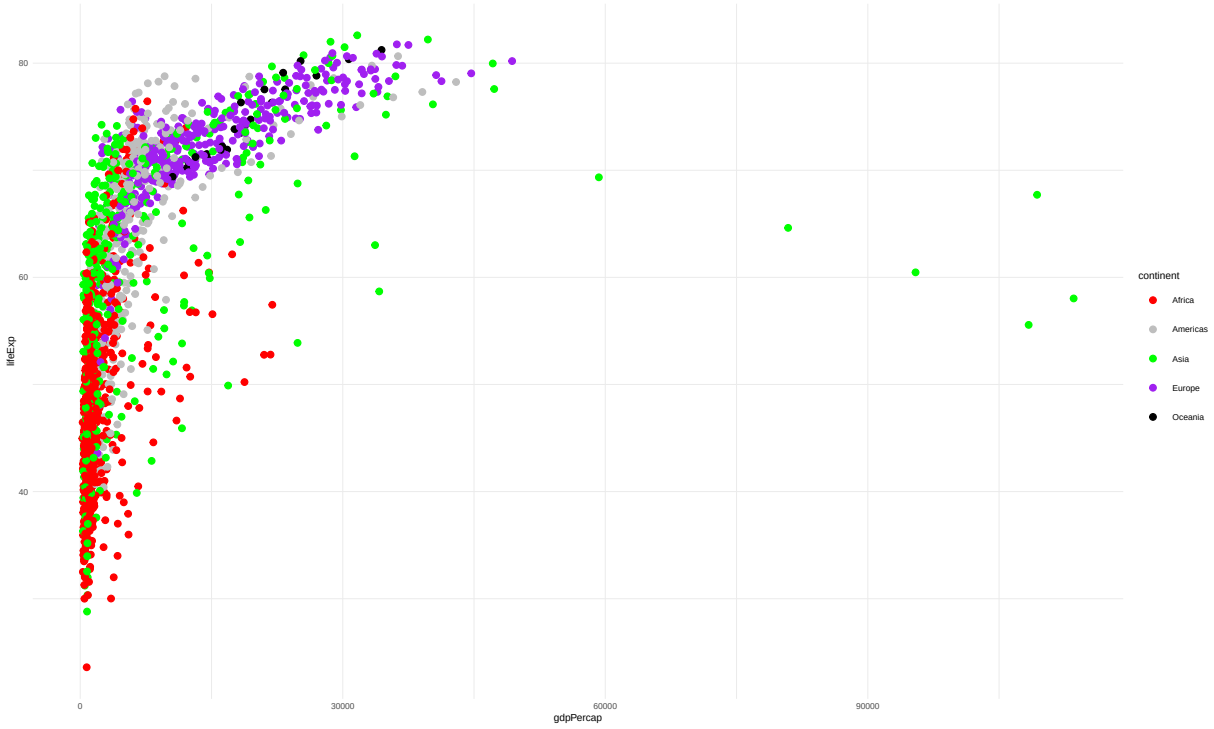
Ejercicio avanzado

Ahora, usando el gráfico de abajo como base, puedes asignar manualmente colores a los continentes?

```r
ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point()
```



Un ejemplo de resultado posible es este:



Nuestra primera idea podría ser asignar colores directamente dentro de `geom_point()` tal y como se ve abajo. Pero si intentamos asignar colores manualmente a los continentes de este

modo, recibimos un error:

```
ggplot(gapminder, aes(gdpPercap, lifeExp)) +
  geom_point(color = c("red", "grey", "green", "purple", "black"))
#  Aesthetics must be either length 1 or the same as the data (1704)
#   Fix the following mappings: `colour`
```

Soluciones

Tenemos que:

- indicar que el color depende de `continent` dentro de `aes()`
- usar `scale_color_manual()`, con el parámetro `values` para asignar los colores (`values = c("green", "blue", ...)`; ver ejemplos en la ayuda de la función)

3.3.5 Combinando gráficas

Con {cowplot} podemos combinar gráficas de manera muy simple. Otro paquete muy interesante es {patchwork}.

```
plot1 = ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(alpha = .1) +
  scale_y_continuous(breaks = seq(0, 100, 5)) +
  scale_x_log10(labels = scales::dollar_format(prefix = "$", suffix = "M")) +
  theme(legend.position = "top")

plot2 = ggplot(gapminder, aes(continent, ..prop.., group = 1)) +
  geom_bar() +
  scale_y_continuous(labels = scales::percent) +
  coord_flip()

cowplot::plot_grid(plot2, plot1, rel_widths = c(.3, 0.7))
```

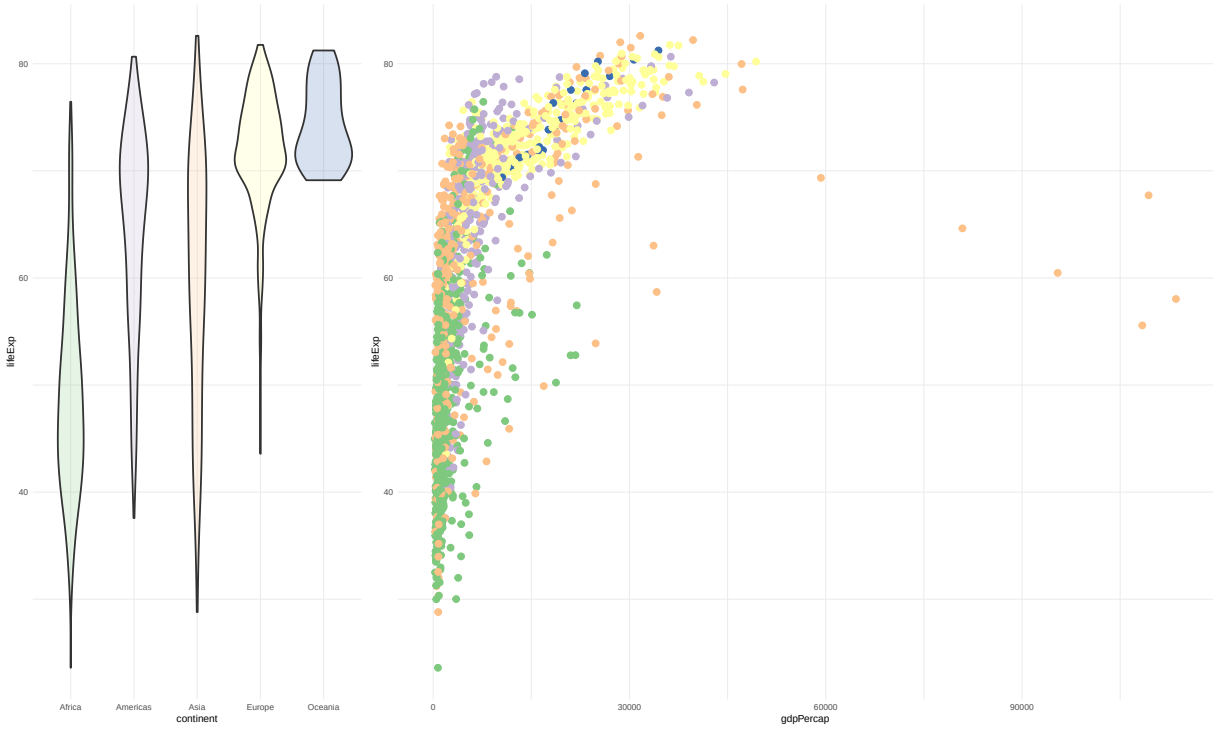
Ejercicio

1) Combina los dos plots del ejercicio anterior, con las siguientes modificaciones:

- Elimina las leyendas asociadas a colores y rellenos
- Usa la paleta "Accent" para los colores y rellenos

Soluciones

- guides(fill = "none") quita la leyenda asociada a `fill`… - scale_fill_brewer(palette = "Accent") asigna la paleta "Accent" a los rellenos (`fill`)

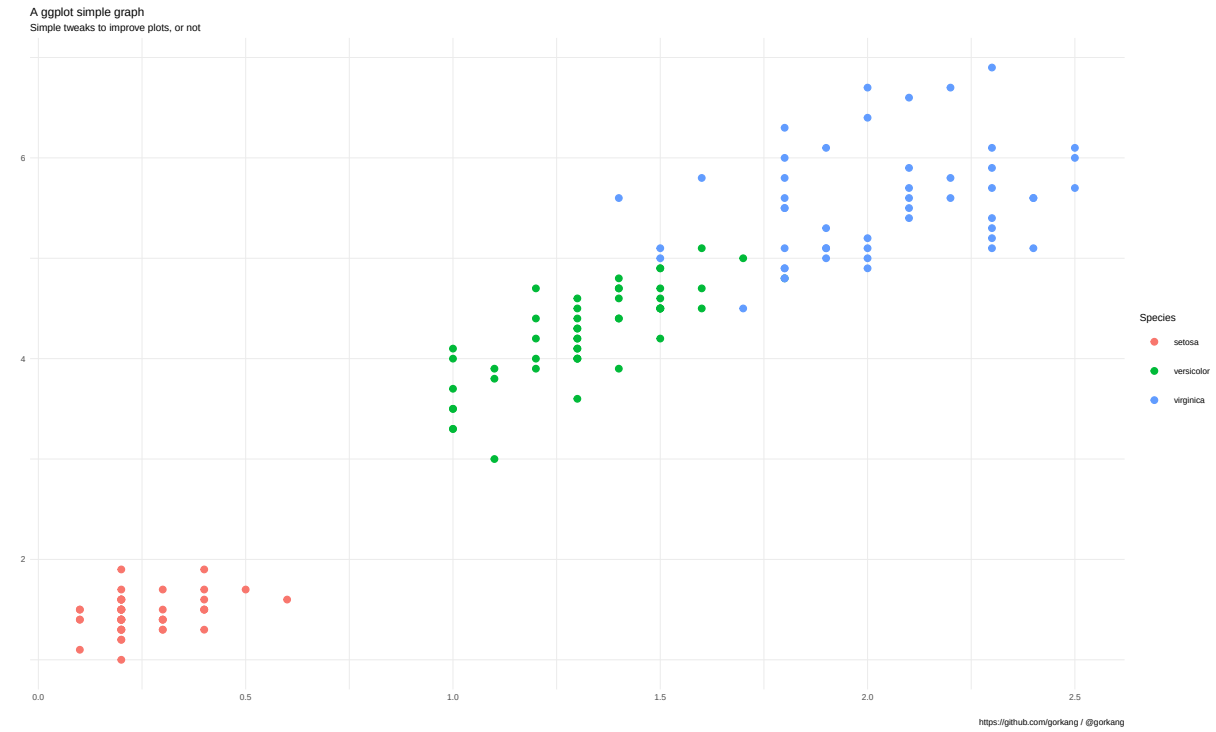


3.3.6 Estilos

Los estilos nos permiten personalizar los gráficos de manera muy sencilla, por ejemplo, usando {ggtheme}. Podéis ver un tutorial aquí.

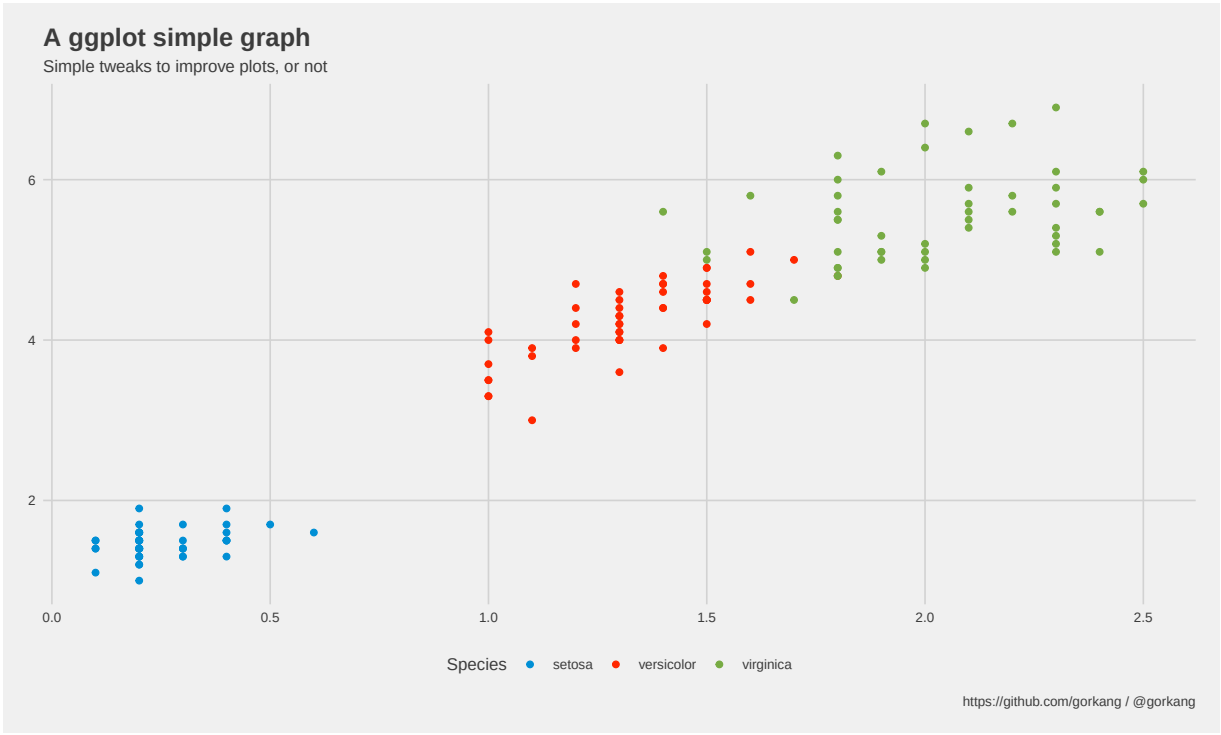
Primero creamos un gráfico sobre el que aplicaremos estilos.

```
p <- ggplot(iris, aes(Petal.Width, Petal.Length, color = Species)) +
  geom_point() +
  labs(title = 'A ggplot simple graph',
       subtitle = 'Simple tweaks to improve plots, or not',
       x = '',
       y = '',
       caption = 'https://github.com/gorkang / @gorkang')

p
```



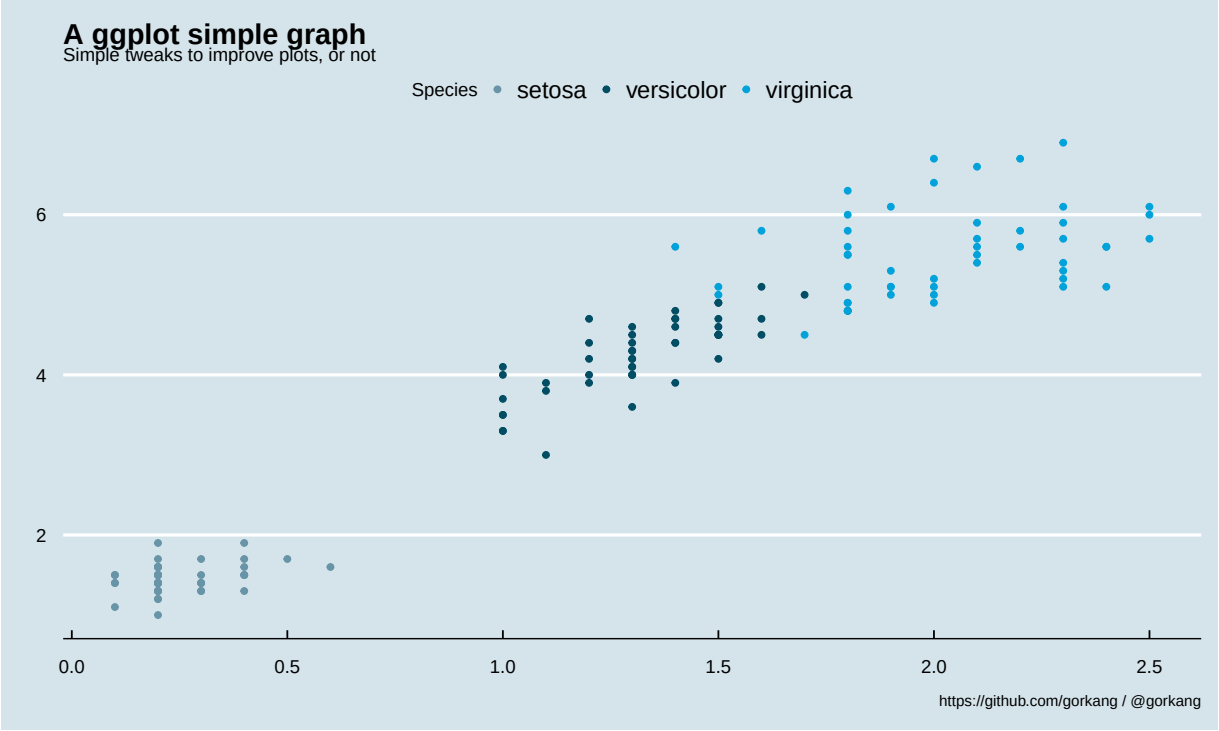
Usando el tema fivethirtyeight:

```
p +
  ggthemes::scale_color_fivethirtyeight() +
  ggthemes::theme_fivethirtyeight(base_size = 10)
```



Usando el tema economist:

```
p +
  ggthemes::scale_color_economist() +
  ggthemes::theme_economist(base_size = 10)
```



Ejercicios

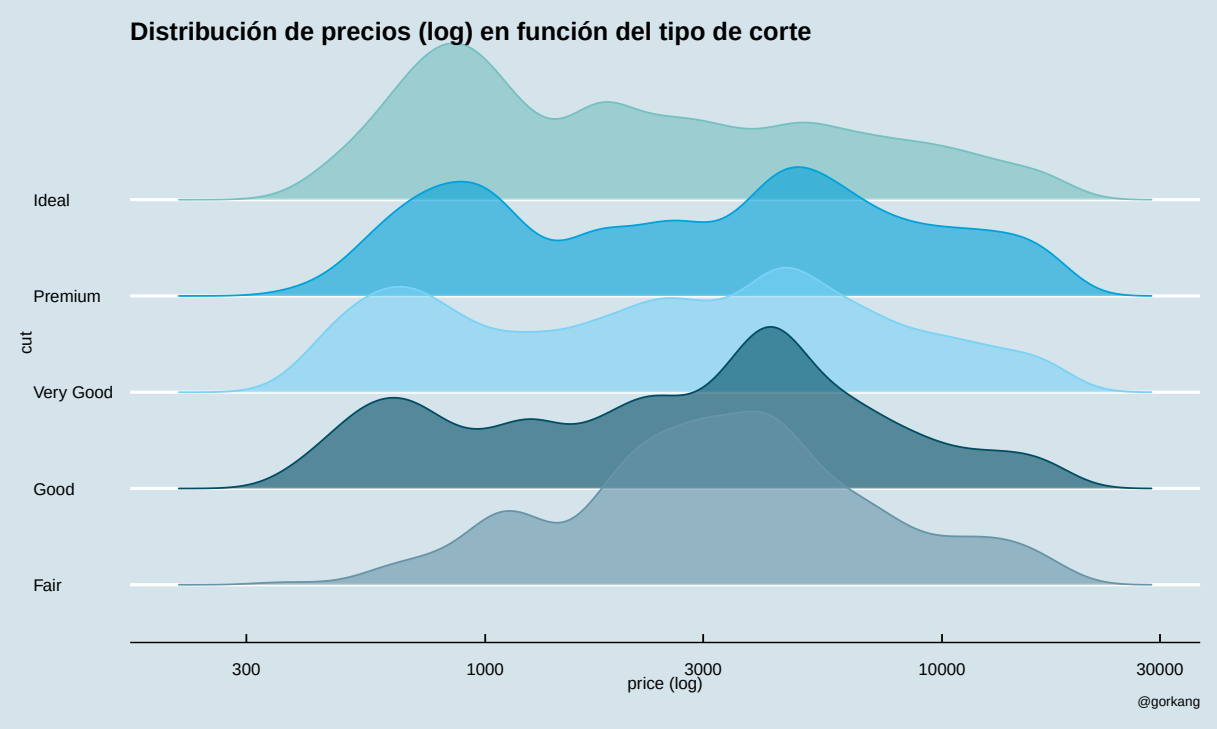
1) Serías capaz de reproducir este gráfico, usando el data frame `diamonds` y el `theme_economist`?

Gráfica inicial (verás que parecen datos distintos! Si te fijas bien en el eje `x` de la gráfica que queremos conseguir, entenderás porque):

```
ggplot(diamonds, aes(price, cut, fill = cut, color = cut, group = cut)) +
  ggridges::geom_density_ridges(alpha = .6)
```

Soluciones

- scale_x_log10() nos permite transformar el eje x a una escala logarítmica - Hay que aplicar un `ggthemes::scale_*` para cada elemento: `color`, `fill`...



---

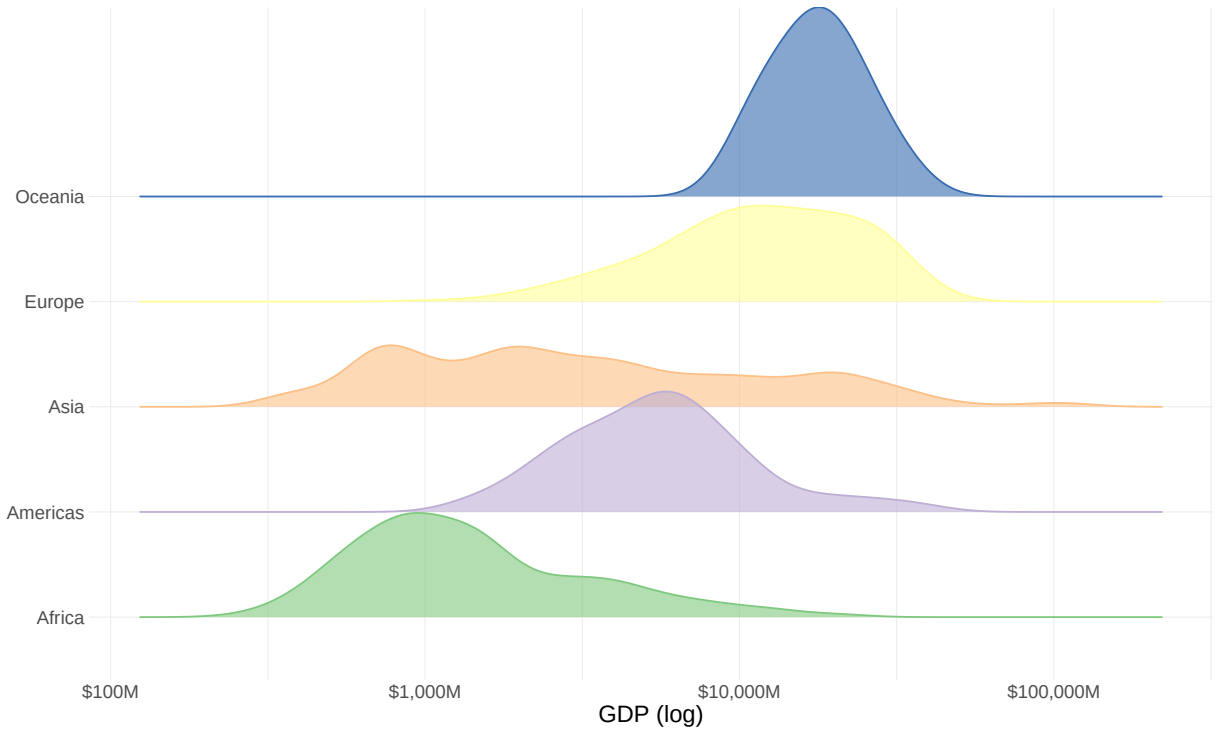
2) Serías capaz de reproducir este gráfico, usando el data frame `gapminder` y la paleta `Accent`?

Gráfica inicial:

```
ggplot(gapminder, aes(gdpPercap, continent, fill = continent, color = continent)) +
  ggridges::geom_density_ridges(alpha = .6)
```

Soluciones

- `scales::dollar_format()` aplicado al parámetro `labels` de las funciones `scale_x_*` nos permite darle formato de moneda a las etiquetas de la escala x

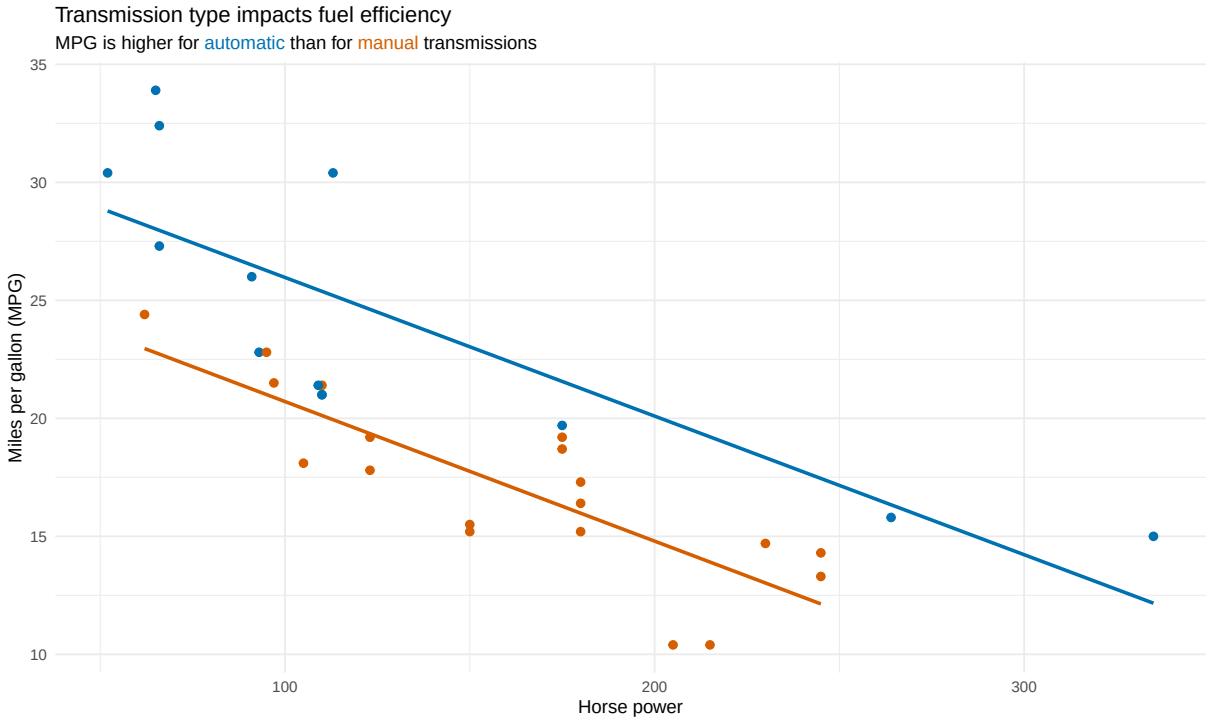


3.3.7 Estilos en textos

Con {ggtext} podemos incluir estilos en los textos, por ejemplo, en el título de nuestras gráficas.

```
# Ejemplo adaptado de https://wilkelab.org/ggtext/articles/theme_elements.html
mtcars |>
  mutate(
    transmission = ifelse(am == 1, "automatic", "manual")
  ) |>
  ggplot(aes(hp, mpg, color = transmission)) +
  geom_point(size = 2) +
  geom_smooth(se = FALSE, method = "lm") +
  scale_color_manual(
    values = c(automatic = "#0072B2", manual = "#D55E00"),
    guide = "none"
  ) +
  labs(
    x = "Horse power",
    y = "Miles per gallon (MPG)",
    title = "Transmission type impacts fuel efficiency",
    subtitle = "MPG is higher for <span style = 'color:#0072B2;'>automatic</span>
            than for <span style = 'color:#D55E00;'>manual</span> transmissions"

  ) +
  theme_minimal() +
  theme(
```

```
  # plot.title.position = "plot",
  plot.subtitle = element_markdown(size = 11, lineheight = 1.2)
)
```

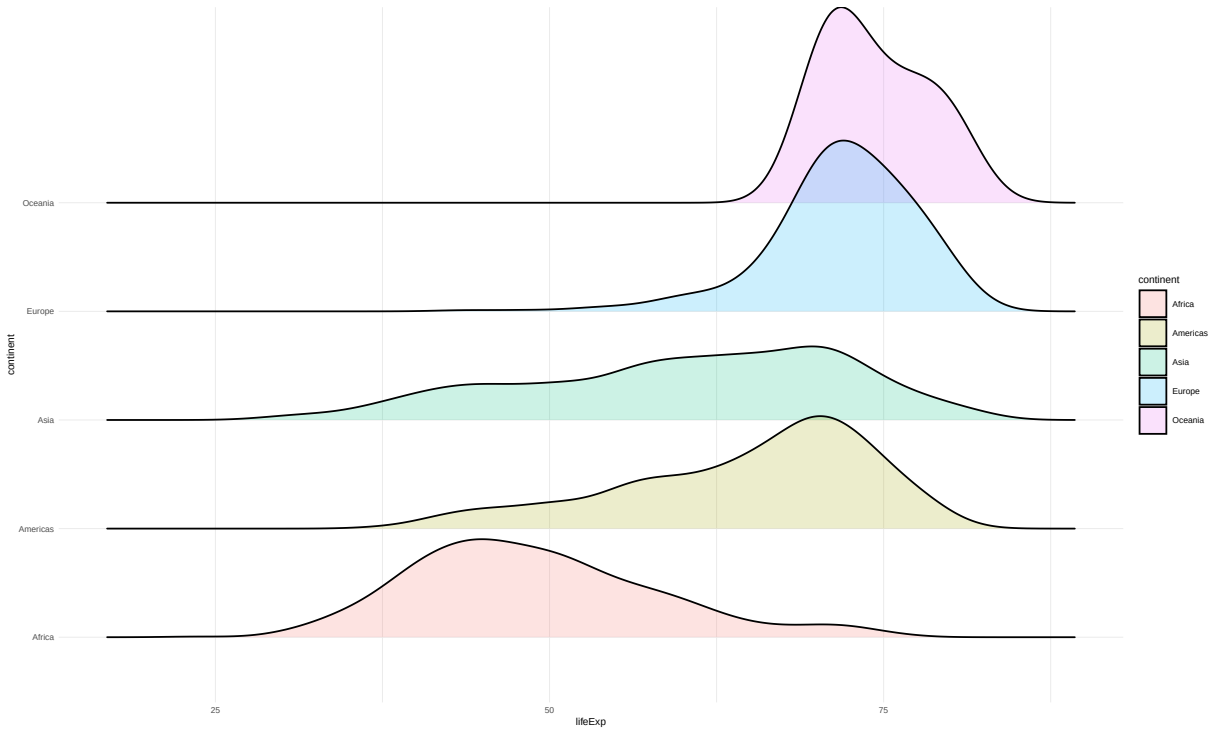


3.4 Otras gráficas

3.4.0.1 geom_density_ridges

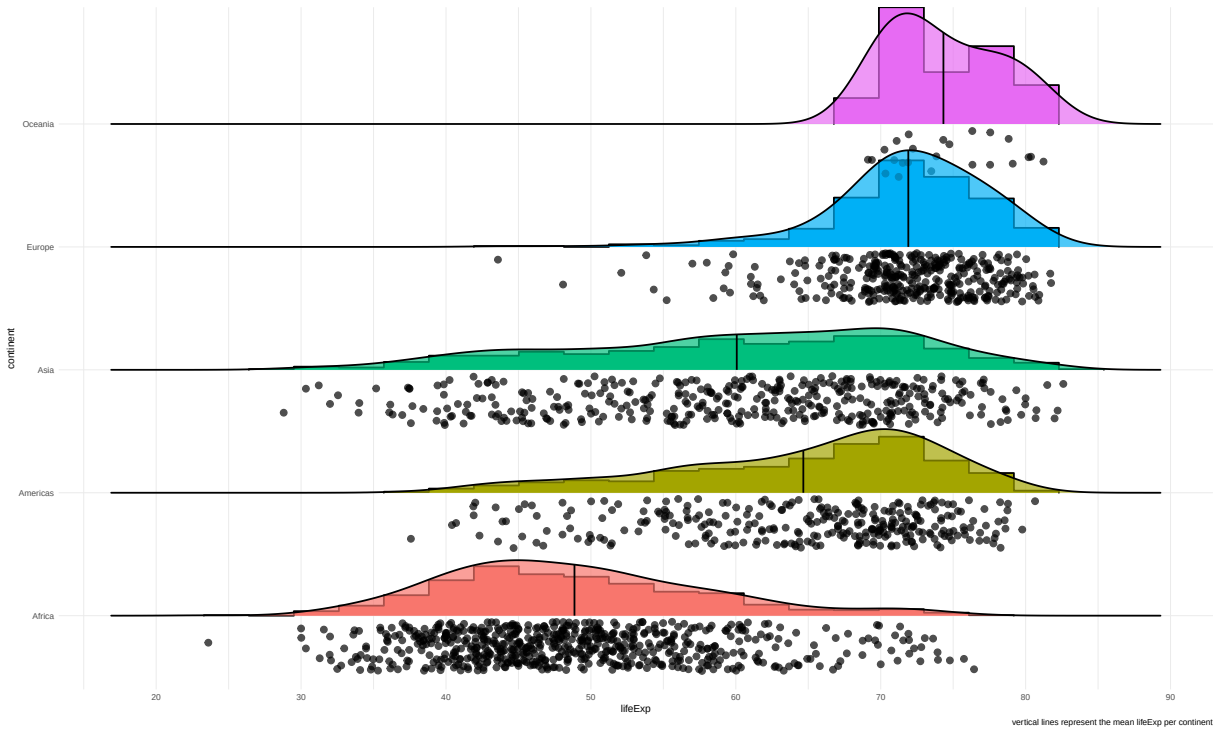
Uno de mis `geoms` favoritos para comparar distribuciones es `geom_density_ridges`:

```
ggplot(gapminder, aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(alpha = .2)
```



Especialmente porque podemos incluir en el mismo gráfico información sobre distribuciones y puntos individuales.

```
ggplot(gapminder, aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.7,
    scale = 0.9,
    quantile_lines = TRUE,
    quantile_fun = mean
  ) +
  theme(legend.position = "none") +
  scale_x_continuous(n.breaks = 10) +
  labs(caption = "vertical lines represent the mean lifeExp per continent")
```

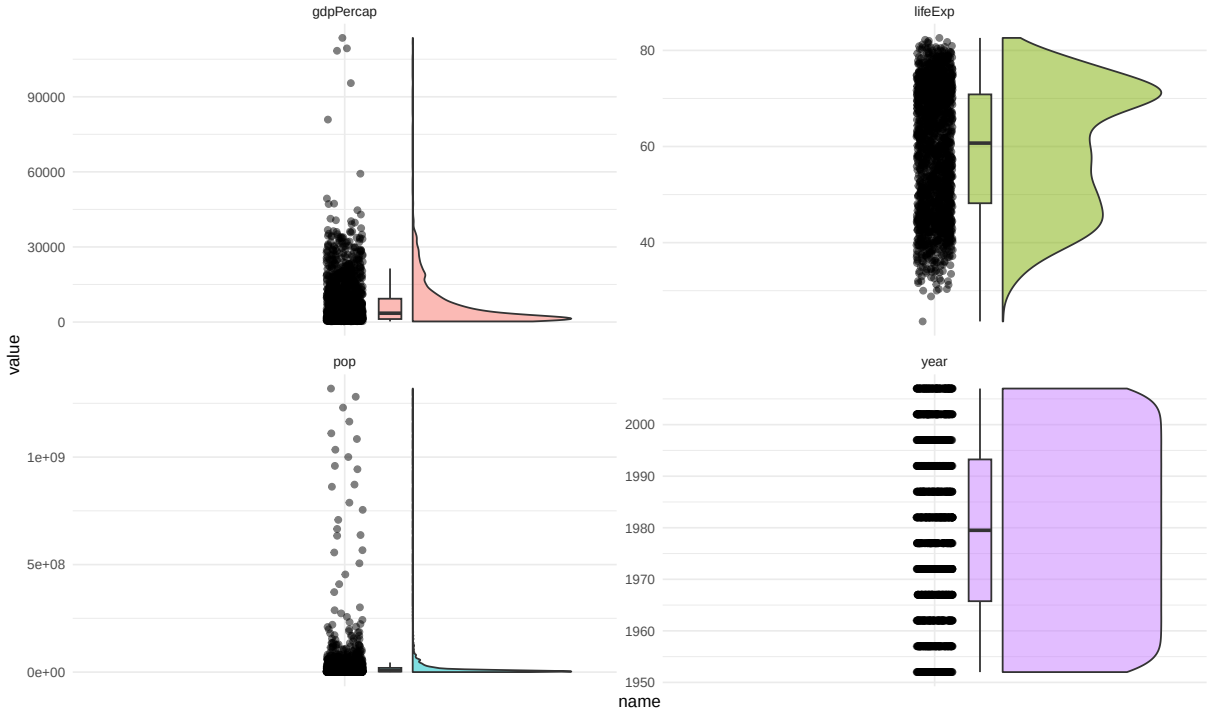


3.4.1 Raincloud plots

Otra gráfica genial para mostrar simultáneamente observaciones individuales, distribuciones, y cambios, es el raincloudplot.

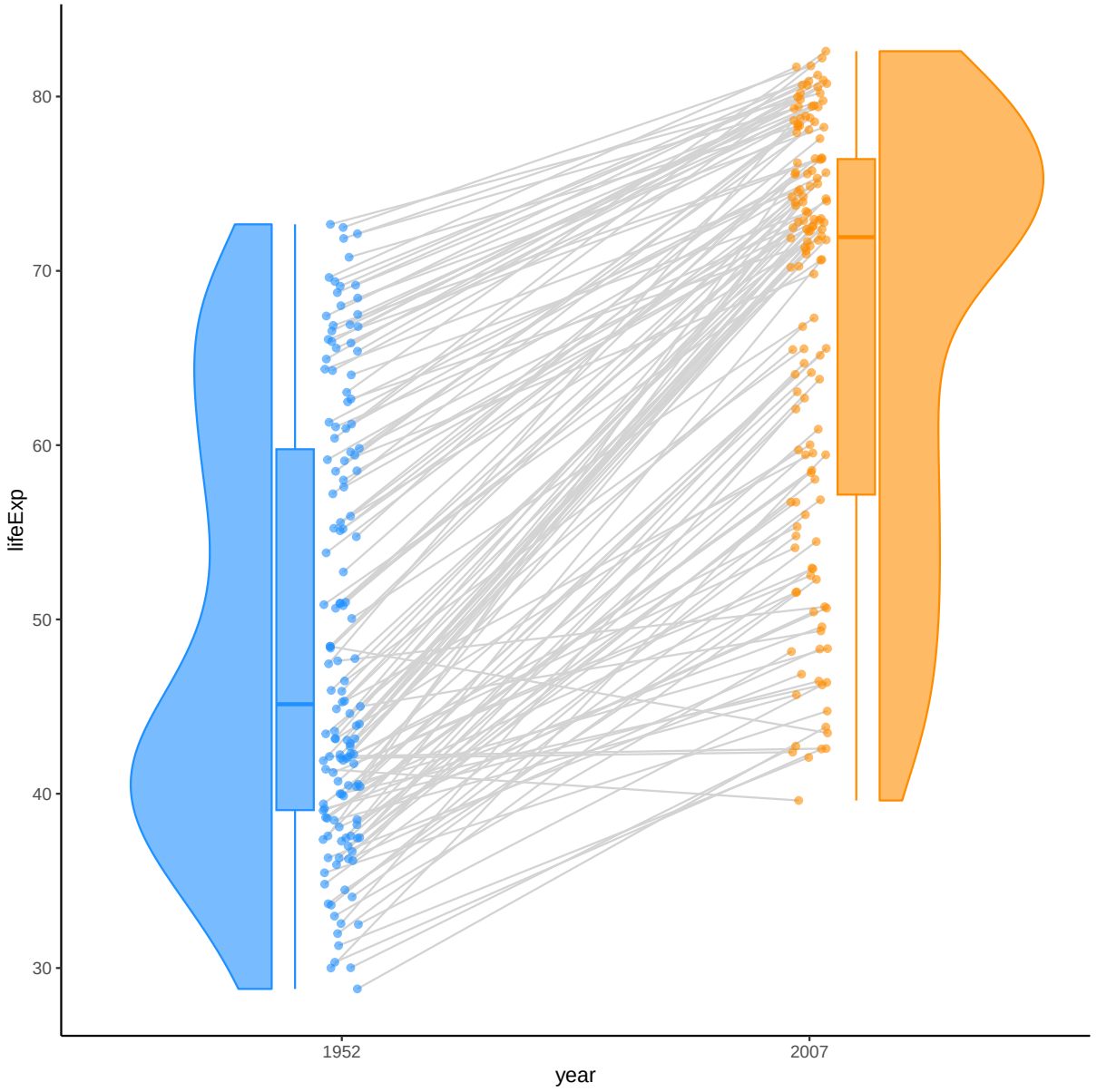
Se pueden mostrar las distribuciones para todas las variables numéricas de una base de datos.

```
gapminder |>
  select(where(is.numeric)) |>
  pivot_longer(everything()) |>
  ggplot(aes(name, value, fill = name)) +
  ggrain::geom_rain(alpha = .5) +
  theme_minimal(base_size = 10) +
  guides(fill = 'none', color = 'none') +
  facet_wrap(~ name, scales = "free") +
  theme(axis.text.x = element_blank())
```



O la relación entre dos condiciones, momentos, etc.

```
gapminder_min_max = gapminder |> filter(year == max(year) | year == min(year)) |> mutate(yea

gapminder_min_max |>
  ggplot(aes(year, lifeExp, fill = year, color = year)) +
  geom_rain(alpha = .6, rain.side = 'f1x1', id.long.var = "country", line.args = list(color
  theme_classic() +
  scale_fill_manual(values=c("dodgerblue", "darkorange")) +
  scale_color_manual(values=c("dodgerblue", "darkorange")) +
  guides(fill = 'none', color = 'none')
```

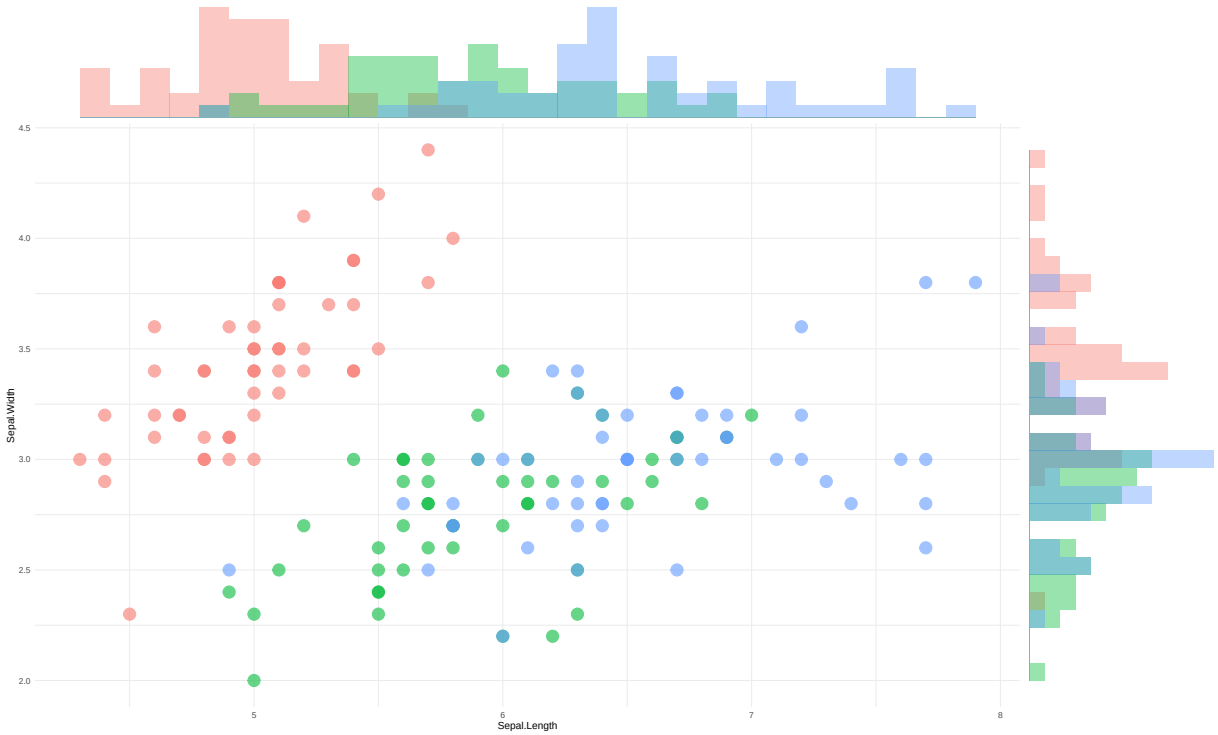


3.4.1.1 Combinando múltiples gráficas

Podemos combinar múltiples gráficas y llegar a hacer cosas mucho más complejas como combinar un scatteplot con un par de histogramas:

```
# Set up scatterplot
scatterplot <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3, alpha = 0.6) +
  guides(color = "none") +
  theme(plot.margin = margin())


# Define marginal histogram
```

```r
marginal_distribution <- function(x, var, group) {
  ggplot(x, aes(x = get(var), fill = get(group))) +
    geom_histogram(bins = 30, alpha = 0.4, position = "identity") +
    # geom_density(alpha = 0.4, size = 0.1) +
    guides(fill = "none") +
    theme_void() +
    theme(plot.margin = margin())
}

# Set up marginal histograms
x_hist <- marginal_distribution(iris, "Sepal.Length", "Species")
y_hist <- marginal_distribution(iris, "Sepal.Width", "Species") +
  coord_flip()

# Align histograms with scatterplot
aligned_x_hist <- align_plots(x_hist, scatterplot, align = "v")[[1]]
aligned_y_hist <- align_plots(y_hist, scatterplot, align = "h")[[1]]


# Arrange plots
cowplot::plot_grid(
  aligned_x_hist, NULL, scatterplot, aligned_y_hist,
  ncol = 2, nrow = 2,
  rel_heights = c(0.2, 1), rel_widths = c(1, 0.2)
  )
```



3.4.2 Visualización interactiva

El paquete {plotly} nos permite crear gráficas con algunos niveles de interactividad usando funciones propias, o modificando gráficas creadas con ggplot.

3.4.2.1 ggplots interactivos con plotly

Scatterplot creado con ggplot donde se puede ver el valor de los puntos, seleccionar áreas, etc.

```
plotly::ggplotly(
  ggplot(
    gapminder |> filter(year == 2007),
    aes(gdpPercap, lifeExp, color = continent, size = country)
  ) +
    geom_point(alpha = .3, point = 2) +
    scale_y_continuous(breaks = seq(0, 100, 5)) +
    scale_x_log10(labels = scales::dollar_format(prefix = "$", suffix = "M")) +
    theme(legend.position = "none")
)
```

80
75

3.4.3 Surface plots con plotly

Surface plot creado con plotly donde se muestra la relación entre 3 variables en un entorno interactivo 3D.

```
DF_RAW = structure(c(181, 163, 60, 124, 76, 62, 73, 59, 17, 21, 26, 7, 1, 2, 3,
                     188, 145, 61, 130, 61, 59, 62, 57, 20, 22, 22, 6, 4, 5, 5,
                     137, 154, 54, 191, 75, 56, 65, 56, 22, 27, 33, 14, 5, 5, 5,
                     126, 185, 65, 109, 51, 71, 57, 38, 25, 23, 21, 10, 5, 5, 5,
                     150, 144, 44, 123, 58, 24, 48, 41, 19, 26, 21, 5, 5, 5, 5,
                     138, 137, 61, 130, 67, 34, 60, 44, 19, 21, 16, 4, 5, 5, 5,
                     121, 146, 101, 92, 70, 74, 88, 33, 18, 39, 24, 12, 5, 5, 5,
                     100, 160, 129, 117, 70, 61, 42, 35, 22, 25, 21, 7, 10, 23, 8,
                     100, 129, 130, 107, 64, 61, 44, 25, 23, 30, 18, 11, 20, 58, 40,
                     100, 136, 131, 96, 53, 31, 51, 37, 43, 31, 19, 2, 22, 40, 41,
                     100, 124, 154, 74, 62, 44, 34, 15, 26, 23, 20, 6, 23, 10, 19,
                     100, 126, 251, 76, 73, 84, 47, 40, 32, 25, 32, 6, 13, 10, 13,
                     100, 129, 194, 91, 53, 99, 46, 34, 60, 21, 17, 6, 14, 14, 26,
                     100, 115, 119, 88, 64, 108, 37, 24, 49, 26, 17, 6, 15, 15, 47),
                   .Dim = 15:14,
                   .Dimnames = list(c("1", "2", "3", "4", "5", "6", "7", "8", "9",
                                      "10", "11", "12", "13", "14", "15"),
                                    c("2006", "2007", "2008", "2009", "2010",
                                      "2011", "2012", "2013", "2014", "2015",
                                      "2016", "2017", "2018", "2019")))

DF = DF_RAW
plot1 = plotly::plot_ly(x = ~ colnames(DF),
                        y = ~ rownames(DF),
                        z = ~ DF) |>
  plotly::add_surface(
    name = "3D mesh",
    connectgaps = TRUE,
    hidesurface = TRUE,
    showscale = FALSE,
    contours = list(
      x = list(
        show = TRUE,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
        highlight = TRUE
      ),
      y = list(
        show = TRUE,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
```

```
        highlight = TRUE
      ),
      z = list(
        show = FALSE,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
        highlight = FALSE
      )
    )
  ) |>
  plotly::add_surface(
    name = "surface",
    connectgaps = FALSE,
    contours = list(
      x = list(
        show = F,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
        highlight = TRUE
      ),
      y = list(
        show = F,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
        highlight = TRUE
      ),
      z = list(
        show = FALSE,
        width = 1,
        highlightwidth = 2,
        highlightcolor = "#41a7b3",
        highlight = FALSE
      )
    )
  )

if (!knitr::is_latex_output()) plot1
```

3.4.4 Animando gráficas con gganimate

{gganimate} nos permite crear ggplots añadiendo la dimensión temporal

```
if (!require('gganimate')) remotes::install_github('thomasp85/gganimate'); library('gganimat
#sudo apt-get install ffmpeg
```

```
p = ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, colour = country)) +
  geom_point(alpha = 0.7, show.legend = FALSE) +
  scale_colour_manual(values = country_colors) +
  scale_size(range = c(2, 12)) +
  scale_x_log10() +
  facet_wrap(~continent) +

  # Here comes the gganimate specific bits
  labs(title = 'Year: {frame_time}', x = 'GDP per capita', y = 'life expectancy') +
  transition_time(year) +
  ease_aes('linear')

  # Create animated plot
  animate(p, renderer = ffmpeg_renderer(),
          height = 6, width = 10, units = "in", res = 300)

  # Save plot
    # anim_save("name_file.mp4", animation = last_animation())
```

gganimate animation

Bibliografía

- Matejka, J., & Fitzmaurice, G. (2017, May). Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (pp. 1290-1294). ACM.
- https://bbc.github.io/rcookbook/
- https://github.com/bbc/bbplot
- https://github.com/dreamRs/esquisse
- Garrick Aden-Buie. A Gentle Guide to the Grammar of Graphics with ggplot2: https://github.com/gadenbuie/gentle-ggplot2
- Michael Toth. You Need to Start Branding Your Graphs. Here's How, with ggplot!: https://michaeltoth.me/you-need-to-start-branding-your-graphs-heres-how-with-ggplot.html
- Claus Wilke: https://wilkelab.org/practicalgg/
- Thomas Lin Pedersen:
  - Part 1: https://www.youtube.com/watch?v=h29g21z0a68
  - Part 2: https://www.youtube.com/watch?v=0m4yywqNPVY
- Big Book or R : https://www.bigbookofr.com/index.html

4 Preparación y transformación de datos

En este capítulo vamos a aprender a importar y exportar todo tipo de archivos. También veremos como, filtrar, crear nuevas variables, seleccionar columnas, transformar bases de datos, etc.

Paquetes para este capítulo

Como en cada capítulo, crea un nuevo script de R (CNTRL+ SHIFT + N), guárdalo con el nombre/número del capítulo (i.e. `capitulo4.R`), copia y pega las líneas de abajo y ejecútalas.

```
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require("DT")) install.packages("DT"); library("DT")
if (!require("ggplot2")) install.packages("ggplot2"); library("ggplot2")
if (!require("googlesheets4")) install.packages("googlesheets4"); library("googlesheets4")
if (!require("haven")) install.packages("haven"); library("haven")
if (!require("here")) install.packages("here"); library("here")
if (!require("janitor")) install.packages("janitor"); library("janitor")
if (!require("purrr")) install.packages("purrr"); library("purrr")
if (!require('readr')) install.packages('readr'); library('readr')
if (!require("readxl")) install.packages("readxl"); library("readxl")
if (!require("readODS")) install.packages("readODS"); library("readODS")
if (!require("tidyr")) install.packages("tidyr"); library("tidyr")
if (!require("waldo")) install.packages("waldo"); library("waldo")
if (!require("writexl")) install.packages("writexl"); library("writexl")

if (!require('regexplain')) remotes::install_github("gadenbuie/regexplain"); library('regexp
```

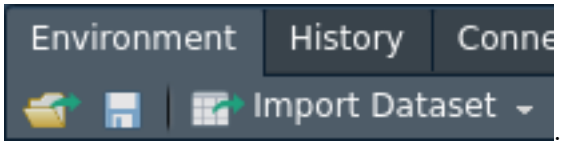
Proyecto de RStudio del curso

En como empezar se mostró como tener un proyecto de RStudio con todos los materiales del curso. Si no lo tienes, este es el momento de hacerlo, ya que vamos a necesitar varios de los archivos del proyecto:

```
if (!require('usethis')) install.packages('usethis'); library('usethis')
usethis::use_course("gorkang/R_preparacion_visualizacion_datos")
```

4.1 Importar y exportar datos

Hasta ahora hemos trabajado con data frames como `mpg` o `gaminder`, que forman parte de la instalación de alguno de los paquetes de R. Pero habitualmente trabajaremos con datos propios, por lo que necesitaremos leer uno o varios archivos.

RStudio tiene un menú para ayudar a importar datos 

. No obstante, aquí aprenderemos a hacerlo todo en código, para que nuestros scripts sean autocontenidos.

Podemos ver algunas de las funciones de esta sección y cómo usarlas en la Cheatsheet importar datos

4.1.1 Importar un solo archivo

Empezaremos por la situación básica más común, cómo importar un solo archivo. Vamos a ver con más detalle los archivos CSV (*comma separated values*). Las funciones para importar archivos Excel, Libreoffice, SPSS, etc. tienen parámetros similares.

4.1.1.1 Archivos CSV

Usaremos las siguientes funciones del paquete `readr`:

> **Funciones para leer archivos csv**
>
> - `readr::read_csv()`: valores separados por coma (“,”)
> - `readr::read_csv2()`: valores separados por punto y coma (“;”), típicamente usado en países de habla Hispana
> - `readr::read_delim( , delim = "|")`: valores separados por un delimitador arbitrario

Leemos el archivo `02-read-csv.csv` de la carpeta `data/files/`:

```
DF_name = read_csv("data/files/02-read-csv.csv")
```

Generalmente haremos esto en dos pasos. Primero guardaremos la ubicación del archivo en una variable, y después usaremos esa variable como parámetro de la función que lee los datos:

```
# Guardamos en la variable `name_of_file` la ubicación del archivo
name_of_file = "data/files/02-read-csv.csv"

# Usamos la función read_csv() para leer el archivo
DF_name = read_csv(name_of_file)
```

En este manual verás que usamos `name_of_file = here::here("mi/archivo.csv")`. La función `here::here` nos ayuda a evitar problemas con la ruta a los archivos (nos devuelve la ruta absoluta). Esto suele ser importante cuando trabajamos con quarto or rmarkdown, pero normalmente lo podemos ignorar.

```
# En name_of_file almacenamos la ruta absoluta al archivo
name_of_file = here::here("data/files/02-read-csv.csv")

# Leemos el archivo que esta en name_of_file
DF_name = read_csv(name_of_file)

DF_name
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Si usamos un repositorio online para almacenar los archivos, podemos leer directamente de una URL.

```
URL = "https://raw.githubusercontent.com/gorkang/R_preparacion_visualizacion_datos/master/da
read_csv(URL)
```

La función `read_csv()` tiene varios parámetros muy útiles como `skip` o `col_types` (si el cursor está encima de la función, F1 os llevará a la ayuda).

- Con `skip` podemos saltarnos líneas del inicio del archivo. Muy útil cuando las hojas de datos no empiezan en la primera fila.
- Con `col_types` podemos especificar el tipo de datos que contiene cada columna (texto, números, factores…). Al ser explícitos con el tipo de datos evitamos sorpresas, y de paso reducimos el output de la Consola.

```
# En name_of_file almacenamos la ruta completa al archivo
name_of_file = here::here("data/files/02-read-csv.csv")

# Leemos el archivo que esta en name_of_file
read_csv(name_of_file,
         skip = 0,
         col_types = cols(
           .default = col_double(),
```

```
          condition = col_character(),
          condition2 = col_character()
          )
        )
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

4.1.1.2 Otros tipos de archivos

Para otros tipos de archivos usaremos otras funciones.

Archivos excel

Para leer archivos Excel podemos usar `read_excel()` del paquete `{readxl}`. Cada una de estas funciones tiene parámetros que nos pueden resultar muy útiles. Por ejemplo, `read_excel()` tiene el parámetro `sheet`, que nos permite seleccionar qué hoja del archivo excel leer.

```
name_of_file = here::here("data/files/02-read-xlsx.xlsx")
readxl::read_excel(name_of_file, sheet = 1)
```

Archivos SPSS

Para archivos .sav usamos `haven::read_sav()`.

```
name_of_file = here::here("data/files/02-read-sav.sav")
haven::read_sav(name_of_file)
```

Archivos Libreoffice

Para archivos .ods usamos `readODS::read_ods()`.

```
name_of_file = here::here("data/files/02-read-ods.ods")
df_ODS = readODS::read_ods(name_of_file)

# Vemos las primeras filas
head(df_ODS)
```

Google sheets

Para poder leer una gsheet debemos:

1) Crear un enlace para compartirla: `"Share" -> "General access" -> "Anyone with the link"`

2) Extraemos el identificador de la google sheet:

- De `https://docs.google.com/spreadsheets/d/1v3cCKaQ3akGEH8Z2Plu7qpq93GNk6Y6v337iI4KyZ2I/e`
- Usaremos: `1v3cCKaQ3akGEH8Z2Plu7qpq93GNk6Y6v337iI4KyZ2I`

```
if (!require("googlesheets4")) install.packages("googlesheets4"); library("googlesheets4")
name_of_sheet = "1v3cCKaQ3akGEH8Z2Plu7qpq93GNk6Y6v337iI4KyZ2I"
googlesheets4::read_sheet(name_of_sheet)
```

Ejercicios - Importar datos

En el repositorio R para preparación y visualización de datos - DNSC - UAI de la Open Science Foundation podrás ver una carpeta llamada `OSF_files`. Si no tenéis conexión a Internet, podéis encontrar los archivos en `data/files/OSF_files`.

Descarga e importa los archivos que ahí aparecen, asegurándote que los nombres de columna se leen adecuadamente:

- 02-extralines-1.xlsx
- 02-extralines-2.xlsx
- 02-extralines-3.xlsx
- 02-spanish.csv

Solución

La función `read_excel()` tiene parámetros como `skip`, que permite no leer las primeras n lineas, o `sheet`, con la que puedes indicar que pestaña leer.
Recuerda que estás leyendo archivos excel o csv, por lo que puedes verlos en Libreoffice o Excel para saber como son.

4.1.2 Importar múltiples archivos

En ocasiones tenemos múltiples archivos en una carpeta (e.g. uno por participante) y queremos combinarlos todos en un solo data frame.

Por suerte, las funciones como `read_csv()` admiten un vector con varios archivos.

Para importar todos los archivos que están en la carpeta `data/files/02-CSVs`:

```r
# Directorio donde se encuentran los archivos
name_of_folder = here::here("data/files/02-CSVs")

# Listamos los archivos a leer
files <- list.files(name_of_folder, full.names = TRUE)

# Leemos todos los archivos, combinándolos en un data frame
full <- read_csv(files)

full
#> # A tibble: 1,600 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <chr> <chr>       <dbl> <chr> <chr>      <chr>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 1,594 more rows
```

Lamentablemente, cuando leamos otro tipo de archivos (e.g. `.xlsx`), no podemos usar la función `read_csv()`. Veamos una manera de hacer leer múltiples archivos que se puede usar con cualquier función para leer datos.

Usaremos `map_df()` del paquete `{purrr}`, que aplica la función que queramos a cada uno de los archivos que le indiquemos, uno a uno:

```r
# Directorio donde se encuentran los archivos
name_of_folder = here::here("data/files/02-CSVs")

# Listamos los archivos a leer
files <- list.files(name_of_folder, full.names = TRUE)

# Leemos todos los archivos de uno en uno, combinándolos en un data frame
full <- purrr::map_df(files, read_csv)

# Mostramos lo que contiene full
full
#> # A tibble: 1,600 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <chr> <chr>       <dbl> <chr> <chr>      <chr>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
```

```
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 1,594 more rows
```

4.1.2.1 Incluir nombres de archivos

Habitualmente será importante saber a que archivo pertenecen los datos que hemos leído.

Podemos incluir los nombres de archivo en una columna:

```
# Nombre de la carpeta
name_of_folder = here::here("data/files/02-CSVs")

# Listamos los archivos dentro de esa carpeta
files_simple <- list.files(name_of_folder, full.names = TRUE)

# Asignamos nombres a los elementos del vector
files = set_names(files_simple, basename(files_simple))

# Con el parámetro .id, almacenamos los nombres en la columna "file"
full2 <- map_df(files, read_csv, .id = "file")
```

4.1.2.2 Con parametros

Añadimos parámetros a la función de lectura. En este caso, definimos el tipo de columna esperado con la función `col_types()`. Con esto nos aseguraremos que si alguno de los archivos tiene el tipo de datos "incorrecto", aparecerán warnings en la importación:

```
name_of_folder = here::here("data/files/02-CSVs")
files <- list.files(name_of_folder, full.names = TRUE)
full <- map_df(files, read_csv,
               # Ponemos los parámetros separados por comas, después de la función
               col_types = cols(
                 .default = col_character(),
                 Sex = col_factor(),
                 trialN = col_integer(),
                 Valence = col_factor(),
                 rT = col_double()
                 )
               )

full
#> # A tibble: 1,600 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <fct> <chr>       <int> <chr> <chr>      <fct>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
```

```
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 1,594 more rows
```

Otra manera de hacer exactamente lo mismo con `map_df()`. El código es algo más complejo, pero nos da más flexibilidad, y podemos usar las funciones del mismo modo que habitualmente.

```
name_of_folder = here::here("data/files/02-CSVs")
files <- list.files(name_of_folder, full.names = TRUE)
full2 <- purrr::map_df(1:length(files), ~ {
  cli::cli_alert_info("Reading file {basename(files[.x])}")
  read_csv(files[.x], col_types =  cols(.default = col_character(),
                                        Sex = col_factor(),
                                        trialN = col_integer(),
                                        Valence = col_factor(),
                                        rT = col_double()
                                        )
           )
  })


# Comprobamos que no hay diferencias
waldo::compare(full, full2)
#> v No differences
```

Ejercicios - Importar múltiples archivos

1. Cuando más arriba importamos los archivos que están en la carpeta `data/files/02-CSVs`:

- ¿Qué archivos importamos exactamente?

> ¿Ves algún problema en lo que hicimos?
>
> Revisa el número de filas y el contenido de la variable `files`.

El resultado final debería ser así:

```
#> # A tibble: 1,200 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <chr> <chr>       <dbl> <chr> <chr>      <chr>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
```

```
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 1,194 more rows
```

2. Leed los archivos .xlsx de la carpeta `data/files/02-XLSs`, combinándolos en un único DF. El resultado final debería ser como se ve a continuación:

> **Pista**
>
> Tendréis que usar `list.files()` usando el parámetro `pattern`
>
> - Te recomiendo abrir los archivos excel para ver su estructura, las pestañas que tienen... ahí te darás cuenta de que necesitas otros parámetros de `read_xlsx()` como `sheet` o `skip`

```
#> # A tibble: 1,200 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <chr> <chr>       <dbl> <chr> <chr>      <chr>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 1,194 more rows
```

4.1.3 Limpiar nombres de columnas

El paquete {janitor} tiene una función muy útil llamada `clean_names()` que aplica algunas reglas sencillas para estandarizar los nombres de columnas:

```
DF_name = read_csv(here::here("data/files/02-read-csv.csv"))
names(DF_name)
#> [1] "...1"         "ID"           "Genero"       "Edad"
#> [5] "Educacion"    "FollowUP"     "condition"    "condition2"
#> [9] "PPV_DECLARED"

DF_names_clean = janitor::clean_names(DF_name)
names(DF_names_clean)
#> [1] "x1"           "id"           "genero"       "edad"
#> [5] "educacion"    "follow_up"    "condition"    "condition2"
#> [9] "ppv_declared"
```

4.1.4 Exportar datos

Muchas veces guardaremos los datos una vez procesados. Esto se puede hacer con la familia de funciones `write_*`.

Siguiendo el ejercicio anterior, después de leer los archivos excel de una carpeta y limpiar los nombres de columna, queremos guardar el data frame resultante en la carpeta `data_clean` (¡es importante asegurarnos que esa carpeta existe!).

4.1.4.1 Archivos CSV

Si queremos que el archivo guardado sea un `csv`, usaremos `write_csv()` del paquete {`readr`}.

```
# Lo que hicimos antes, leemos todos los archivos excel de un directorio
name_of_folder = here::here("data/files/02-XLSs")
files <- list.files(name_of_folder, pattern = "xls", full.names = TRUE)
DF_all = janitor::clean_names(map_df(files, read_xlsx, sheet = 2, skip = 5))

readr::write_csv(DF_all, "data_clean/DF_all.csv")
```

4.1.4.2 Otros Archivos

La función a usar cambiará en función del formato que queramos:

- `xlsx`: `writexl::write_xlsx()`
- `sav`: `haven::write_sav()`
- `ods`: `readODS::write_ods()`
- `rds`: `readr::write_rds()`

```
writexl::write_xlsx(DF_all, "data_clean/DF_all.xlsx")

haven::write_sav(DF_all, "data_clean/DF_all.sav")

readODS::write_ods(DF_all, "data_clean/DF_all.ods")

readr::write_rds(DF_all, "data_clean/DF_all.rds")
```

4.2 Preparación y transformación de datos

Para la preparación y transformación de datos usaremos fundamentalmente `dplyr`. Hay otros paquetes más rápidos como `data.table`. Si trabajas con datos gigantescos (millones de filas), sin duda notarás la diferencia. La desventaja es que la sintaxis es (habitualmente) menos intuitiva.

4.2.1 The pipe

Con ggplot enlazábamos instrucciones (capas) usando +. Cuando trabajemos con datos, usaremos "tuberias" (pipes) |>.

Las pipes |> o %>% (CONTROL + SHIFT + M) nos permiten enlazar operaciones de transformación de datos de manera sencilla. Simplemente encadenamos instrucciones, poniendo una pipe para que cada instrucción llegue a la siguiente linea.

Imaginad que queremos realizar una serie de operaciones sobre unos datos, y guardarlos en un data frame llamado DATOS_limpios. Partimos de unos DATOS_originales, seleccionamos las columnas que nos interesan y filtramos los datos.

De manera esquemática:

```
DATOS_limpios =
  DATOS_originales |>
  seleccionamos_columnas(parámetros) |>
  filtramos_datos(parámetros)
```

4.2.2 Tidy data

Existen tres sencillas reglas que definen la *Tidy data*:

1. Cada variable tiene su columna propia
2. Cada observación tiene su fila propia
3. Cada valor tiene su celda propia

Las ventajas fundamentales son:

- Uso de una manera consistente de trabajar, que se alinea con el tidyverse
- Facilidad para trabajar con la lógica vectorizada

Por ejemplo. De manera muy sencilla y rápida podemos crear una nueva columna realizando algún cómputo arbitrario con los valores de otra columna.

```
# Computa ratio por 100,000
table1 |>
  mutate(rate_per_100K = cases / population * 100000)
#> # A tibble: 6 x 5
#>   country      year  cases population rate_per_100K
#>   <chr>       <dbl>  <dbl>      <dbl>         <dbl>
#> 1 Afghanistan  1999    745   19987071          3.73
#> 2 Afghanistan  2000   2666   20595360         12.9
#> 3 Brazil       1999  37737  172006362         21.9
#> 4 Brazil       2000  80488  174504898         46.1
```

```
#> 5 China        1999 212258 1272915272         16.7
#> 6 China        2000 213766 1280428583         16.7
```

O contar el número de casos por valor de una variable.

```
# Cuenta cuantos registros tenemos por año
table1 |>
  count(year)
#> # A tibble: 2 x 2
#>    year     n
#>   <dbl> <int>
#> 1  1999     3
#> 2  2000     3

# Cuenta cuantos registros tenemos por año y país
table1 |>
  count(year, country)
#> # A tibble: 6 x 3
#>    year country         n
#>   <dbl> <chr>       <int>
#> 1  1999 Afghanistan     1
#> 2  1999 Brazil          1
#> 3  1999 China           1
#> 4  2000 Afghanistan     1
#> 5  2000 Brazil          1
#> 6  2000 China           1
```

También podemos hacer cosas algo más complejas, por ejemplo, aplicar funciones como `across` o `where(is.numeric)` para sumar todas las columnas de tipo numérico.

```
table1 |>
  mutate(suma_absurda =  rowSums(across(where(is.numeric))))
#> # A tibble: 6 x 5
#>   country      year  cases population suma_absurda
#>   <chr>       <dbl>  <dbl>      <dbl>        <dbl>
#> 1 Afghanistan  1999    745   19987071     19989815
#> 2 Afghanistan  2000   2666   20595360     20600026
#> 3 Brazil       1999  37737  172006362    172046098
#> 4 Brazil       2000  80488  174504898    174587386
#> 5 China        1999 212258 1272915272   1273129529
#> 6 China        2000 213766 1280428583   1280644349
```

Y, como no, `ggplot` funciona con datos `tidy`, en formato long.

4.2.3 Verbos dplyr

Usaremos {dplyr}, un paquete muy potente para la manipulación de datos. Su sintaxis, además, es bastante intuitiva (¡son verbos en inglés!).

Podemos ver más detalle y ejemplos en la Cheatsheet de dplyr.

Verbos esenciales

- filter(): filtrar filas
- arrange(): ordenar filas
- select(): seleccionar columnas
- rename(): renombrar columnas
- mutate(): crear columnas, modificar columnas, etc.

Tabla resumen dplyr

	tarea	funcion	ejemplo
1	Filtrar	filter()	datos |> filter(Sexo == 1)
2	Ordenar	arrange()	datos |> arrange(Sexo)
3	Seleccionar/eliminar variables	select()	datos |> select(-Sexo)
4	Renombrar variables	rename()	datos |> rename(Genero = Sexo)
5	Separar contenidos de variable	separate()	datos |> separate(var_name, c('First', 'Second'), sep = '_')
6	Extraer valores únicos	distinct()	datos |> distinct(Edad, .keep_all = T)
7	Crear/modificar variables	mutate()	datos |> mutate(Viejo = Edad > 30)
8	Omitir NAs	drop_na()	datos |> drop_na(Sexo)
9	Wide to long	pivot_longer()	datos |> pivot_longer(4:6, names_to = 'Condition', values_to = 'VD')
10	Long to wide	pivot_wider()	datos |> pivot_wider(names_from = Condition, values_from = VD)
11	Combinar bases de datos	left_join()	left_join(datos1, datos2, by = 'ID')
12	Recodificar valores	ifelse()	datos |> mutate(Edad = ifelse(Edad > 30, 'Viejo', 'Joven'))
13	Recodificar valores	case_when()	datos |> mutate(Edad = case_when(Edad > 30 ~ 'Viejo', 'Joven')

4.2.3.1 Filtrar y ordenar filas

A partir de los siguientes datos:

```
name_of_file = here::here("data/files/02-read-csv.csv")
DF_name = read_csv(name_of_file)
DF_name
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Podemos usar el verbo `filter()` para mostrar las filas que cumplan cualquier regla lógica o combinación de reglas (e.g. Genero == 1 & Edad >= 30).

En el ejemplo siguiente, nos quedamos con las filas donde el valor de Educación sea mayor de 8:

```
DF_name |>
  filter(Educacion > 8)
#> # A tibble: 3 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1   157 12207      1    26         9       57 PPV_Cond2 100RInt
#> 2   287 60873      1    72        10       51 PPV_Cond3 100RAmp
#> 3   381 64486      2    19         9       80 PPV_Cond4 100RAmp
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Si queremos usar varias condiciones lógicas, las reparamos de `&` (AND).

Aquí nos quedamos con las filas donde el valor de Educación sea mayor de 8 **Y** el Genero sea igual a 1 (fíjate que usamos `==` para indicar la condición lógica de igualdad):

```
DF_name |>
  filter(Educacion > 8 & Genero == 1)
#> # A tibble: 2 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1   157 12207      1    26         9       57 PPV_Cond2 100RInt
#> 2   287 60873      1    72        10       51 PPV_Cond3 100RAmp
#> # i 1 more variable: PPV_DECLARED <dbl>
```

También podemos ordenar las filas usando `arrange()` solo o en combinación con `desc()`.

Por ejemplo, ordenamos a partir de Educación, de menor a mayor:

```
DF_name |>
  arrange(Educacion, desc(Genero))
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1   350 20439      2    41         1       81 PPV_Cond4 90LAmp
#> 2   399 81379      1    36         1       90 PPV_Cond4 90RAmp
#> 3    42 20361      2    37         2       60 PPV_Cond1 100LInt
#> 4   364 19201      2    21         2       67 PPV_Cond4 90LAmp
#> 5   412 60292      1    28         2       90 PPV_Cond4 100LAmp
#> 6    44 92735      2    30         3       95 PPV_Cond1 100RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Si queremos ordenar a partir de dos o más criterios, simplemente los separamos por coma. Además, si queremos que alguno de ellos sea de manera descendente (por defecto es ascendente), envolvemos el nombre de columna en `desc()`.

Abajo, ordenamos educación de manera ascendente, y de Género descendente:

```
DF_name |>
  arrange(Educacion, desc(Genero))
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1   350 20439      2    41         1       81 PPV_Cond4 90LAmp
#> 2   399 81379      1    36         1       90 PPV_Cond4 90RAmp
#> 3    42 20361      2    37         2       60 PPV_Cond1 100LInt
#> 4   364 19201      2    21         2       67 PPV_Cond4 90LAmp
#> 5   412 60292      1    28         2       90 PPV_Cond4 100LAmp
#> 6    44 92735      2    30         3       95 PPV_Cond1 100RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

4.2.3.2 Seleccionar, ordenar y renombrar columnas

Seleccionamos las columnas que queremos listando las variables en el orden deseado dentro del verbo `select()`:

```
DF_name |>
  select(Genero, Edad)
#> # A tibble: 103 x 2
#>   Genero  Edad
```

```
#>    <dbl> <dbl>
#> 1      1    47
#> 2      2    21
#> 3      2    29
#> 4      2    27
#> 5      1    29
#> 6      2    28
#> # i 97 more rows
```

Eliminamos columnas precediendo las variables a eliminar con `-` dentro de `select()`:

```
DF_name |>
  select(-...1)
#> # A tibble: 103 x 8
#>      ID Genero  Edad Educacion FollowUP condition condition2 PPV_DECLARED
#>   <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>             <dbl>
#> 1 41904      1    47         8       80 PPV_Cond1 90LInt               99
#> 2 95041      2    21         6       90 PPV_Cond1 90RInt               99
#> 3 74594      2    29         6       10 PPV_Cond1 100LInt              99
#> 4 72903      2    27         7       75 PPV_Cond1 100RInt               1
#> 5 21260      1    29         5       35 PPV_Cond1 90LInt               24
#> 6 50315      2    28         6       14 PPV_Cond1 90RInt               NA
#> # i 97 more rows
```

Ordenamos y eliminamos columnas listando las variables en el orden deseado dentro del verbo `select()` y precediendo las variables a eliminar con `-`:

```
DF_name |>
  select(ID, Edad, Genero, everything(), -...1)
#> # A tibble: 103 x 8
#>      ID  Edad Genero Educacion FollowUP condition condition2 PPV_DECLARED
#>   <dbl> <dbl>  <dbl>     <dbl>    <dbl> <chr>     <chr>             <dbl>
#> 1 41904    47      1         8       80 PPV_Cond1 90LInt               99
#> 2 95041    21      2         6       90 PPV_Cond1 90RInt               99
#> 3 74594    29      2         6       10 PPV_Cond1 100LInt              99
#> 4 72903    27      2         7       75 PPV_Cond1 100RInt               1
#> 5 21260    29      1         5       35 PPV_Cond1 90LInt               24
#> 6 50315    28      2         6       14 PPV_Cond1 90RInt               NA
#> # i 97 more rows
```

Para renombrar variables usamos el verbo `rename()`. Es importante recordar que en hay que indicar `rename(NOMBRE_NUEVO = NOMBRE_ANTIGUO)`:

```
DF_name |>
  rename(Identificador = ID,
         Sexo = Genero)
```

```
#> # A tibble: 103 x 9
#>    ...1 Identificador  Sexo  Edad Educacion FollowUP condition condition2
#>   <dbl>         <dbl> <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4         41904     1    47         8       80 PPV_Cond1 90LInt
#> 2     5         95041     2    21         6       90 PPV_Cond1 90RInt
#> 3     6         74594     2    29         6       10 PPV_Cond1 100LInt
#> 4    15         72903     2    27         7       75 PPV_Cond1 100RInt
#> 5    16         21260     1    29         5       35 PPV_Cond1 90LInt
#> 6    18         50315     2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Ejercicios - verbos dplyr simples

- Cuenta los registros por año en el data frame `mpg`

- Filtra los datos para quedarnos solo con los del año 1999
- Renombra la variable displ para que se llame "engine displacement"
  - Si aparece el error `Error: unexpected symbol in ...`, puedes ver la ayuda de la función ?make.names, o este post
- Ordena los datos (no las columnas) por consumo en ciudad `cty` y clase de vehículo `class`
- Crea un data frame que no contenga la variable `model`

Soluciones

```
- mpg |> count(year)
- mpg |> filter(year == 1999)
- mpg |> rename(engine displacement = displ) #ERROR
- mpg |> rename(engine_displacement = displ) #SOLUCION1
- mpg |> rename(`engine displacement` = displ) #SOLUCION2
- mpg |> arrange(cty, class)
- mpg |> select(-model)
```

4.2.3.3 Selección avanzada con select_helpers()

El `everything()` que usamos dentro de `select()` más arriba es uno de los `select_helpers()` existentes. Estos nos ayudan a realizar operaciones de selección de variables sin necesidad de escribir a mano todas las variables.

select_helpers()

- starts_with(): Empieza con un prefijo (e.g. starts_with("CI_"))
- ends_with(): Acaba con un sufijo

- contains(): Contiene una cadena de texto específica
- matches(): Matches a regular expression
- num_range(): Matches a numerical range like x01, x02, x03
- one_of(): Matches variable names in a character vector
- everything(): Matches all variables
- last_col(): Select last variable

Trabajaremos con los datos del paper Cognitive and Socio-affective Predictors of Social Adaptation, de Neely et al. Estos se pueden encontrar en un repositorio público de la OSF. Empezaremos con la base RAW en formato wide. Leemos la base y mostramos los títulos de las 291 columnas:

```
  df_wide = read_csv("https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affectiv
  cat(names(df_wide))
#> ID dem_genero dem_edad dem_nivedu WVOC_01_cod WVOC_02_cod WVOC_03_cod WVOC_04_cod WVOC_05
```

Con `contains()` seleccionamos variables que contienen la cadena de texto `dem`:

```
  df_wide |>
    select(contains("dem"))
#> # A tibble: 232 x 3
#>   dem_genero dem_edad dem_nivedu
#>        <dbl>    <dbl>      <dbl>
#> 1          1       38          4
#> 2          0       67          2
#> 3          0       24          4
#> 4          0       30          4
#> 5          0       38          3
#> 6          0       45          4
#> # i 226 more rows
```

Usando `ends_with()` seleccionamos variables que acaban con la cadena de texto `cod`:

```
  df_wide |>
    select(ID, ends_with("cod"))
#> # A tibble: 232 x 79
#>      ID WVOC_01_cod WVOC_02_cod WVOC_03_cod WVOC_04_cod WVOC_05_cod
#>   <dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
#> 1     1           2           2           2           1           2
#> 2     2           2           2           2           1           0
#> 3     3           2           2           2           1           2
#> 4     4           2           1           1           1           2
```

```
#> 5     5           2           2           1           1           0
#> 6     6           1           1           2           1           2
#> # i 226 more rows
#> # i 73 more variables: WVOC_06_cod <dbl>, WVOC_07_cod <dbl>, ...
```

Finalmente, `matches()` nos permite usar todo la potencia de las expresiones regulares. En este caso, también le pedimos variables que acaban con la cadena de texto `cod`:

```
  df_wide |>
    select(ID, matches("cod$")) # $: fin de la cadena de texto
#> # A tibble: 232 x 79
#>      ID WVOC_01_cod WVOC_02_cod WVOC_03_cod WVOC_04_cod WVOC_05_cod
#>   <dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
#> 1     1           2           2           2           1           2
#> 2     2           2           2           2           1           0
#> 3     3           2           2           2           1           2
#> 4     4           2           1           1           1           2
#> 5     5           2           2           1           1           0
#> 6     6           1           1           2           1           2
#> # i 226 more rows
#> # i 73 more variables: WVOC_06_cod <dbl>, WVOC_07_cod <dbl>, ...
```

4.2.3.4 Modificar y añadir variables

Seguimos usando verbos de {dplyr}. Para crear nuevas variables o modificarlas reemplazando los valores usaremos `mutate()`. Dentro de mutate podemos hacer cualquier operación con una o más variables.

Primero leemos nuestra base:

```
name_of_file = here::here("data/files/02-read-csv.csv")
DF_name = read_csv(name_of_file)
DF_name
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Si usamos una variable ya existente, en este caso `PPV_DECLARED`, sus valores se sobrescriben:

```
DF_name |>
  mutate(PPV_DECLARED = PPV_DECLARED/100)
#> # A tibble: 103 x 9
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 1 more variable: PPV_DECLARED <dbl>
```

Si usamos una variable no existente (`PPV_DECLARED_PCT`), la creamos:

```
DF_name |>
  mutate(PPV_DECLARED_PCT = PPV_DECLARED/100)
#> # A tibble: 103 x 10
#>    ...1    ID Genero  Edad Educacion FollowUP condition condition2
#>   <dbl> <dbl>  <dbl> <dbl>     <dbl>    <dbl> <chr>     <chr>
#> 1     4 41904      1    47         8       80 PPV_Cond1 90LInt
#> 2     5 95041      2    21         6       90 PPV_Cond1 90RInt
#> 3     6 74594      2    29         6       10 PPV_Cond1 100LInt
#> 4    15 72903      2    27         7       75 PPV_Cond1 100RInt
#> 5    16 21260      1    29         5       35 PPV_Cond1 90LInt
#> 6    18 50315      2    28         6       14 PPV_Cond1 90RInt
#> # i 97 more rows
#> # i 2 more variables: PPV_DECLARED <dbl>, PPV_DECLARED_PCT <dbl>
```

4.2.3.5 Resúmenes agrupados

La combinación de verbos `group_by()` y `summarise()` es una de las más usadas. Con esta podemos calcular promedios, medianas, etc. por condición de manera sencilla.

`summarise()` nos permite 'resumir' datos, calculando promedios, medianas, o cualquier otro estadístico de interés. En este caso, vemos el promedio de la variable `PPV_DECLARED` usando `mean(PPV_DECLARED, na.rm = TRUE)`. El parámetro `na.rm = TRUE` ignora los valores `NA` que pueda haber en la base:

```
DF_name |>
  summarise(Promedio_PPV = mean(PPV_DECLARED, na.rm = TRUE),
            N = n())
#> # A tibble: 1 x 2
#>   Promedio_PPV     N
#>          <dbl> <int>
#> 1         68.1   103
```

Si incluimos `group_by()` en la pipeline, esos 'resúmenes' se mostraran para cada valor de la variable por la que agrupamos:

```
DF_name |>
  group_by(Genero) |>
  summarise(Promedio_PPV = mean(PPV_DECLARED, na.rm = TRUE),
            N = n())
#> # A tibble: 2 x 3
#>   Genero Promedio_PPV     N
#>    <dbl>        <dbl> <int>
#> 1      1         65.9    40
#> 2      2         69.4    63
```

Podemos agrupar por múltiples variables, y calcular tantas cosas como queramos. En ese caso la media, mediana, desviación estándar y el numero de observaciones:

```
DF_name |>
  group_by(Genero, condition) |>
  summarise(promedio_PPV = mean(PPV_DECLARED, na.rm = TRUE),
            mediana_PPV = median(PPV_DECLARED, na.rm = TRUE),
            SD = sd(PPV_DECLARED, na.rm = TRUE),
            N = n())
#> # A tibble: 8 x 6
#> # Groups:   Genero [2]
#>   Genero condition promedio_PPV mediana_PPV    SD     N
#>    <dbl> <chr>            <dbl>       <dbl> <dbl> <int>
#> 1      1 PPV_Cond1         63.8          88  43.1     9
#> 2      1 PPV_Cond2         50.2          46  10.3    13
#> 3      1 PPV_Cond3         75.6          80  32.5     5
#> 4      1 PPV_Cond4         79.1          91  20.0    13
#> 5      2 PPV_Cond1         72.6          99  43.8    19
#> 6      2 PPV_Cond2         49.4          46  16.3     8
#> # i 2 more rows
```

Ejercicios - verbos dplyr

1. Usando la base df_wide (puedes usar el código de abajo), haz las siguientes cosas, una a una:

```
df_wide = read_csv("https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-
```

- Filtra el DF para quedarnos solo con edades entre 18 y 50 años
- Ordena los datos por genero y edad, esta última decreciente

- Selecciona las columnas para quedarnos solo con ID, variables demográficas, y respuestas crudas (raw)

- Crea una nueva variable llamada `niv_edu_porc`, en la que calcules el nivel educativo al que han llegado dividido por el máximo de la base de datos, pero en porcentaje (nivel educativo persona / nivel educativo máximo; en porcentaje)

2. Ahora combina el resultado de todas las operaciones anteriores en un DF
3. Calcula el promedio y desviación típica de edad para cada género

Pistas

1. Paso a paso:
    - `filter(CONDICION1 & CONDICION2)` o `filter(CONDICION1, CONDICION2)`
    - `arrange()` o `arrange(desc())`
    - `select(starts_with("ALGUN_PATRON"))` o `select(ends_with("ALGUN_PATRON"))`
    - `mutate()` usando también `max()`
2. `DF_resultado = df_wide |> operacion1 |> operation2 |> ...`
3. `group_by() |> summarize()`

4.3 Verbos avanzados y otras criaturas indómitas

4.3.1 Wide to long simple

Empecemos con un ejemplo muy sencillo. 2 participantes, cada uno en una condición, y 2 ítems.

```
df_simple_wide =
  tibble(
    ID = c("Participante1", "Participante2"),
    condition = c("calor", "frio"),
    Item1 = c(22, 33),
    Item2 = c(88, 99)
    )

df_simple_wide
#> # A tibble: 2 x 4
#>   ID           condition Item1 Item2
#>   <chr>        <chr>     <dbl> <dbl>
#> 1 Participante1 calor        22    88
#> 2 Participante2 frio         33    99
```

Pasamos esta base de datos 'ancha' (una fila por participante), a una base en formato 'largo' (una fila por observación) usando la función `pivot_longer()`. Solo tenemos que indicar el rango

de columnas, asignar nombre a la columna donde enviaremos los nombres de las columnas, y asignar nombre a la columna donde estarán los valores:

```
df_simple_long = df_simple_wide |>
  pivot_longer(cols = Item1:Item2,
               names_to = "Item",
               values_to = "Response")


df_simple_long
#> # A tibble: 4 x 4
#>   ID            condition Item  Response
#>   <chr>         <chr>     <chr>    <dbl>
#> 1 Participante1 calor     Item1       22
#> 2 Participante1 calor     Item2       88
#> 3 Participante2 frio      Item1       33
#> 4 Participante2 frio      Item2       99
```

4.3.2 Long to wide simple

Siguiendo con el ejemplo anterior, podemos devolver la base en formato 'largo' a 'ancho' de nuevo usando `pivot_wider()`:

```
df_simple_long |>
  pivot_wider(names_from = Item, values_from = Response)
#> # A tibble: 2 x 4
#>   ID            condition Item1 Item2
#>   <chr>         <chr>     <dbl> <dbl>
#> 1 Participante1 calor        22    88
#> 2 Participante2 frio         33    99
```

4.3.2.1 ¿Para que sirve tener los datos en formato long?

Hay algunos análisis para los que necesitamos formato long (e.g. anovas, modelos mixtos…), muchas gráficas con `ggplot` asumen formato long, y varias cosas se simplifican cuando los datos están en formato largo.

Por ejemplo, si queremos usar resúmenes agrupados para obtener la media, mediana, desviación estándar… por ítem, con el formato WIDE necesitaremos 3 lineas de código para cada ítem que tenga nuestra base (imagina con 100 ítems…). Con el formato long, el código de abajo es suficiente.

```
# En formato wide podríamos usar cosas como:
  # skimr::skim(df_simple_wide)

# Añadir para cada ítem 3 líneas
```

```r
df_simple_wide |>
  summarise(mean_Item1 = mean(Item1),
            mean_Item2 = mean(Item2),
            # ...
            median_Item1 = median(Item1),
            median_Item2 = median(Item2),
            # ...
            sd_Item1 = sd(Item1),
            sd_Item2 = sd(Item2),
            # ...
            N = n()
            # NO aparece N por ítem
            )
#> # A tibble: 1 x 7
#>   mean_Item1 mean_Item2 median_Item1 median_Item2 sd_Item1 sd_Item2     N
#>        <dbl>      <dbl>        <dbl>        <dbl>    <dbl>    <dbl> <int>
#> 1       27.5       93.5         27.5         93.5     7.78     7.78     2

# Sirve para 1 ítem o para 10 millones de ítems
df_simple_long |>
  group_by(Item) |>
  summarise(MEAN = mean(Response),
            MEDIAN = median(Response),
            SD = sd(Response),
            N = n())
#> # A tibble: 2 x 5
#>   Item   MEAN MEDIAN    SD     N
#>   <chr> <dbl>  <dbl> <dbl> <int>
#> 1 Item1  27.5   27.5  7.78     2
#> 2 Item2  93.5   93.5  7.78     2
```

4.3.3 Wide to long complex

Ahora pasemos a un ejemplo mas complejo. Leemos la base de datos y seleccionamos las puntuaciones a los 11 ítems de la lipkus numeracy scale de 232 participantes, ademas de datos demográficos. Usamos `DT::datatable()` para visualizar la base de manera interactiva.

```r
# Leemos documento en formato WIDE
df_wide_complex = read_csv(
  "https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-predictors-of-soc
) |>
  # Seleccionamos solo algunas de las filas
  select(ID,
         dem_genero,
         dem_edad,
         dem_nivedu,
         matches("lkns_[0-9]{2}_raw"))
```

```
DT::datatable(df_wide_complex)
```

Show 10 entries Search:

	ID	dem_genero	dem_edad	dem_nivedu	lkns_01_raw	lkns_02_raw	lkns_03_raw	lkns_04_raw	lkns_05_raw
1	1	1	38	4	500	10	10	b) 1 de cada 1000	b)10%
2	2	0	67	2	0	0	0	c) 1 de cada 10	c) 5%
3	3	0	24	4	700	100	0.1	a) 1 de cada 100	b)10%
4	4	0	30	4	500	30	1	c) 1 de cada 10	b)10%
5	5	0	38	3	6	2	3	a) 1 de cada 100	b)10%
6	6	0	45	4	40	200	2	c) 1 de cada 10	b)10%
7	7	1	58	3	0	0	0	c) 1 de cada 10	b)10%
8	8	1	47	4	500	100	0.1	c) 1 de cada 10	b)10%
9	9	1	52	3	600	1	1	c) 1 de cada 10	c) 5%
10	10	1	49	4	600	500	70	b) 1 de cada 1000	b)10%

Showing 1 to 10 of 232 entries Previous 1 2 3 4 5 … 24 Next

De nuevo, para pasar de formato ‘ancho’ a ‘largo’ completamos los mismos parámetros que antes.

Lo único que añadimos es `values_transform = list(Response = as.character)` para poder incluir en la columna `Response` tanto valores numéricos como caracteres.

```
df_long_complex =
  df_wide_complex |>
  pivot_longer(
    cols = lkns_01_raw:lkns_11_raw,
    names_to = "Item",
    values_to = "Response",
    values_transform = list(Response = as.character)
  )

# Podemos usar select_helpers!
  # Reemplaza lkns_01_raw:lkns_11_raw por matches("lkns")


DT::datatable(df_long_complex)
```

Show 10 entries Search:

	ID	dem_genero	dem_edad	dem_nivedu	Item	Response
1	1	1	38	4	lkns_01_raw	500
2	1	1	38	4	lkns_02_raw	10
3	1	1	38	4	lkns_03_raw	10
4	1	1	38	4	lkns_04_raw	b) 1 de cada 1000
5	1	1	38	4	lkns_05_raw	b)10%
6	1	1	38	4	lkns_06_raw	2
7	1	1	38	4	lkns_07_raw	0
8	1	1	38	4	lkns_08_raw	0
9	1	1	38	4	lkns_09_raw	0
10	1	1	38	4	lkns_10_raw	0

Showing 1 to 10 of 2,552 entries

Previous 1 2 3 4 5 … 256 Next

4.3.4 Long to wide complex

Nos sirve el mismo código que con el ejemplo más simple:

```r
df_long_complex |>
  pivot_wider(names_from = Item, values_from = Response)
#> # A tibble: 232 x 15
#>      ID dem_genero dem_edad dem_nivedu lkns_01_raw lkns_02_raw lkns_03_raw
#>   <dbl>      <dbl>    <dbl>      <dbl> <chr>       <chr>       <chr>
#> 1     1          1       38          4 500         10          10
#> 2     2          0       67          2 0           0           0
#> 3     3          0       24          4 700         100         0.1
#> 4     4          0       30          4 500         30          1
#> 5     5          0       38          3 6           2           3
#> 6     6          0       45          4 40          200         2
#> # i 226 more rows
#> # i 8 more variables: lkns_04_raw <chr>, lkns_05_raw <chr>, ...
```

Ejercicios - wide to long

De nuevo trabajaremos con datos del paper Cognitive and Socio-affective Predictors of Social Adaptation, de Neely et al., pero esta vez van a ser los datos *procesados*. Estos se pueden encontrar en un repositorio público de la OSF. Empezaremos con la base final en formato wide (Dentro de https://osf.io/egxy5/, ver archivo: `/outputs/data/sa-prepared.csv`).

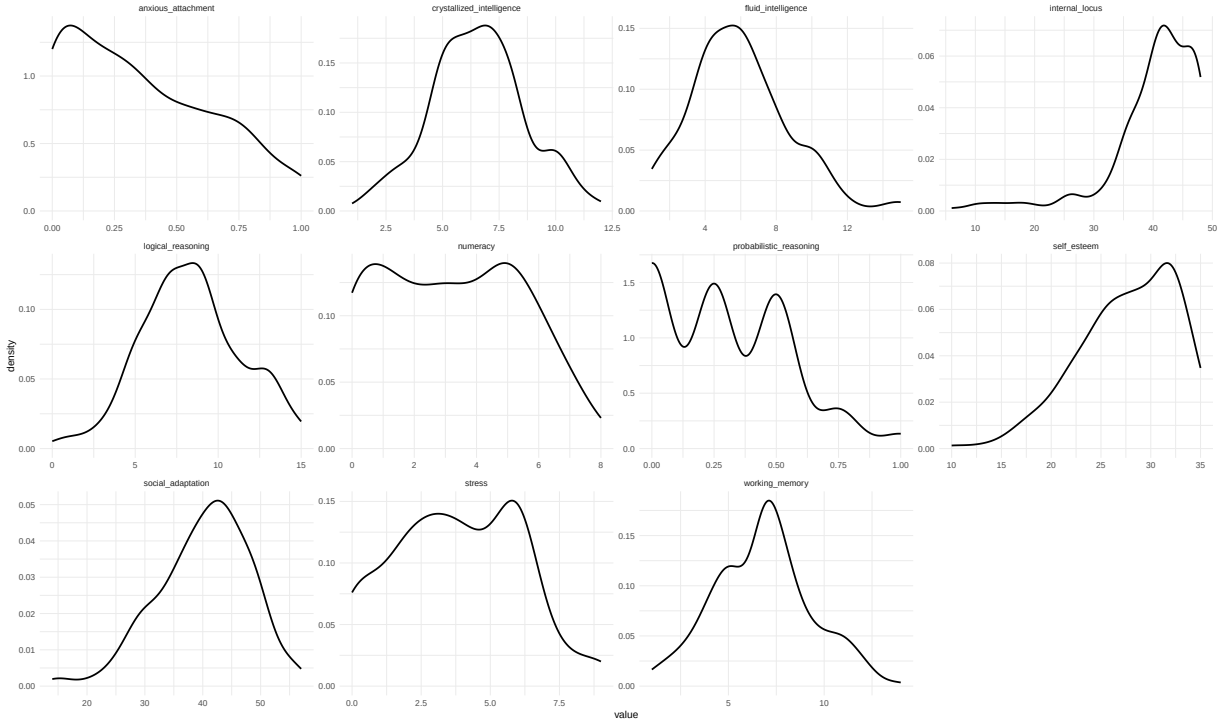
Para simplificar el proceso, puedes importar los datos y limpiar los nombres de variables así:

```r
DF_wide = read_csv(
  "https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-predictors-of-soc
) |>
  janitor::clean_names()
```

1. Cambia el orden de las variables para que ID sea la primera columna.
2. Transforma la base a formato long (eso sí, mantén las variables demográficas en formato wide).
3. Aprovechando que tenemos la base en formato long, sabrías hacer una gráfica con un histograma o densidad para cada una de las variables no demográficas?

> Pistas
>
> 1. Tendrás que usar la función `select()` y el select helper `everything()`
> 2. `pivot_longer(primera_variable:ultima_variable)`
> 3. `facet_wrap(~name, scales = "free")` te ayudara a crear paneles para cada nombre, donde las escalas x/y pueden variar libremente.

El gráfico final se debería ver así:



4.4 Separate, omit, ifelse, case_when, tipos de variables...

Para transformaciones algo más complejas, pero muy habituales, usaremos algunos verbos del paquete {tidyr}, y variaciones con {dplyr}

```
# Base original
DF_name = read_csv(here::here("data/files/02-read-csv.csv")) |>
  select(-...1, -Educacion, -Edad, -condition2)

DT::datatable(DF_name)
```

Show 10 entries Search:

	ID	Genero	FollowUP	condition	PPV_DECLARED
1	41904	1	80	PPV_Cond1	99
2	95041	2	90	PPV_Cond1	99
3	74594	2	10	PPV_Cond1	99
4	72903	2	75	PPV_Cond1	1
5	21260	1	35	PPV_Cond1	24
6	50315	2	14	PPV_Cond1	
7	21774	2	2	PPV_Cond1	99
8	20881	2	89	PPV_Cond1	99
9	39751	2	6	PPV_Cond1	99
10	99384	1	0	PPV_Cond1	1

Showing 1 to 10 of 103 entries

Previous 1 2 3 4 5 … 11 Next

Podemos separar la columna de condición usando la función `separate()` y un separador (`sep = "_"`). La separación puede ser en columnas o en filas. En este primer caso, separamos enviando

cada parte de `condition` a una columna distinta:

```
# Separate
DF_separated = DF_name |>
  separate(condition, c("primer_chunk", "segundo_chunk"), sep = "_")

DF_separated
#> # A tibble: 103 x 6
#>      ID Genero FollowUP primer_chunk segundo_chunk PPV_DECLARED
#>   <dbl>  <dbl>    <dbl> <chr>        <chr>                <dbl>
#> 1 41904      1       80 PPV          Cond1                   99
#> 2 95041      2       90 PPV          Cond1                   99
#> 3 74594      2       10 PPV          Cond1                   99
#> 4 72903      2       75 PPV          Cond1                    1
#> 5 21260      1       35 PPV          Cond1                   24
#> 6 50315      2       14 PPV          Cond1                   NA
#> # i 97 more rows
```

También podemos enviar cada parte de `condition` a una fila distinta:

```
DF_name |>
  separate_rows(condition, sep = "_")
#> # A tibble: 206 x 5
#>      ID Genero FollowUP condition PPV_DECLARED
#>   <dbl>  <dbl>    <dbl> <chr>            <dbl>
#> 1 41904      1       80 PPV                 99
#> 2 41904      1       80 Cond1               99
#> 3 95041      2       90 PPV                 99
#> 4 95041      2       90 Cond1               99
#> 5 74594      2       10 PPV                 99
#> 6 74594      2       10 Cond1               99
#> # i 200 more rows
```

Con `unite()` podemos hacer lo contrario, unir columnas con un separador definido:

```
# Unite: inversa de separate
DF_separated |>
  unite(condition, c(primer_chunk, segundo_chunk), sep = "_")
#> # A tibble: 103 x 5
#>      ID Genero FollowUP condition PPV_DECLARED
#>   <dbl>  <dbl>    <dbl> <chr>            <dbl>
#> 1 41904      1       80 PPV_Cond1           99
#> 2 95041      2       90 PPV_Cond1           99
#> 3 74594      2       10 PPV_Cond1           99
#> 4 72903      2       75 PPV_Cond1            1
#> 5 21260      1       35 PPV_Cond1           24
#> 6 50315      2       14 PPV_Cond1           NA
#> # i 97 more rows
```

Si necesitamos recodificar variables, cambiar valores condicionalmente,... podemos usar `ifelse()`, `case_when()` o `recode()`:

`ifelse()`, si se cumple la condición `Genero == 1`, asigna el valor "Hombre", de lo contrario, "Mujer":

```
DF_name |>
  mutate(Genero = ifelse(Genero == 1, "Hombre", "Mujer"))
#> # A tibble: 103 x 5
#>      ID Genero FollowUP condition PPV_DECLARED
#>   <dbl> <chr>     <dbl> <chr>            <dbl>
#> 1 41904 Hombre       80 PPV_Cond1           99
#> 2 95041 Mujer        90 PPV_Cond1           99
#> 3 74594 Mujer        10 PPV_Cond1           99
#> 4 72903 Mujer        75 PPV_Cond1            1
#> 5 21260 Hombre       35 PPV_Cond1           24
#> 6 50315 Mujer        14 PPV_Cond1           NA
#> # i 97 more rows
```

Con `case_when()` podemos establecer varias condiciones lógicas simultáneamente, ademas de un valor por defecto si no se cumple ninguna de ellas. Las condiciones lógicas pueden ser arbitrariamente complejas:

```
DF_name |>
  mutate(Genero =
           case_when(
             Genero == 1 ~ "Hombre",
             Genero == 2 ~ "Mujer",
             Genero == 3 ~ "No binario",
             TRUE ~ NA_character_)
         )
#> # A tibble: 103 x 5
#>      ID Genero FollowUP condition PPV_DECLARED
#>   <dbl> <chr>     <dbl> <chr>            <dbl>
#> 1 41904 Hombre       80 PPV_Cond1           99
#> 2 95041 Mujer        90 PPV_Cond1           99
#> 3 74594 Mujer        10 PPV_Cond1           99
#> 4 72903 Mujer        75 PPV_Cond1            1
#> 5 21260 Hombre       35 PPV_Cond1           24
#> 6 50315 Mujer        14 PPV_Cond1           NA
#> # i 97 more rows
```

Finalmente podemos usar `recode()`, con una sintaxis tal vez algo más sencilla:

```
  DF_name |>
  select(Genero) |>
```

```
    # De número a texto
    mutate(Genero2 = recode(
      Genero,
      `1` = "Hombre",
      `2` = "Mujer",
      .default = "No definido"
    )) |>

    # De texto a número
    mutate(Genero3 = recode(
      Genero2,
      "Hombre" = 1,
      "Mujer" = 0,
      .default = 999
    ))
#> # A tibble: 103 x 3
#>   Genero Genero2 Genero3
#>    <dbl> <chr>     <dbl>
#> 1      1 Hombre        1
#> 2      2 Mujer         0
#> 3      2 Mujer         0
#> 4      2 Mujer         0
#> 5      1 Hombre        1
#> 6      2 Mujer         0
#> # i 97 more rows
```

Otras funciones útiles, extraer los valores de una columna con `pull()` o descartar los `NA` de una columna con `drop_na()`.

Usamos `pull()` para extraer los valores de una columna.

```
DF_name |> pull(PPV_DECLARED)
#>   [1] 99 99 99  1 24 NA 99 99 99  1 94 99 88  0  1 99 99 99 70 99  1 99 99  7
#>  [25] 10 99 99 99 46 45 46 40 NA 46 46 73 46 50 46 45 46 87 46 49 30 46 50 70
#>  [49] 44 80 99 99 99 99 80 51 99 20 30  1 20  5 30 99 99 99 99 80 98 99 80 59
#>  [73] 64 16 79 92 92 80 90 60 93 92 28 92 92 77 74 90 10 92 92 92 65 20 92 92
#>  [97] 92 92 90 92 NA 92 80
```

Calculamos el promedio de una columna (con pipes).

```
DF_name |> pull(PPV_DECLARED) |> mean()
#> [1] NA
```

Calculamos el promedio usando `na.rm = TRUE` para que la función `mean()` ignore los NA's:

```
DF_name |> pull(PPV_DECLARED) |> mean(na.rm = TRUE)
#> [1] 68.06
```

Calculamos la media de manera más sencilla, en este caso usando base R. No siempre son necesarias las pipes. Si escribes `DF_name$` y presionas el tabulador, aparecerán todas las columnas de esa base:

```
mean(DF_name$PPV_DECLARED, na.rm = TRUE)
#> [1] 68.06
```

Eliminamos las filas que tienen valores perdidos en la variable indicada. En este caso pasamos de 103 a 100 observaciones:

```
DF_name |>  drop_na(PPV_DECLARED)
#> # A tibble: 100 x 5
#>      ID Genero FollowUP condition PPV_DECLARED
#>   <dbl>  <dbl>    <dbl> <chr>            <dbl>
#> 1 41904      1       80 PPV_Cond1           99
#> 2 95041      2       90 PPV_Cond1           99
#> 3 74594      2       10 PPV_Cond1           99
#> 4 72903      2       75 PPV_Cond1            1
#> 5 21260      1       35 PPV_Cond1           24
#> 6 21774      2        2 PPV_Cond1           99
#> # i 94 more rows
```

Ejercicios - verbos avanzados dplyr

1. Importa los datos y limpia los nombres de columna:

> Para limpiar nombres de columnas automáticamente:
>
> `clean_names()`

```
# Leemos los datos y usamos janitor::clean_names() para limpiar los nombres de las columnas
 DF_wide =
  read_csv("https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-predicto
```

2. En un nuevo DF (`DF_split`), crea una variable llamada `social_adaptation_split` con la median split para la variable social_adaptation. La mitad superior se llamará `high_social_adaptation` y la mitad inferior `low_social_adaptation`.

> Suele ser más facil si dividimos la tarea en varios pasos
>
> 1. Calculamos mediana 2. Usamos `case_when()`

3. Asegúrate que no hay valores NA.

> Pista
>
> La función `drop_na()` .

El resultado final debería ser:

Show 10 entries Search:

	id	social_adaptation	social_adaptation_split
1	1	38	low_social_adaptation
2	2	32	low_social_adaptation
3	3	43	high_social_adaptation
4	4	46	high_social_adaptation
5	5	32	low_social_adaptation
6	6	39	low_social_adaptation
7	7	41	high_social_adaptation
8	8	44	high_social_adaptation
9	9	45	high_social_adaptation
10	10	50	high_social_adaptation

Showing 1 to 10 of 226 entries

Previous 1 2 3 4 5 … 23 Next

4.5 Regular expressions

Las expresiones regulares son una herramienta tan potente como difícil de utilizar. Eso si, podemos hacer algunas cosas básicas muy útiles, sin demasiado esfuerzo. Hay cheatsheets (Basic Regular Expressions Cheatsheet) y libros (introduction to Regular Expressions) que nos pueden ayudar a familiarizarnos con ellas.

Figure 4.1: SOURCE: https://xkcd.com/208/

Imagina que tenemos que trabajar con la columna `condition2`, donde están codificadas 3 variables importantes:

```
DF_regexp = read_csv(here::here("data/files/02-read-csv.csv")) |>
  select(-...1, -Educacion, -Edad, -condition)

DF_regexp
#> # A tibble: 103 x 5
#>      ID Genero FollowUP condition2 PPV_DECLARED
#>   <dbl>  <dbl>    <dbl> <chr>             <dbl>
#> 1 41904      1       80 90LInt               99
#> 2 95041      2       90 90RInt               99
#> 3 74594      2       10 100LInt              99
#> 4 72903      2       75 100RInt               1
#> 5 21260      1       35 90LInt               24
#> 6 50315      2       14 90RInt               NA
#> # i 97 more rows
```

Cuando no tenemos separadores explícitos como vimos antes con `separate()`, podemos usar `mutate()` junto a `gsub()` y expresiones regulares para extraer, una a una, las condiciones.

La función `gsub()` nos sirve para eliminar partes de una cadena de texto, para extraer un número, etc.:

```
DF_regexp |>
  mutate(cond_NM = gsub("([0-9]{2,3}).*", "\\1", condition2),
         cond_LR = gsub("[0-9]{2,3}([LR]).*", "\\1", condition2),
         cond_IA = gsub("[0-9]{2,3}[LR](.*)", "\\1", condition2))
#> # A tibble: 103 x 8
#>      ID Genero FollowUP condition2 PPV_DECLARED cond_NM cond_LR cond_IA
#>   <dbl>  <dbl>    <dbl> <chr>             <dbl> <chr>   <chr>   <chr>
#> 1 41904      1       80 90LInt               99 90      L       Int
#> 2 95041      2       90 90RInt               99 90      R       Int
#> 3 74594      2       10 100LInt              99 100     L       Int
#> 4 72903      2       75 100RInt               1 100     R       Int
#> 5 21260      1       35 90LInt               24 90      L       Int
#> 6 50315      2       14 90RInt               NA 90      R       Int
#> # i 97 more rows
```

Extraemos la misma información, de una manera ligeramente distinta, siendo mucho más explícitos sobre la estructura esperada de la columna condition2:

```
DF_regexp |>
  mutate(cond_NM = gsub("^([0-9]{2,3})([LR])(.*)$", "\\1", condition2),
         cond_LR = gsub("^([0-9]{2,3})([LR])(.*)$", "\\2", condition2),
         cond_IA = gsub("^([0-9]{2,3})([LR])(.*)$", "\\3", condition2))
#> # A tibble: 103 x 8
```

```
#>      ID Genero FollowUP condition2 PPV_DECLARED cond_NM cond_LR cond_IA
#>   <dbl>  <dbl>    <dbl> <chr>             <dbl> <chr>   <chr>   <chr>
#> 1 41904      1       80 90LInt               99 90      L       Int
#> 2 95041      2       90 90RInt               99 90      R       Int
#> 3 74594      2       10 100LInt              99 100     L       Int
#> 4 72903      2       75 100RInt               1 100     R       Int
#> 5 21260      1       35 90LInt               24 90      L       Int
#> 6 50315      2       14 90RInt               NA 90      R       Int
#> # i 97 more rows
```

Con `select()` y `matches()` seleccionamos columnas usando la siguiente regular expression `lkns_[0-9]{2}_raw`:

- `lkns_` contiene esta cadena de texto

- `[0-9]{2}` a continuación, contiene cualquier dígito del 0 al 9, dos veces.
- `_raw`a continuación, contiene esta cadena de texto

```
read_csv("https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-predictors
  # Seleccionamos solo algunas de las filas
  select(ID, dem_genero, dem_edad, dem_nivedu, matches("lkns_[0-9]{2}_raw"))
#> # A tibble: 232 x 15
#>      ID dem_genero dem_edad dem_nivedu lkns_01_raw lkns_02_raw lkns_03_raw
#>   <dbl>      <dbl>    <dbl>      <dbl>       <dbl>       <dbl>       <dbl>
#> 1     1          1       38          4         500          10        10
#> 2     2          0       67          2           0           0         0
#> 3     3          0       24          4         700         100         0.1
#> 4     4          0       30          4         500          30         1
#> 5     5          0       38          3           6           2         3
#> 6     6          0       45          4          40         200         2
#> # i 226 more rows
#> # i 8 more variables: lkns_04_raw <chr>, lkns_05_raw <chr>, ...
```

4.5.1 Ayuda con regular expressions

Es muy fácil cometer errores cuando usamos expresiones regulares. Algunas recomendaciones:

1) Solo usar expresiones regulares cuando sea necesario

2) Usar expresiones regulares lo más explícitas y definidas posible

3) Verificar que están funcionando bien!

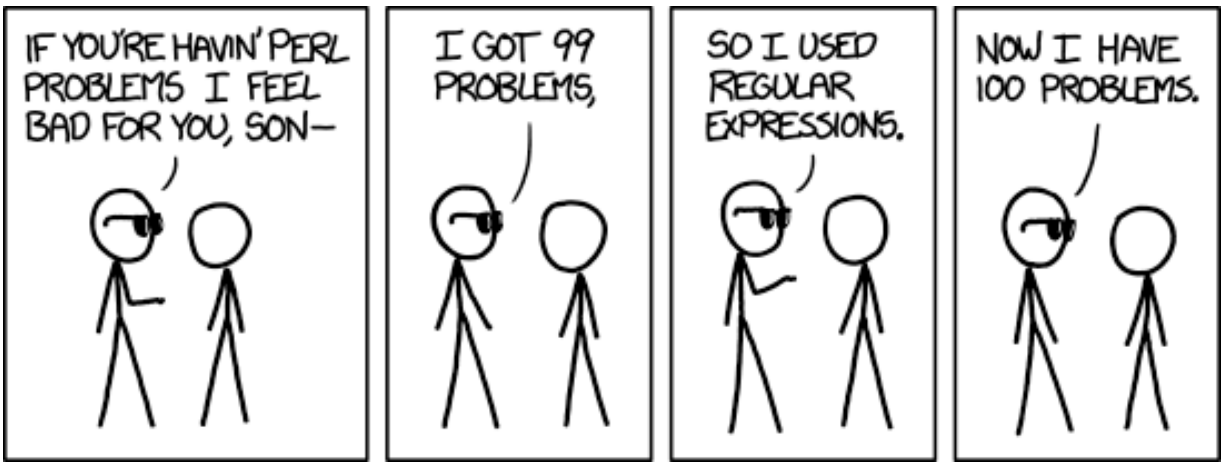


Figure 4.2: SOURCE: https://xkcd.com/1171/

Hay una aplicación Shiny muy útil que nos ayudará a construir Regular Expressions:

```
regexplain::regexplain_gadget()
```

Ejercicios - Calcular puntajes de escalas usando regular expressions

Ahora volvemos a usar con los datos brutos (`sa-raw-anonymised.csv`) del paper Cognitive and Socio-affective Predictors of Social Adaptation, de Neely et al.

En estos datos tenemos las puntuaciones crudas (e.g. WMAT_01_raw) y ya codificadas/corregidas (WMAT_01_cod) para los ítems de varias escalas Para preparar los datos de cara al análisis final, necesitamos calcular el puntaje para cada participante y escala. Empezaremos con la prueba de Matrices de WAIS (`WMAT_`).

1. Calcula el puntaje para cada participante en la prueba de Matrices de WAIS (ítems WMAT_[NUMEROS]_cod)

Hay al menos dos estrategias posibles:

A) Selecciona las columnas relevantes y haz la suma de columnas

B) Convierte a long, filtra para quedarte con las filas correspondientes a la prueba relevante, y haz una suma agrupada

> Pista para seleccionar o filtrar columnas:
>
> Recuerda que usamos `select()` para seleccionar columnas, o `filter()` para filtrar.

Pista para seleccionar columnas:

Podemos usar `matches("WMAT_[0-9]{2}_cod")` para seleccionar o filtrar todas las columnas o ítems que contienen: `WMAT_`, 2 números del 0 al 9, y acaban en `_cod`.

Pista para suma de columnas:

`rowSums()` es la función que podemos usar, pero su sintaxis es algo complicada.

Pista para suma agrupada:

Usamos `group_by() |> summarise()` poniendo parámetros dentro de cada función.

Importar datos:

```
df_wide_raw = read_csv("https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affect
```

Bibliografía

Cheatsheets RStudio

Cheatsheet dplyr

Tidyexplain

5 Combinar datos

Combinar distintos data frames es una tarea muy común cuando preparamos datos. En ocasiones trabajaremos con distintos archivos que tendremos que combinar, y otras veces, separaremos nuestra base original en distintos data frames, para realizar procesamientos diferenciados, y más adelante volver a combinar los data frames en una base final.

Paquetes para este capítulo

```
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require("DT")) install.packages("DT"); library("DT")
if (!require("ggplot2")) install.packages("ggplot2"); library("ggplot2")
if (!require("here")) install.packages("here"); library("here")
if (!require("janitor")) install.packages("janitor"); library("janitor")
if (!require("purrr")) install.packages("purrr"); library("purrr")
if (!require('readr')) install.packages('readr'); library('readr')
if (!require("readxl")) install.packages("readxl"); library("readxl")
if (!require("tidyr")) install.packages("tidyr"); library("tidyr")
if (!require("waldo")) install.packages("waldo"); library("waldo")
```

5.1 Bind rows or columns

El método más sencillo. Simplemente unimos las filas o columnas de los data frames.

Primero importamos dos DFs:

```
DF1 = read_csv(here::here("data/files/02-CSVs/01.csv"))
DF2 = read_csv(here::here("data/files/02-CSVs/02.csv"))
```

Con `bind_rows()` podemos añadir las *filas* de DF2 a DF1:

```
DF1 |> bind_rows(DF2)
#> # A tibble: 800 x 9
#>   Sex   Priming    trialN Block Adjective  Valence  Answer Arrow    rT
#>   <chr> <chr>       <dbl> <chr> <chr>      <chr>    <chr>  <chr> <dbl>
#> 1 male  Collective      1 we    ofensivo   negative yes    left    623
#> 2 male  Collective      2 we    resentido  negative no     right  1235
#> 3 male  Collective      3 we    ego sta    negative yes    left    335
#> 4 male  Collective      4 we    indiscreto negative yes    left    355
#> 5 male  Collective      5 we    sumiso     negative yes    left    618
#> 6 male  Collective      6 we    agradable  positive yes    left    328
#> # i 794 more rows
```

Con `bind_cols()` añadimos las *columnas* de DF2 a DF1. `bind_cols()` renombra automáticamente los nombres de las columnas para que no haya coincidencias:

```
DF1 |> bind_cols(DF2)
#> # A tibble: 400 x 18
#>   Sex...1 Priming...2 trialN...3 Block...4 Adjective...5 Valence...6
#>   <chr>   <chr>            <dbl> <chr>     <chr>         <chr>
#> 1 male    Collective           1 we        ofensivo      negative
#> 2 male    Collective           2 we        resentido     negative
#> 3 male    Collective           3 we        ego sta       negative
#> 4 male    Collective           4 we        indiscreto    negative
#> 5 male    Collective           5 we        sumiso        negative
#> 6 male    Collective           6 we        agradable     positive
#> # i 394 more rows
#> # i 12 more variables: Answer...7 <chr>, Arrow...8 <chr>, rT...9 <dbl>, ...
```

5.2 Joins

El paquete {dplyr} tiene funciones que permiten trabajar combinando, filtrando, etc. distintos data frames. Podéis ver más detalle y algunas ilustraciones fantásticas (como la de abajo; **inner_join()**) en el capítulo relational data de r4ds.

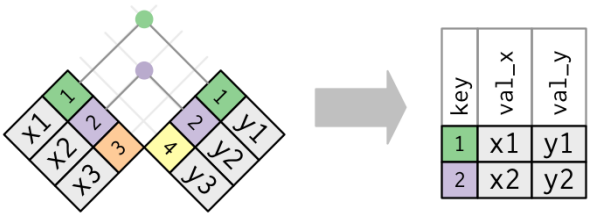


Figure 5.1: SOURCE: https://r4ds.had.co.nz/relational-data.html#mutating-joins

En https://github.com/gadenbuie/tidyexplain se pueden ver animaciones mostrando estas operaciones.

Tipos de Join

Estas operaciones tendrán la forma: `DF_x |> WHATEVER_join(DF_y)`

- **Mutating joins**:
  - inner_join(): preserva pares de observaciones de `DF_x` y de `DF_y` con claves iguales
  - left_join(): preserva las observaciones de `DF_x`, añadiendo las de `DF_y` con claves iguales
  - right_join(): preserva las observaciones de `DF_y`, añadiendo las de `DF_x` con claves iguales
  - full_join(): preserva todas las observaciones de `DF_x` y `DF_y`, alineándolas cuando tengan claves iguales
- **Filtering joins**:
  - semi_join(): preserva solo aquellas observaciones de `DF_x` cuyas claves aparezcan en `DF_y`
  - anti_join(): preserva solo aquellas observaciones de `DF_x` cuyas claves NO aparezcan en `DF_y`
- **Nesting joins**:
  - nest_join(): preserva las observaciones de `DF_x`, añadiendo las de `DF_y` con claves iguales

5.2.1 Mutating joins

Importamos datos

Tenemos los siguientes data frames:

- DF_IDs: Variables demográficas de participantes
- DF_results: Resultados en variables de interés de participantes
- DF_BAD: Grupo de participantes "selectos"

```
# Importar CSVs para los joins
DF_IDs = read_csv(here::here("data/files/02-join-IDs.csv"))
DF_results = read_csv(here::here("data/files/02-join-results.csv"))
DF_BAD = read_csv(here::here("data/files/02-join-BAD.csv"))
```

5.2.1.1 Inner join

Preserva pares de observaciones de `DF_x` y de `DF_y` con claves iguales (fijaros en el mensaje que aparece en la Consola: `Joining, by = "ID"`).

See animation

```
DF_inner_joined =
  DF_IDs |>
  inner_join(DF_results)

#nrow(DF_inner_joined)

DT::datatable(DF_inner_joined)
```

Show 10 entries

Search:

	ID	Sex	Age	Education Level	Fluid Intelligence	Crystallized Intelligence	Self-Esteem	Internal Locus	Logical Reasoning	Anxious Attachment	S Adapt
1	1	1	38	4	6	7	26	48	13	0.083333333333333	
2	2	0	67	2	6	9	26	37	9	0	
3	3	0	24	4	9	7	29	40	15	0.25	
4	4	0	30	4	4	5	35	46	9	0	
5	5	0	38	3	5	6	22	34	4	0.166666666666667	
6	6	0	45	4	6	7	23	36	14	0.166666666666667	
7	7	1	58	3	6	7	29	35	8	0.25	
8	8	1	47	4	6	9	27	40	11	0	
9	9	1	52	3	5	8	27	48	11	0.333333333333333	
10	10	1	49	4	9	5	28	41	9	0.666666666666667	

Showing 1 to 10 of 232 entries

Previous 1 2 3 4 5 … 24 Next

5.2.1.2 Left join

Preserva las observaciones de `DF_x`, añadiendo las de `DF_y` con claves iguales (columnas con el mismo nombre).

See animation

```
DF_left_joined = DF_IDs |>
   left_join(DF_results)

# Vemos el número de filas de cada data frame
# nrow(DF_left_joined)
# map(list("DF_left_joined" = DF_left_joined, "DF_IDs" = DF_IDs, "DF_results" = DF_results),

DT::datatable(DF_left_joined)
```

Show 10 entries Search:

	ID	Sex	Age	Education Level	Fluid Intelligence	Crystallized Intelligence	Self-Esteem	Internal Locus	Logical Reasoning	Anxious Attachment	S Adapt
1	1	1	38	4	6	7	26	48	13	0.083333333333333	
2	2	0	67	2	6	9	26	37	9	0	
3	3	0	24	4	9	7	29	40	15	0.25	
4	4	0	30	4	4	5	35	46	9	0	
5	5	0	38	3	5	6	22	34	4	0.166666666666667	
6	6	0	45	4	6	7	23	36	14	0.166666666666667	
7	7	1	58	3	6	7	29	35	8	0.25	
8	8	1	47	4	6	9	27	40	11	0	
9	9	1	52	3	5	8	27	48	11	0.333333333333333	
10	10	1	49	4	9	5	28	41	9	0.666666666666667	

Showing 1 to 10 of 235 entries Previous 1 2 3 4 5 … 24 Next

Si no tenemos columnas con el mismo nombre en ambos data frames, tenemos que indicarle a la función a partir de que dos columnas queremos unir los data frames. Por ejemplo, con `by =`

`c("ID" = "Identificador")` le decimos que la columna `ID` el primer data frame corresponde a `Identificador` del segundo data frame.

```
# Renombramos el identificador para que no coincidan
DF_results2 = DF_results |> rename(Identificador = ID)

# Si no hay variables en común, nos da un error:

# DF_left_joined = DF_IDs |>
#    left_join(DF_results2)
  # Error in `left_join()`:
  # ! `by` must be supplied when `x` and `y` have no common variables.
#   use by = character()` to perform a cross-join.

# Tenemos que indicar explicitamente que identificador del primer data frame (DF_IDs)
  # coincide con que identificador del segundo data frame (DF_results2)
DF_left_joined = DF_IDs |>
   left_join(DF_results2, by = c("ID" = "Identificador"))

# En las últimas versiones de dplyr, han implementado la función `join_by()` que
  # permite usar una sintaxis algo más natural:
  DF_left_joined2 = DF_IDs |>
    left_join(DF_results2, by = join_by(ID == Identificador))

# Comparar si todo es =
  # waldo::compare(DF_left_joined, DF_left_joined2)
```

5.2.1.3 Full join

Preserva todas las observaciones de `DF_x` y `DF_y`, alineándolas cuando tengan claves iguales.

See animation

```
DF_full_joined = DF_IDs |>
   full_join(DF_results)

# CHECK
map(list("DF_full_joined" = DF_full_joined, "DF_IDs" = DF_IDs, "DF_results" = DF_results), n
#> $DF_full_joined
#> [1] 237
#>
#> $DF_IDs
#> [1] 235
#>
#> $DF_results
#> [1] 234

DT::datatable(DF_full_joined)
```

Show 10 entries Search:

	ID	Sex	Age	Education Level	Fluid Intelligence	Crystallized Intelligence	Self-Esteem	Internal Locus	Logical Reasoning	Anxious Attachment	S Adapt
1	1	1	38	4	6	7	26	48	13	0.083333333333333	
2	2	0	67	2	6	9	26	37	9	0	
3	3	0	24	4	9	7	29	40	15	0.25	
4	4	0	30	4	4	5	35	46	9	0	
5	5	0	38	3	5	6	22	34	4	0.166666666666667	
6	6	0	45	4	6	7	23	36	14	0.166666666666667	
7	7	1	58	3	6	7	29	35	8	0.25	
8	8	1	47	4	6	9	27	40	11	0	
9	9	1	52	3	5	8	27	48	11	0.333333333333333	
10	10	1	49	4	9	5	28	41	9	0.666666666666667	

Showing 1 to 10 of 237 entries Previous 1 2 3 4 5 … 24 Next

5.2.2 Filtering joins

5.2.2.1 Anti join

Preserva solo aquellas observaciones de `DF_x` cuyas claves NO aparezcan en `DF_y`.

See animation

```
# AVOID the people present in DF_BAD
DF_anti_joined = DF_IDs |>
  anti_join(DF_BAD, by = "ID") |>
  left_join(DF_results)

# CHECK
map(list("DF_anti_joined" = DF_anti_joined, "DF_IDs" = DF_IDs, "DF_BAD" = DF_BAD, "DF_result
#> $DF_anti_joined
#> [1] 226
#>
#> $DF_IDs
#> [1] 235
#>
#> $DF_BAD
#> [1] 9
#>
#> $DF_results
#> [1] 234


DT::datatable(DF_anti_joined)
```

Show 10 entries Search:

	ID	Sex	Age	Education Level	Fluid Intelligence	Crystallized Intelligence	Self-Esteem	Internal Locus	Logical Reasoning	Anxious Attachment	S Adapt
1	1	1	38	4	6	7	26	48	13	0.083333333333333	
2	3	0	24	4	9	7	29	40	15	0.25	
3	4	0	30	4	4	5	35	46	9	0	
4	5	0	38	3	5	6	22	34	4	0.166666666666667	
5	6	0	45	4	6	7	23	36	14	0.166666666666667	
6	7	1	58	3	6	7	29	35	8	0.25	
7	8	1	47	4	6	9	27	40	11	0	
8	9	1	52	3	5	8	27	48	11	0.333333333333333	
9	10	1	49	4	9	5	28	41	9	0.666666666666667	
10	11	0	57	1	6	7	32	48	5	0.166666666666667	

Showing 1 to 10 of 226 entries Previous 1 2 3 4 5 … 23 Next

5.2.2.2 Semi join

Preserva solo aquellas observaciones de `DF_x` cuyas claves aparezcan en `DF_y`. La diferencia con `inner_join()` es que NO se preservan las observaciones de `DF_y`.

See animation

```
# INCLUDE ONLY the people present in DF_BAD
DF_semi_joined = DF_IDs |>
  semi_join(DF_BAD, by = "ID") |>
  left_join(DF_results)

# CHECK
map(list("DF_semi_joined" = DF_semi_joined,
         "DF_IDs" = DF_IDs,
         "DF_BAD" = DF_BAD,
         "DF_results" = DF_results),
    nrow)
#> $DF_semi_joined
#> [1] 9
#>
#> $DF_IDs
#> [1] 235
#>
#> $DF_BAD
#> [1] 9
#>
#> $DF_results
#> [1] 234

DT::datatable(DF_semi_joined)
```

Show 10 entries Search:

	ID	Sex	Age	Education Level	Fluid Intelligence	Crystallized Intelligence	Self-Esteem	Internal Locus	Logical Reasoning	Anxious Attachment	Sc Adapta
1	2	0	67	2	6	9	26	37	9	0	
2	61	0	65	1	6	8	26	40	6	0.666666666666667	
3	62	0	56	1	4	6	28	47	12	0.25	
4	122	1	59	1	7	7	32	42	9	0.416666666666667	
5	218	0	24	4	5	8	34	38	13	0.083333333333333	
6	229	0	52	4	10	11	24	46	15	0.416666666666667	
7	233	1	23	3							
8	234	1	22	3							
9	235	1	24	3							

Showing 1 to 9 of 9 entries Previous 1 Next

Ejercicios JOINS

Con los DFs de abajo, haz las siguientes operaciones:

```
DF_IDs = read_csv(here::here("data/files/02-join-IDs2.csv"))
DF_results = read_csv(here::here("data/files/02-join-results.csv"))
DF_BAD = read_csv(here::here("data/files/02-join-BAD.csv"))
```

1. Une los datos demográficos con los resultados.

> Pista para unir bases:
>
> Vimos en el apartado left_join() como hacer esto

2. A la base resultante, quítale los sujetos descartados de `DF_BAD`.

> Pista descartar filas:
>
> anti_join()!

3. Crea una nueva base con datos demográficos y resultados para los sujetos descartados.

> Pista para filtrar a partir de una base:
>
> semi_join()!

4. Comprueba si el promedio para `Crystallized Intelligence` de los participantes descartados difiere de la de los no descartados.

> Pista para promedios agrupados:
>
> `group_by() |> summarise()`

5. Haz una gráfica donde se puedan ver las diferencias

---

6. En el ejercicio 3 de verbos avanzados creaste un DF llamado `DF_split` con la median split a partir de la variable `Social.Adaptation`.

```
DF_wide = read_csv(
  "https://raw.githubusercontent.com/gorkang/cognitive-and-socio-affective-predictors-of-soc
) |>
  janitor::clean_names()

median_social_adaptation = DF_wide |>
  pull(social_adaptation) |>
  median(., na.rm = TRUE)

DF_split = DF_wide |>
  mutate(social_adaptation_split =
```

```
            as.factor(
              case_when(
                social_adaptation >= median_social_adaptation ~ "high_social_adaptation",
                social_adaptation < median_social_adaptation ~ "low_social_adaptation",
                TRUE ~ NA_character_
              )
            )) |>
  select(id, social_adaptation, social_adaptation_split) |>
  drop_na(social_adaptation_split)

DF_long = DF_wide |> pivot_longer(fluid_intelligence:working_memory)
```

Uno ese DF al `DF_long` que habías creado en el ejercicio 2 de la misma sección. El DF final se vera así:

Show 10 entries Search:

	sex	age	education_level	id	name	value	social_adaptation	social_adaptation_split
1	1	38	4	1	fluid_intelligence	6	38	low_social_adaptation
2	1	38	4	1	crystallized_intelligence	7	38	low_social_adaptation
3	1	38	4	1	self_esteem	26	38	low_social_adaptation
4	1	38	4	1	internal_locus	48	38	low_social_adaptation
5	1	38	4	1	logical_reasoning	13	38	low_social_adaptation
6	1	38	4	1	anxious_attachment	0.08333333333333333	38	low_social_adaptation
7	1	38	4	1	social_adaptation	38	38	low_social_adaptation
8	1	38	4	1	stress	4	38	low_social_adaptation
9	1	38	4	1	numeracy	3	38	low_social_adaptation
10	1	38	4	1	probabilistic_reasoning	0.5	38	low_social_adaptation

Showing 1 to 10 of 2,486 entries Previous 1 2 3 4 5 … 249 Next

7. Haz un plot donde se vea la distribución para todas las variables de resultados de los dos niveles de `social_adaptation_split`.

5.3 Datasets interesantes

En los siguientes repositorios podréis encontrar datasets interesantes para jugar.

- fivethirtyeight
- Our World in Data
- TidyTuesday

Bibliografía

Cheatsheets RStudio

data-carpentry-week lesson_joins

R4ds - Joins

Tidyexplain

6 Análisis de datos exploratorio

Paquetes para este capítulo

```
if (!require('corrplot')) install.packages('corrplot'); library('corrplot')
if (!require('cowplot')) install.packages('cowplot'); library('cowplot')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('gapminder')) install.packages('gapminder'); library('gapminder')
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
if (!require('ggridges')) install.packages('ggridges'); library('ggridges')
if (!require('haven')) install.packages('haven'); library('haven')
if (!require('inspectdf')) install.packages('inspectdf'); library('inspectdf')
if (!require('tidyr')) install.packages('tidyr'); library('tidyr')
```

En este capítulo vamos a aplicar lo que hemos aprendido en los dos capítulos anteriores, combinando transformación de datos con visualización para entender nuestras bases de datos, buscar patrones interesantes, etc. Podéis encontrar una introducción más completa en el manual R 4 data science - exploratory data analysis.

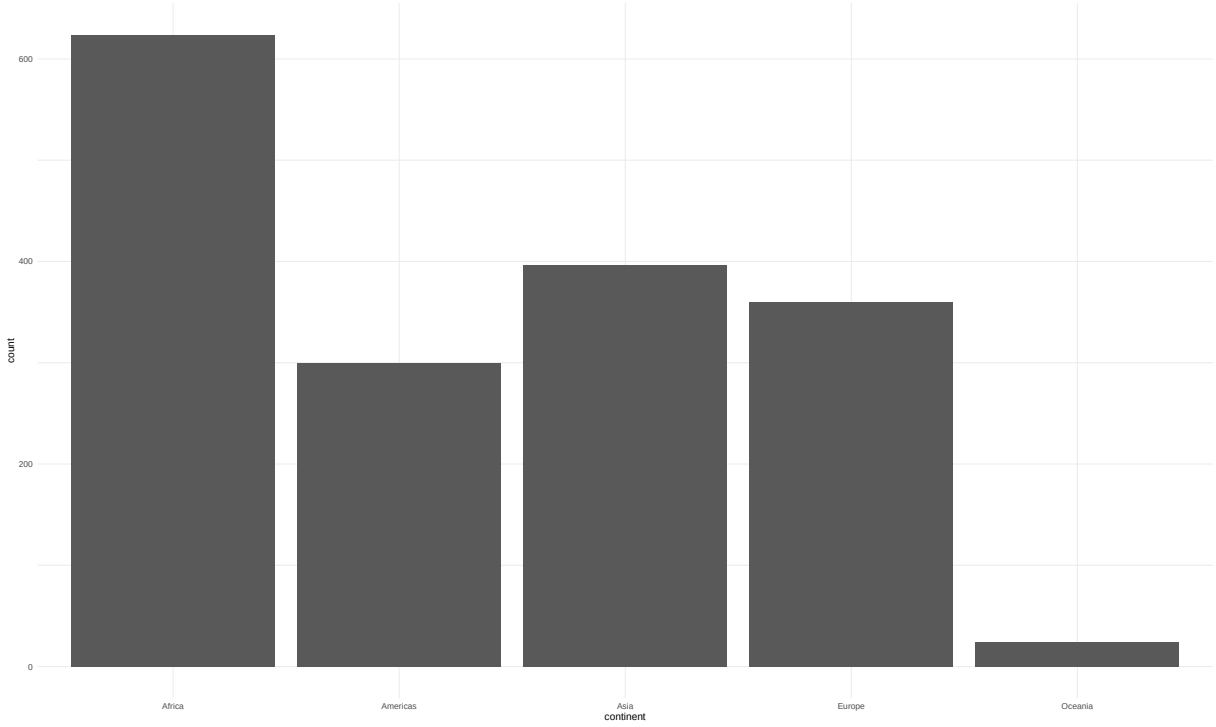
6.1 Visualizando distribuciones

Para visualizar la distribución de nuestras variables, tendremos que seguir estrategias diferentes dependiendo de si se trata de variables categóricas o continuas.

6.1.1 Variables categóricas

```
ggplot(gapminder, aes(continent)) +
  geom_bar()

gapminder |>
  count(continent)
#> # A tibble: 5 x 2
#>   continent     n
```

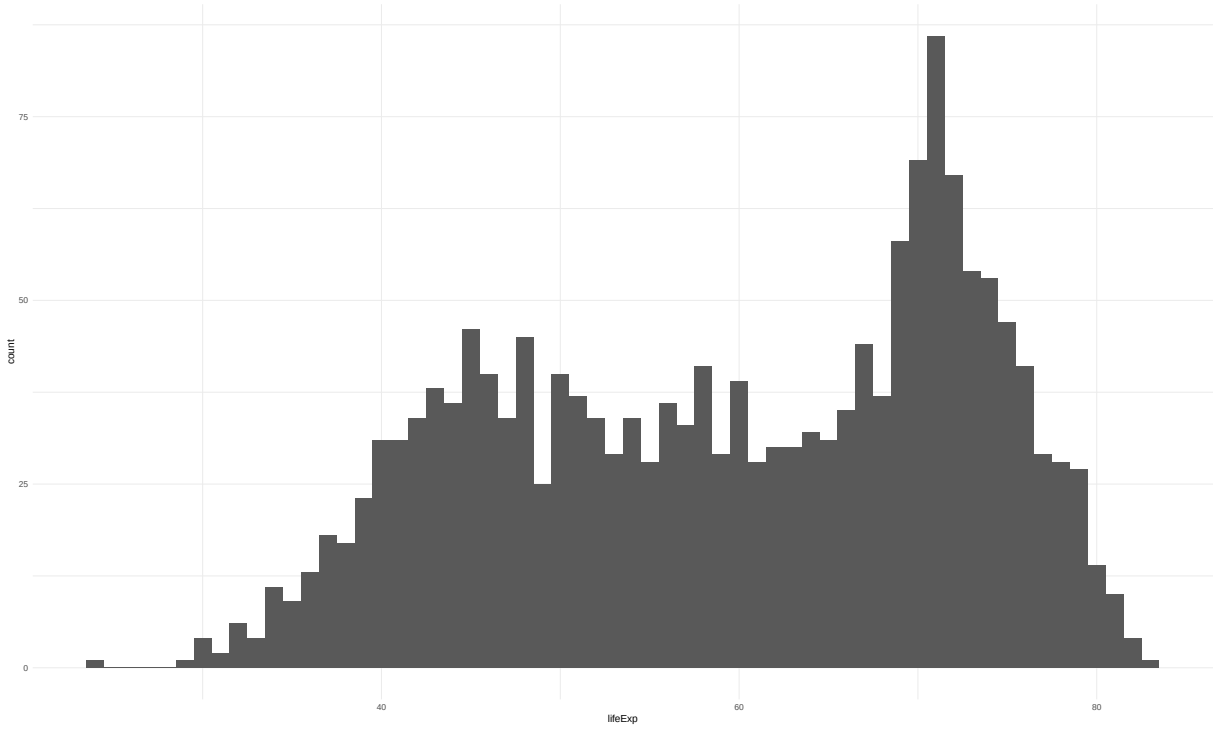
```
#>   <fct>     <int>
#> 1 Africa      624
#> 2 Americas    300
#> 3 Asia        396
#> 4 Europe      360
#> 5 Oceania      24
```



6.1.2 Variables continuas

Para ver la distribución de una variable podemos empezar con un histograma sencillo.

```
ggplot(gapminder, aes(lifeExp)) +
  geom_histogram(binwidth = 1)
```



`summarise()` nos permite ver medias, medianas, etc.

```
gapminder |>
  summarise(MEAN = mean(lifeExp),
            MEDIAN = median(lifeExp),
            SD = sd(lifeExp),
            MAX = max(lifeExp),
            MIN = min(lifeExp))
#> # A tibble: 1 x 5
#>    MEAN MEDIAN    SD   MAX   MIN
#>   <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1  59.5   60.7  12.9  82.6  23.6
```

Alternativamente, hay funciones como `skimr::skim()` que nos muestran una panorámica muy útil de las variables de nuestro data frame. Corre la función en tu Consola para ver el output completo.

```
skimr::skim(gapminder)
```

Table 6.1: Data summary

Name	gapminder
Number of rows	1704

Number of columns	6
Column type frequency:	
factor	2
numeric	4
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
country	0	1	FALSE	142	Afg: 12, Alb: 12, Alg: 12, Ang: 12
continent	0	1	FALSE	5	Afr: 624, Asi: 396, Eur: 360, Ame: 300

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
year	0	1	1979.50	17.27	1952.00	1965.75	1979.50	1993.25	2007.0	
lifeExp	0	1	59.47	12.92	23.60	48.20	60.71	70.85	82.6	
pop	0	1	29601212.32	106157896.74	60011.00	2793664.00	7023595.50	19585221.75	1318683096.0	
gdpPercap	0	1	7215.33	9857.45	241.17	1202.06	3531.85	9325.46	113523.1	

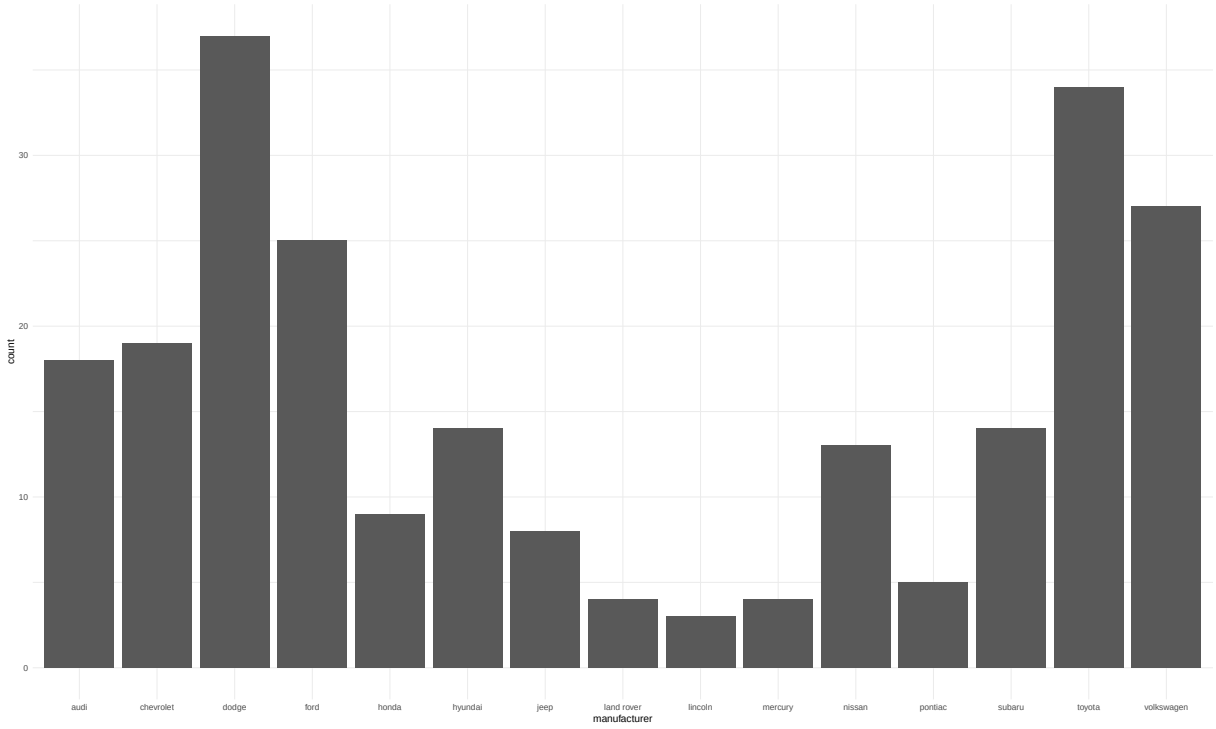
Ejercicios

Variables individuales

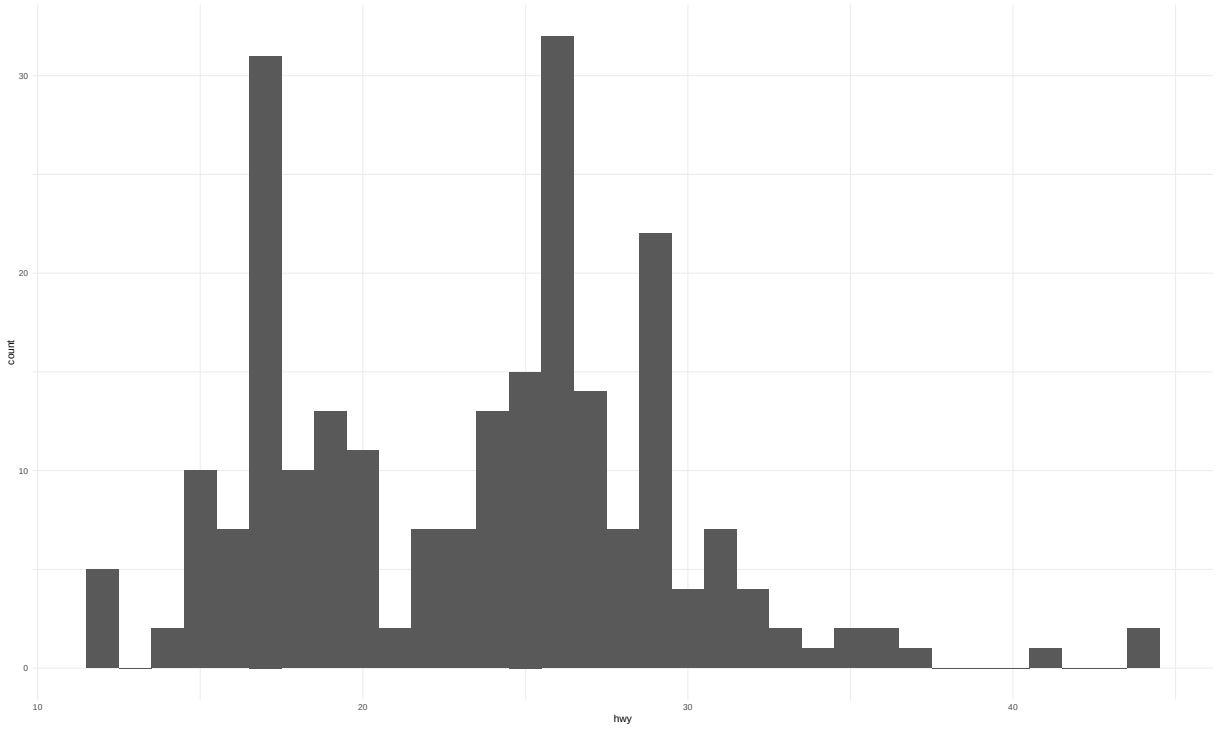
Usando el DF `mpg`, visualiza la distribución de las variables `manufacturer`, y `hwy`. Fíjate que la primera es categórica, y la segunda continua.

> **Pista**
>
> Vas a tener que elegir entre `geom_bar()` y `geom_histogram()`.
> Puedes ver que pasa si usas el parámetro `binwidth = 1` en `geom_histogram()`.

Usamos `geom_bar()` para visualizar variables discretas:



Para variables continuas, `geom_histogram()`:



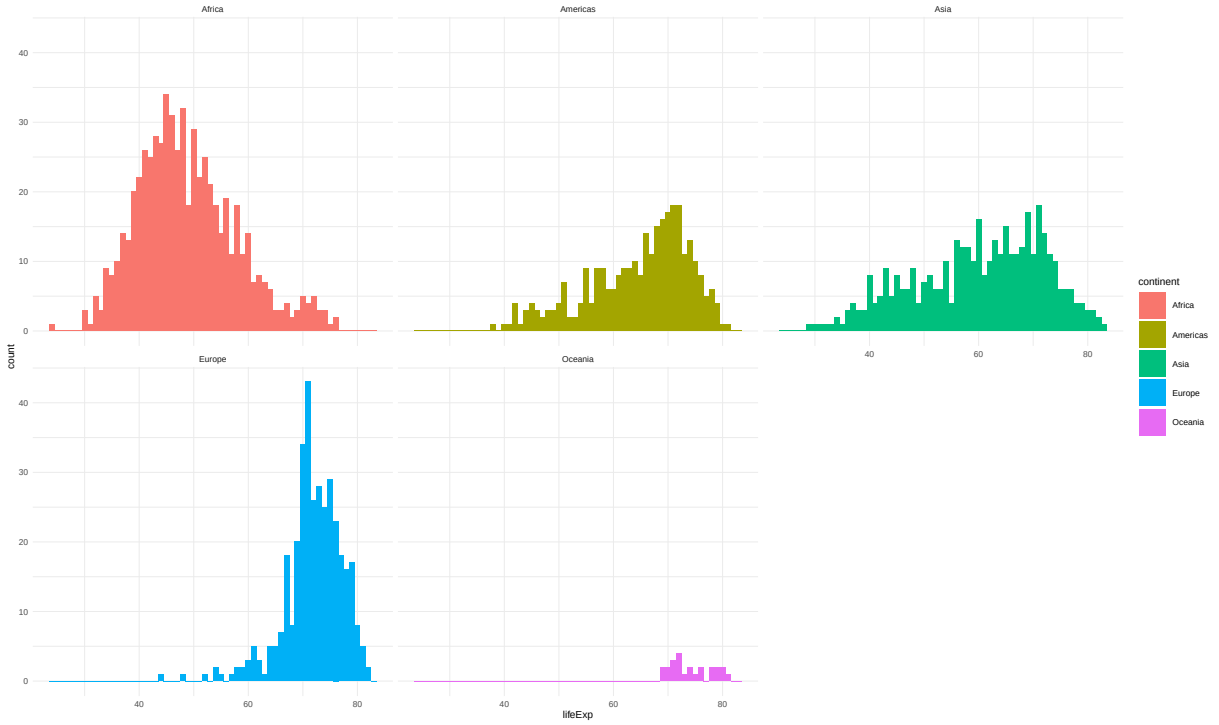
Ejercicios avanzados

Usando como base éste código:

```
ggplot(gapminder, aes(lifeExp, fill = continent)) +
  geom_histogram(binwidth = 1)
```

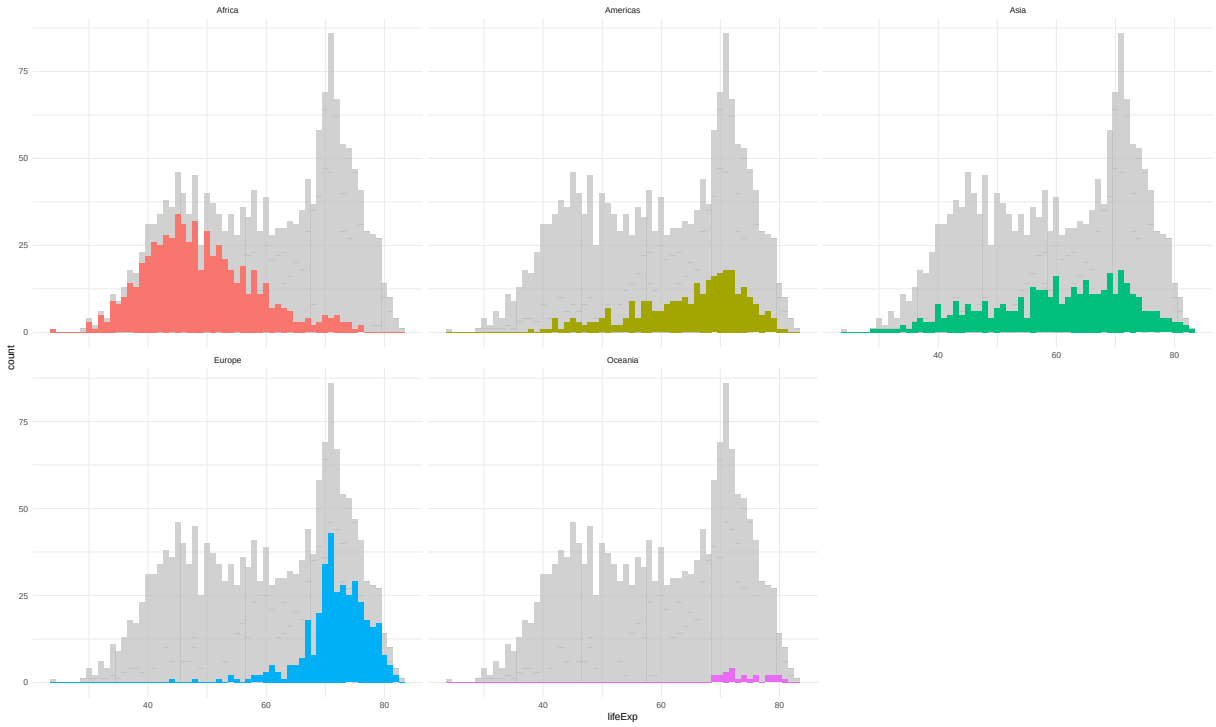
¿Podrías replicar la visualización de abajo? Queremos mostrar un histograma por continente.

> **Lo mejor es dividir el proceso en varios pasos**
>
> 1) Empieza con el histograma de arriba. 2) recuerda que puedes usar el parámetro `fill` (dentro de aes), para asignar un color de relleno por nivel de una variable categórica. 3) Finalmente, usando facetas podrás crear una gráfica para cada nivel de la variable categórica facet_wrap()!



¿Como podemos añadir el histograma general para poder entender donde se ubica cada continente?

> **La solución está en el capítulo 3**
>
> El paquete gghighlight es justo lo que necesitas



También queremos ver los descriptivos por continente, ordenados por el promedio:

```
#> # A tibble: 5 x 6
#>   continent  MEAN MEDIAN    SD   MAX   MIN
#>   <fct>     <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1 Africa     48.9   47.8  9.15  76.4  23.6
#> 2 Asia       60.1   61.8 11.9   82.6  28.8
#> 3 Americas   64.7   67.0  9.35  80.7  37.6
#> 4 Europe     71.9   72.2  5.43  81.8  43.6
#> 5 Oceania    74.3   73.7  3.80  81.2  69.1
```

6.1.3 Visualizando datasets completos

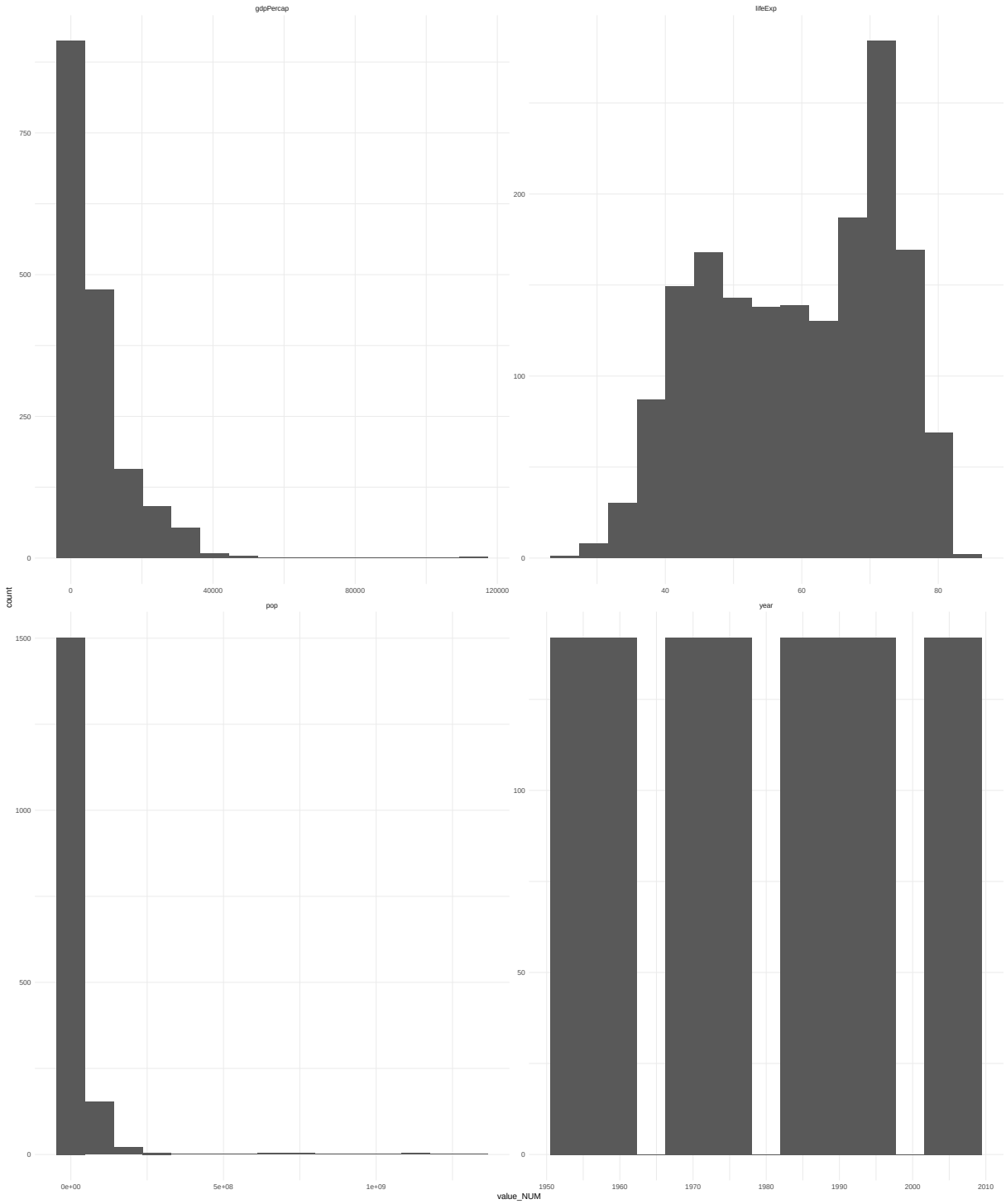
Cuando nos llega una nueva base de datos, una de las primeras cosas que haremos será familiarizarnos con los datos. Cómo se distribuyen, cual es la relación entre distintas variables, etc.

Convertimos la base a formato largo para poder filtrar los valores perdidos (i.e. 999), y crear una columna donde introducir únicamente valores numéricos:

```
d <- gapminder |>
  pivot_longer(everything(), values_transform = list(value = as.character)) |>
  filter(value != 999) |> # Si existiera algún código para missing values, filtrar
  mutate(value_NUM = as.numeric(value))
```

Visualiza las variables numéricas usando la variables `value_NUM` que hemos creado:

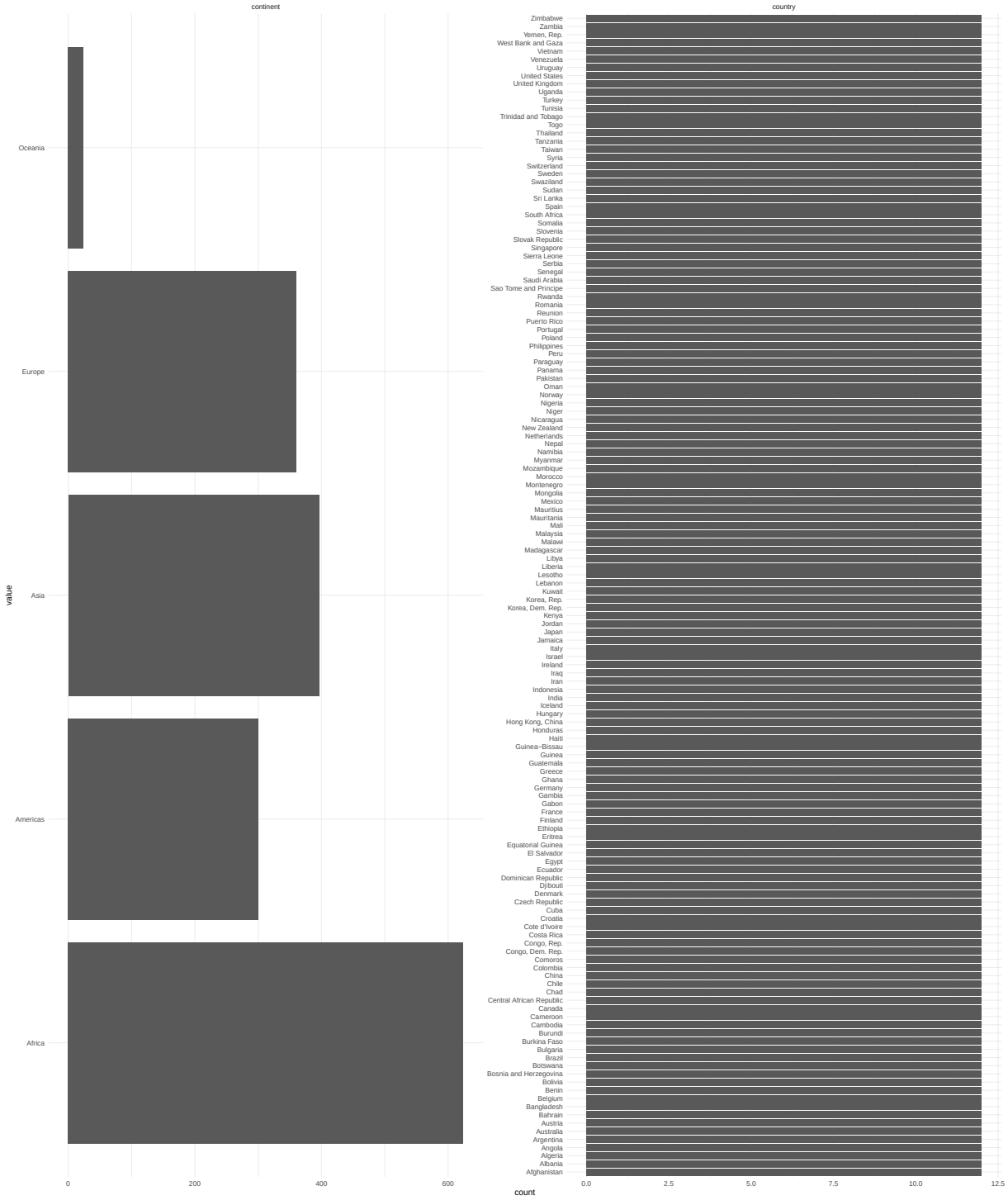
```
d |>
  drop_na(value_NUM) |>
  ggplot(aes(value_NUM)) +
    facet_wrap(~ name, scales = "free") +
    geom_histogram(bins = 15) #+ scale_x_log10()
```



Visualiza variables no numéricas. Para ello, nos quedamos solo con aquellas filas donde

`value_NUM` es una valor perdido. Esto es porque cuando antes hicimos `mutate(value_NUM = as.numeric(value))`, aquellos valores de `value` que no se pudieron convertir a numérico, quedaron como `NA` en `value_NUM`:

```
d |>
  drop_na(value) |>
  filter(is.na(value_NUM)) |>
  ggplot(aes(value)) +
    facet_wrap(~ name, scales = "free") +
    geom_bar() +
    coord_flip()
```



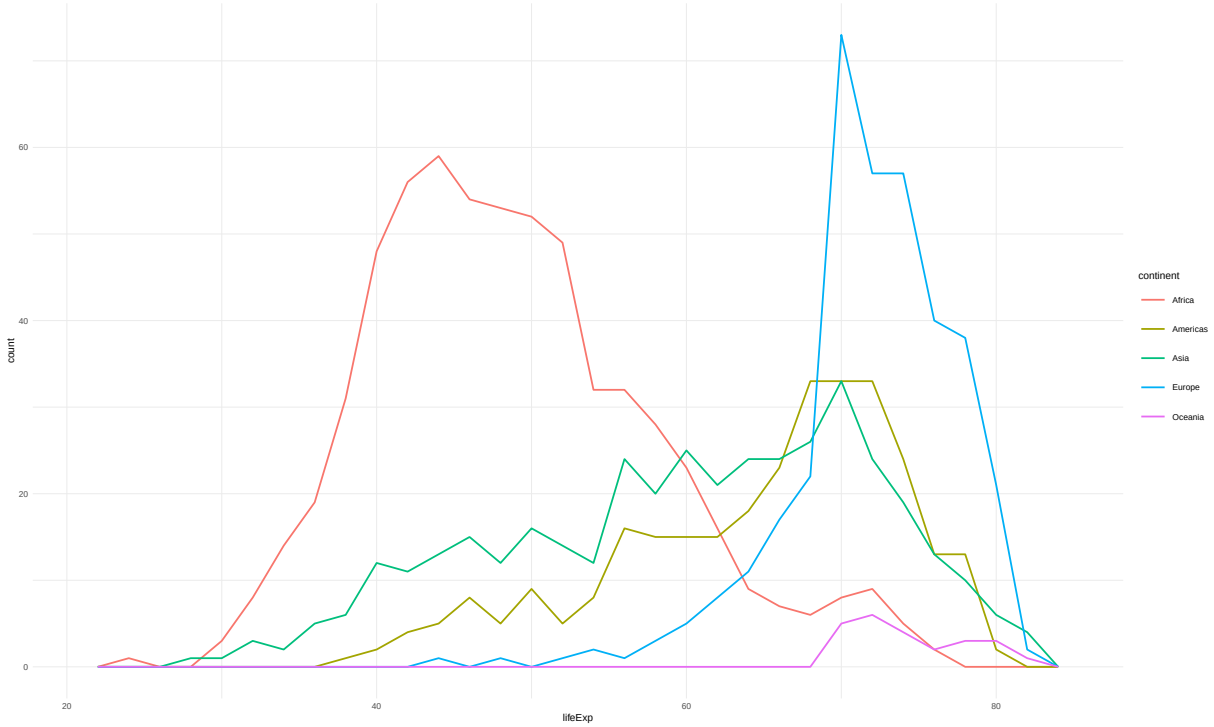
Hay algunos paquetes que ayudan a la exploración inicial de bases de datos, localización de datos perdidos, etc. Por ejemplo {inspectdf}.

6.2 Covariación

6.2.1 Variable categórica y continua

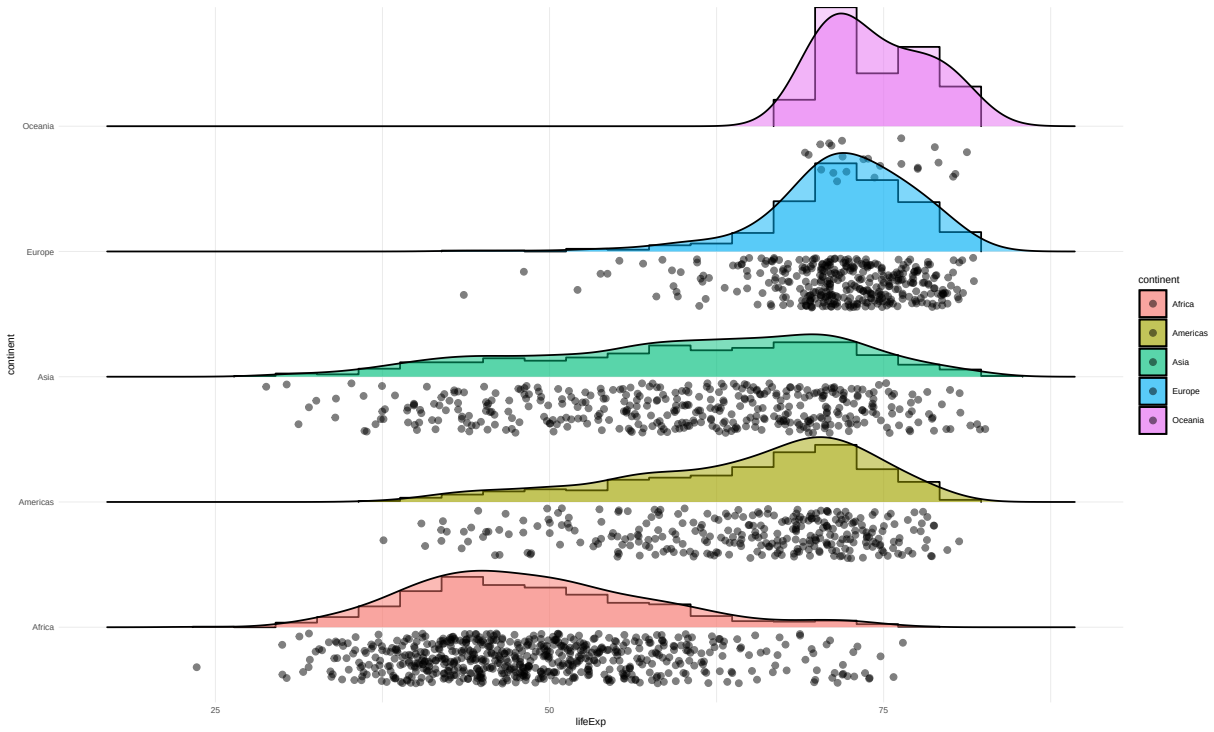
Podemos contar el numero de elementos por nivel de la variable o ver densidad, etc. Esto funciona bien si tenemos pocos niveles de la variable categórica.

```
ggplot(gapminder, aes(lifeExp, colour = continent)) +
  geom_freqpoly(binwidth = 2)
```



Podemos usar geom_density_ridges() para combinar puntos con distribuciones:

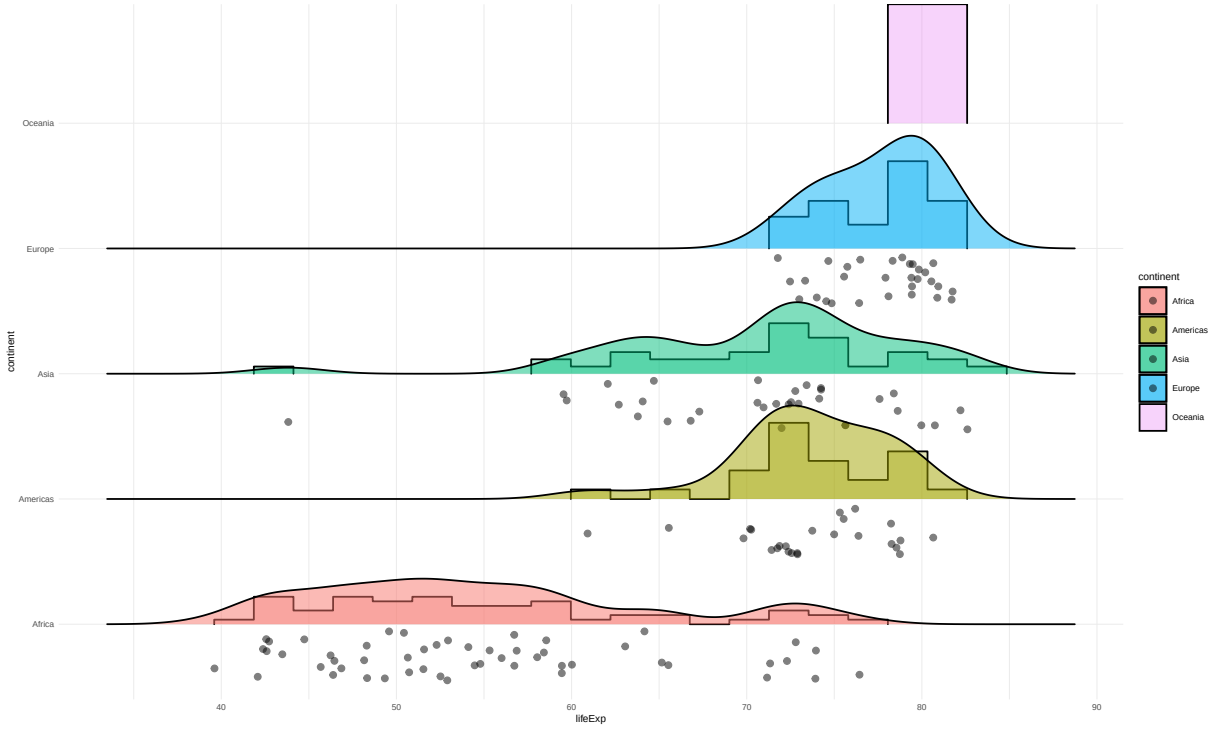
```
ggplot(gapminder, aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE,
    alpha = .3
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.5,
    scale = 0.9
  )
```



¿Qué estamos viendo exactamente arriba? Hay un punto por cada país, y por cada año, lo que da lugar a algo difícil de interpretar.

Podemos ver los datos únicamente del último año:

```
gapminder |> filter(year == max(year)) |>

  ggplot(aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE,
    alpha = .3
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.5,
    scale = 0.9
  )
```



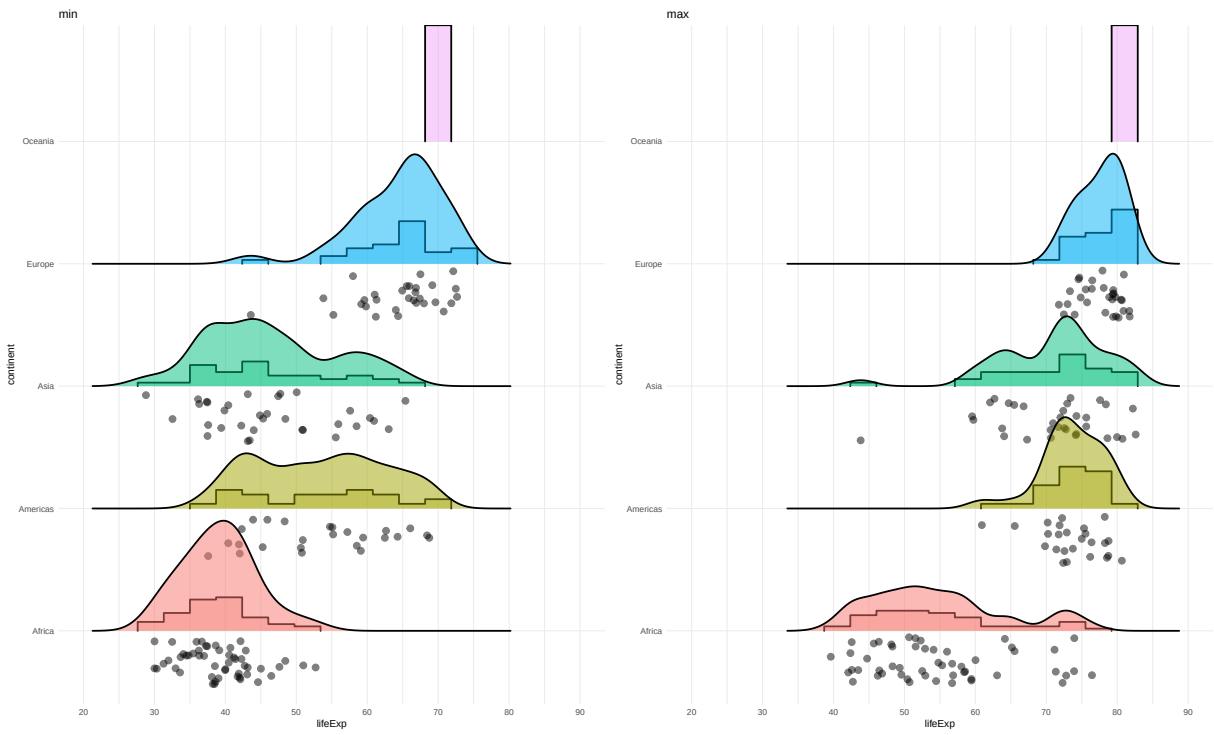
6.2.2 Ejercicio avanzado - Introducción

En este ejercicio vamos a intentar mostrar la como la distribución de esperanza de vida ha cambiado a lo largo del tiempo. Para ello, usando la base `gapminder`, compararemos los valores máximos y mínimos.

Empezamos creando dos gráficos. En cada uno filtramos por el año deseado (e.g. `filter(year == min(year))`). Fíjate que usamos `scale_x_continuous(n.breaks = 10, limits = c(20, 90))` para que ambas gráficas compartan la misma escala en el eje x:

```
A = ggplot(gapminder |> filter(year == min(year)),
           aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE,
    alpha = .3
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.5,
    scale = 0.9
```

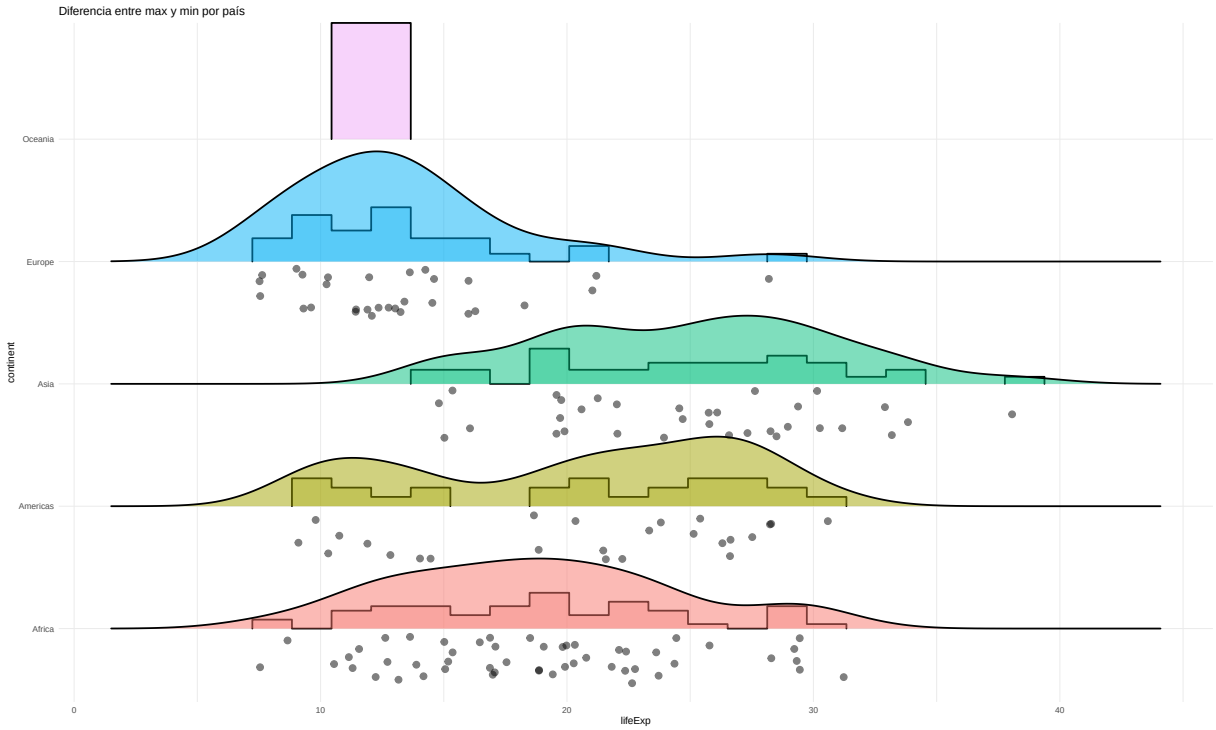
```
  ) +
  theme(legend.position = "none") +
  scale_x_continuous(n.breaks = 10, limits = c(20, 90)) +
  ggtitle("min")

B = ggplot(gapminder |> filter(year == max(year)),
           aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE,
    alpha = .3
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.5,
    scale = 0.9
  ) +
  theme(legend.position = "none") +
  scale_x_continuous(n.breaks = 10, limits = c(20, 90)) +
  ggtitle("max")

cowplot::plot_grid(A, B)
```



Para visualizar la diferencia entre los valores máximos y mínimos, podemos calcular primero

cuanto ha cambiado la esperanza de vida en cada país de cada continente, y crear una gráfica con esa variable:

```
# Cálculo de la diferencia entre el máximo y mínimo de lifeExp para cada country.
  # Incluimos continent en group_by() para poder usar esa variable en la gráfica
DF_gapminder_max_min = gapminder |>
  group_by(continent, country) |>
  summarise(lifeExp = max(lifeExp) - min(lifeExp))

ggplot(DF_gapminder_max_min, aes(lifeExp, continent, fill = continent)) +
  ggridges::geom_density_ridges(
    stat = "binline",
    bins = 20,
    scale = 0.95,
    draw_baseline = FALSE,
    alpha = .3
  ) +
  ggridges::geom_density_ridges(
    jittered_points = TRUE,
    position = "raincloud",
    alpha = 0.5,
    scale = 0.9
  ) +
  theme(legend.position = "none") +
  ggtitle("Diferencia entre max y min por país")
```

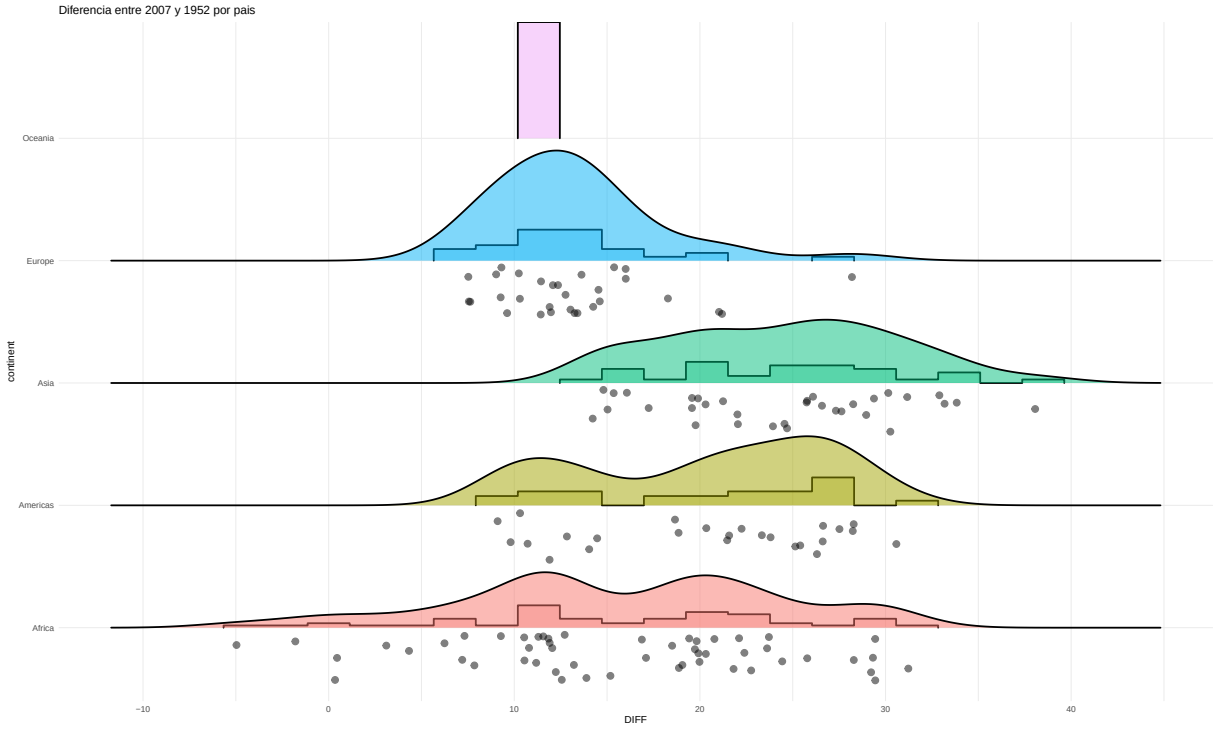


6.2.3 Ejercicio

Arriba estamos restando la esperanza de vida máxima y mínima de cada país. Sabemos que la esperanza de vida ha mejorado con el tiempo, por lo que asumíamos que era equivalente a comparar los datos más antiguos y los más nuevos.

Pero lo que realmente queremos ver es la diferencia entre 2007 y 1952 ¿Podrías rehacer el cálculo para mostrar la diferencia entre 2007 y 1952?

> **Pista**
>
> 1. Crear un DF para cada 2007 y otro para 1952, renombrando la variable lifeExp (e.g. max_lifeExp y min_lifeExp, respectivamente)2. Usando la función `full_join()`, juntamos ambas bases (tendrás que usar el parámetro `by = c("country", "continent")`). 3. Con `mutate()` calculamos la diferencia.



6.2.4 Dos variables categóricas

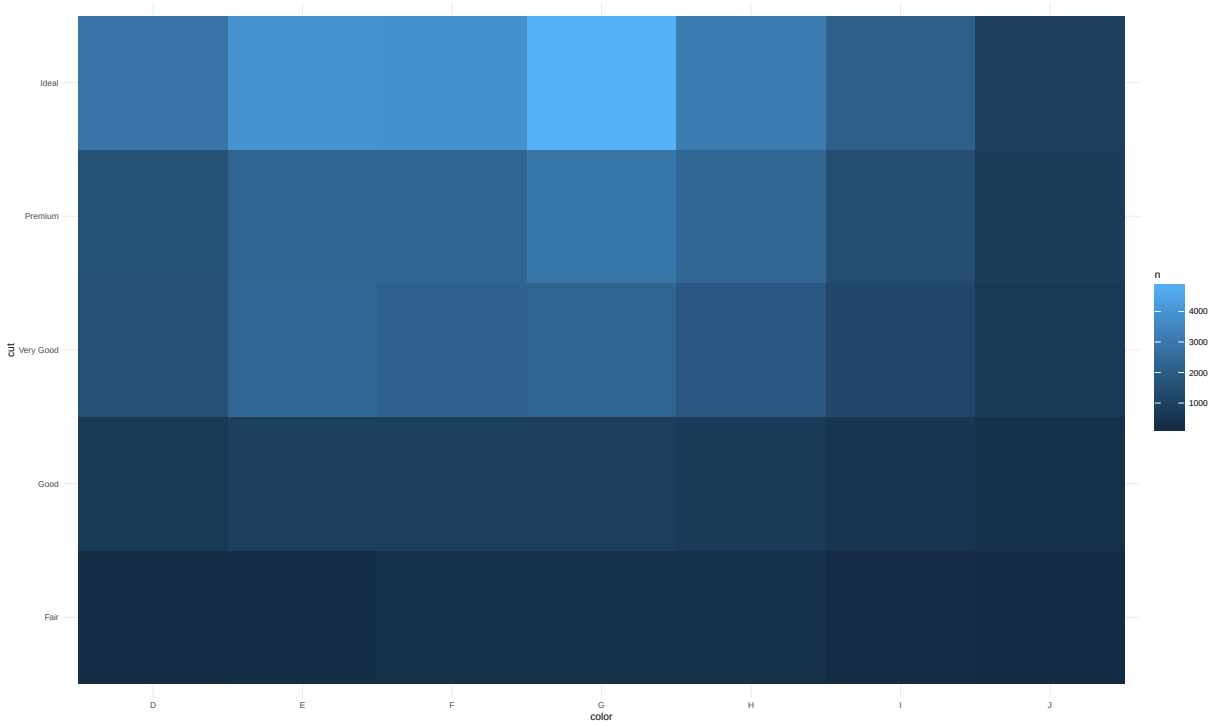
Podemos contar el número de valores para los niveles de dos variables categóricas con `count()`:

```
diamonds |>
  count(color, cut)
#> # A tibble: 35 x 3
#>   color cut           n
#>   <ord> <ord>     <int>
#> 1 D     Fair        163
#> 2 D     Good        662
```

```
#> 3 D     Very Good  1513
#> 4 D     Premium    1603
#> 5 D     Ideal      2834
#> 6 E     Fair        224
#> # i 29 more rows
```

Una manera de visualizar esto es con `geom_tile()`:

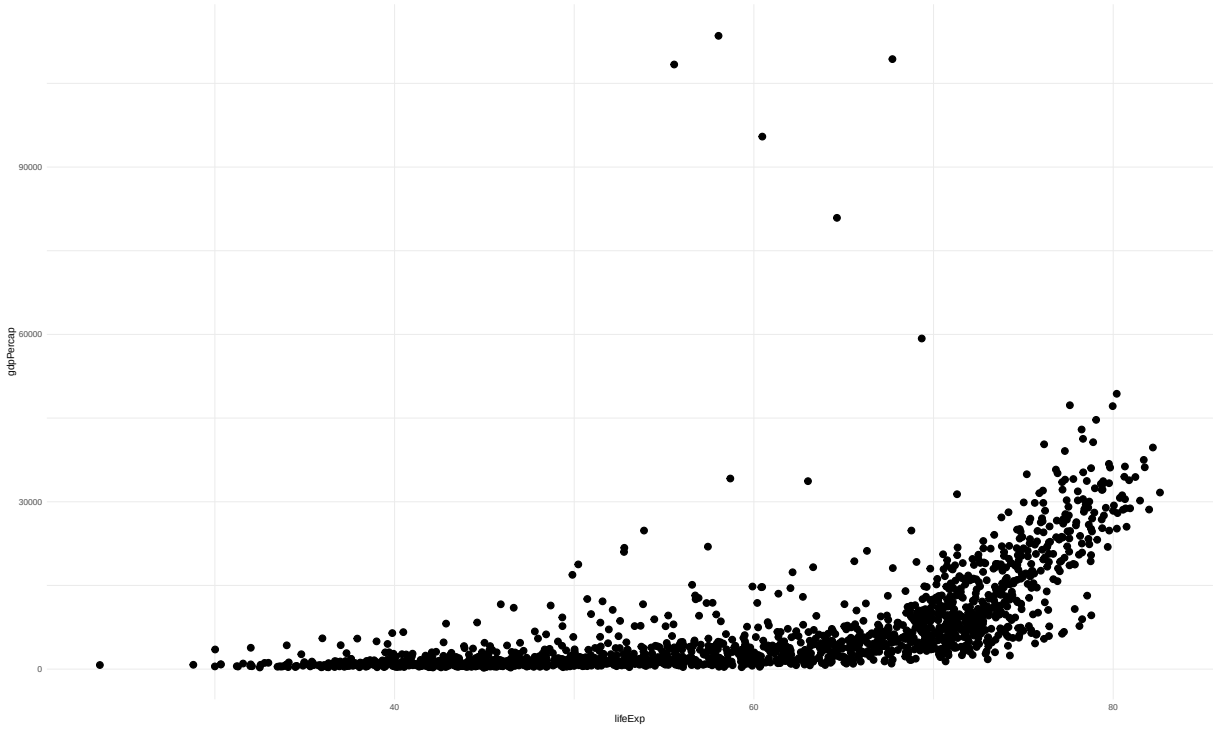
```
diamonds |>
  count(color, cut) |>
  ggplot(aes(color, cut, fill = n)) +
    geom_tile()
```



6.2.5 Dos variables continuas

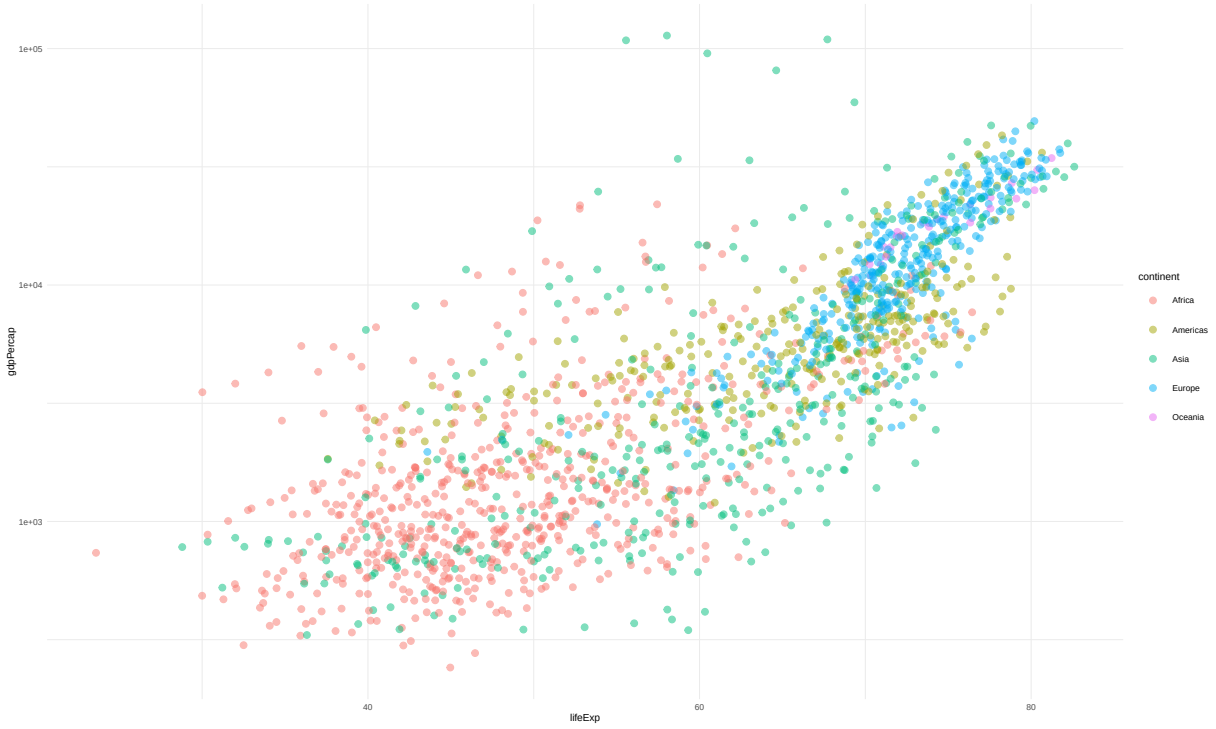
Una herramienta esencial para ver como covarían dos variables continuas es un scatterplot. Usaremos `geom_point()`:

```
ggplot(gapminder, aes(lifeExp, gdpPercap)) +
  geom_point()
```



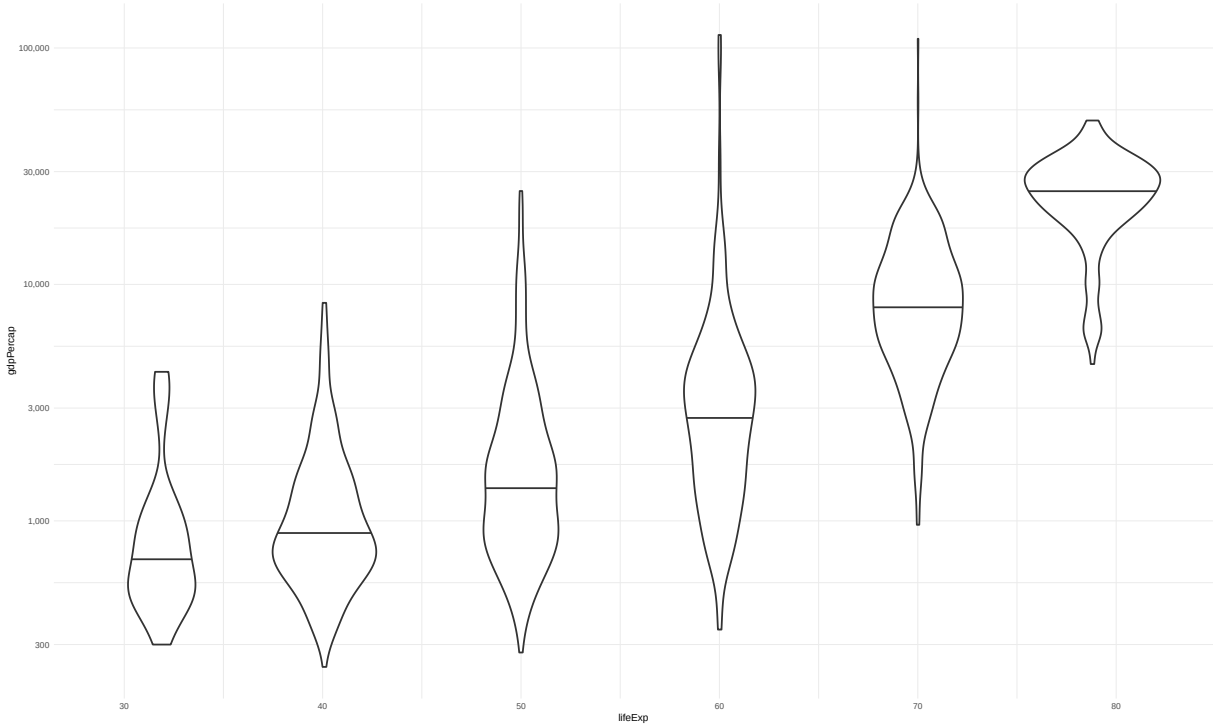
Si añadimos color y cambiamos a escala logarítmica, podemos hacernos una mejor idea:

```
ggplot(gapminder, aes(lifeExp, gdpPercap, color = continent)) +
  geom_point(alpha = 1 / 2) +
  scale_y_log10()
```



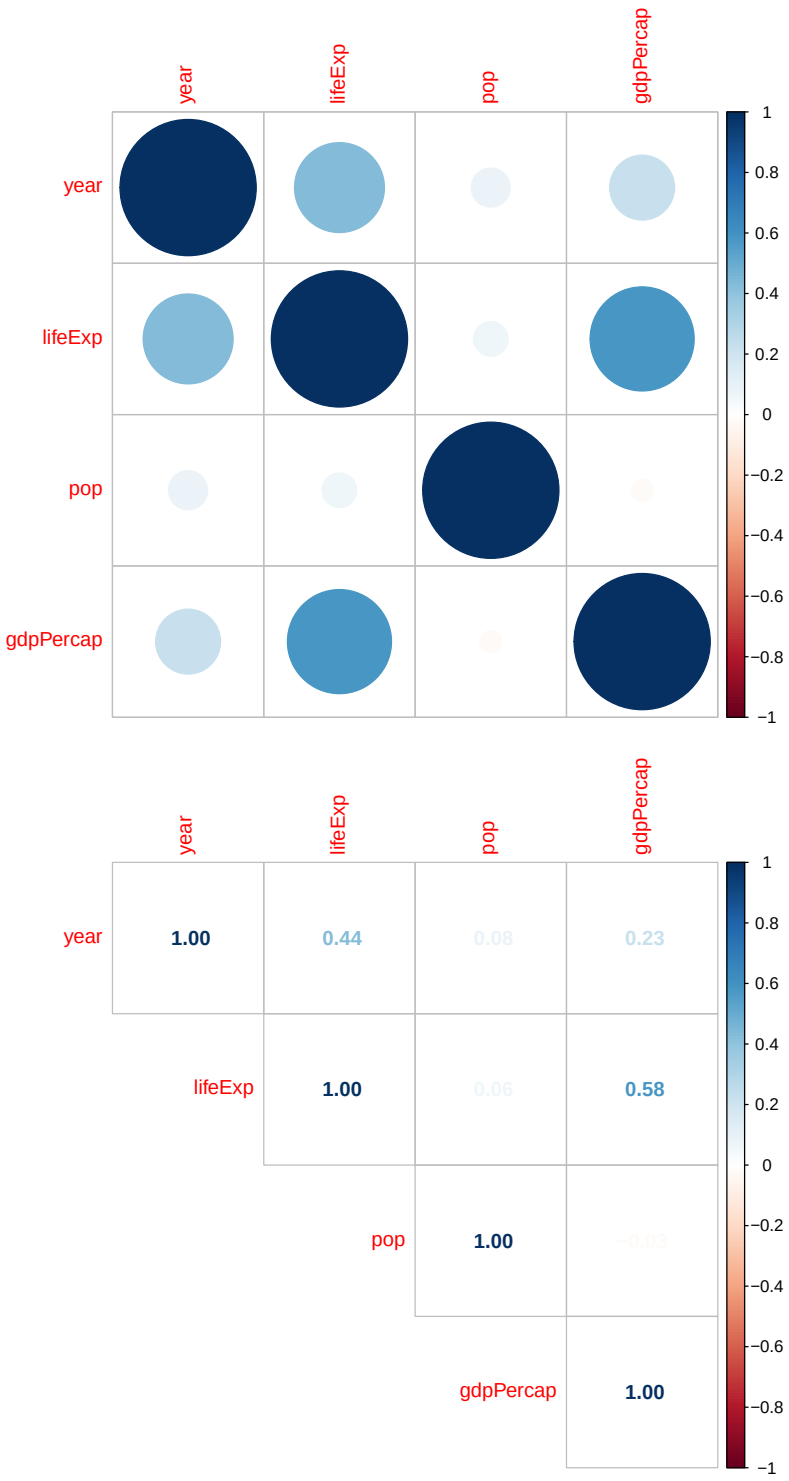
Con `geom_violin()` podemos hacernos una idea rápida de las distribuciones. Usamos `group = cut_width(lifeExp, 10)` para dividir la variable continua `lifeExp` en chunks de 10 unidades:

```
ggplot(gapminder, aes(lifeExp, gdpPercap)) +
  geom_violin(draw_quantiles = .5, aes(group = cut_width(lifeExp, 10))) +
  scale_y_log10(labels = scales::comma, n.breaks = 6)
```



Con `corrplot` podemos visualizar las correlaciones entre variables numéricas de nuestra base de datos.

```
corrplot(cor(gapminder |> select(where(is.numeric))), method = "circle")
corrplot(cor(gapminder |> select(where(is.numeric))), method = "number", type = "upper")
```



6.2.6 Ejercicio covariación 2

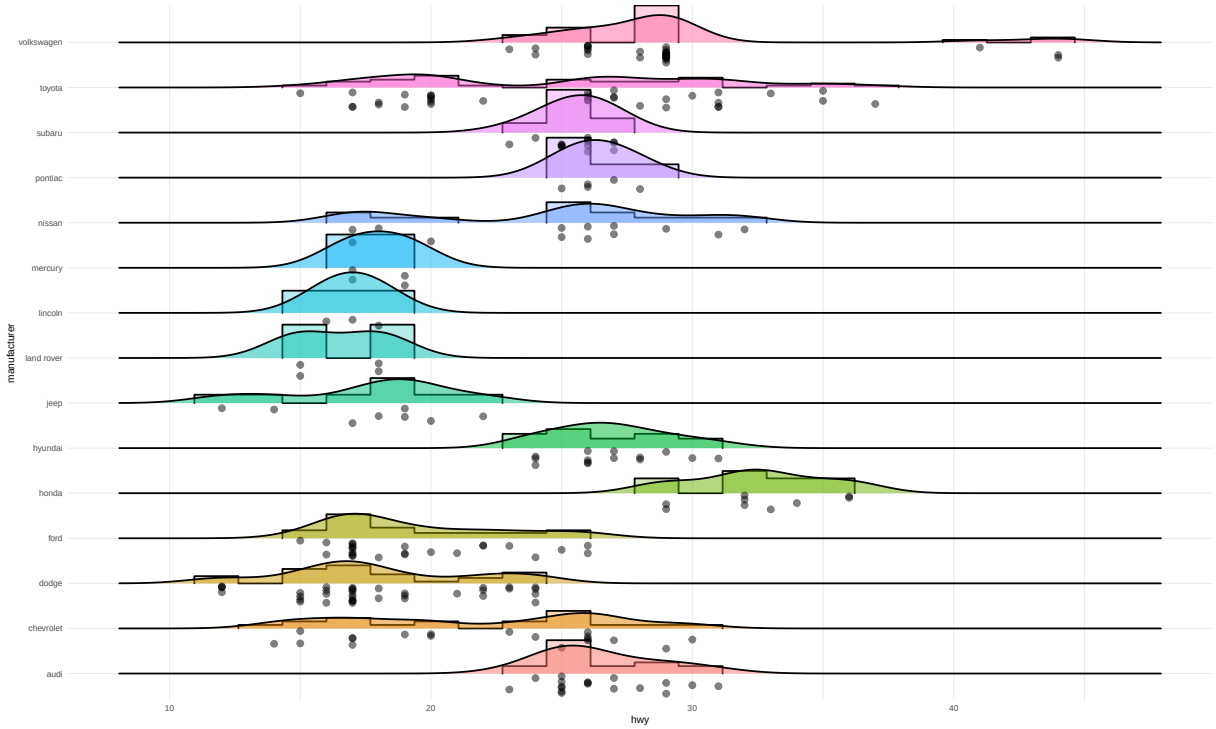
Usando el DF `mpg`, visualiza la covariación entre:

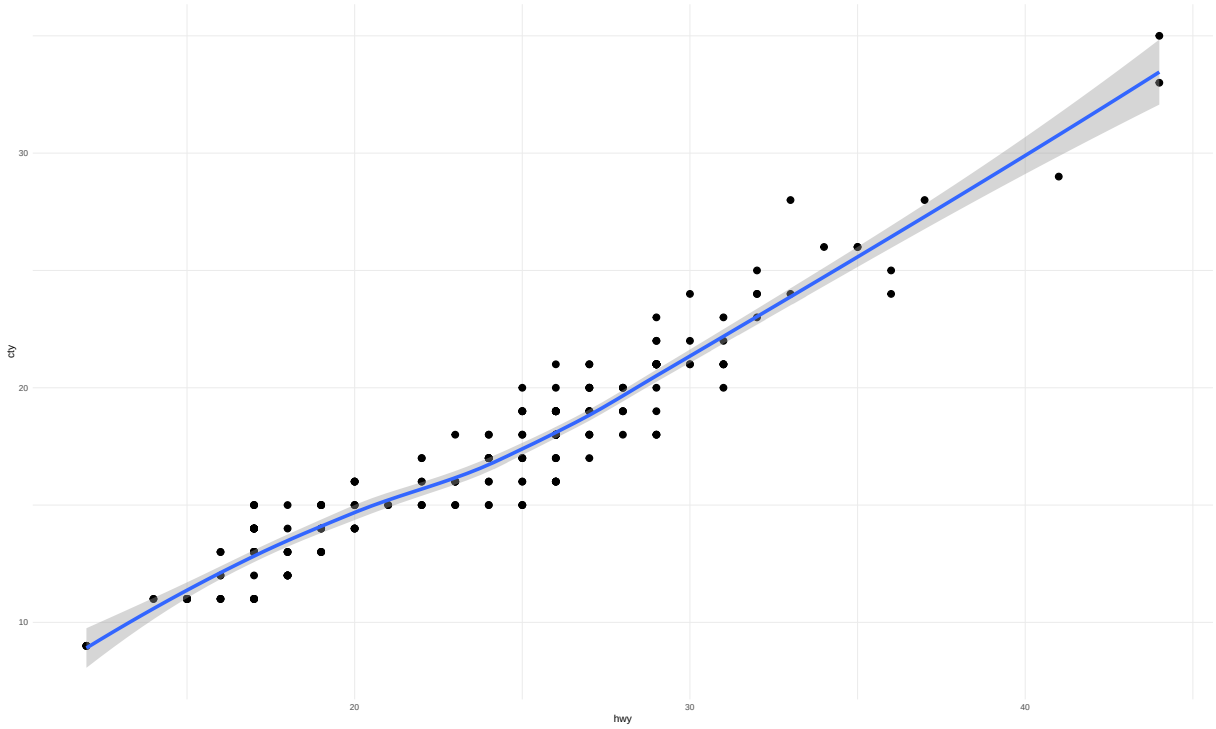
- `manufacturer` y `hwy`
- `class` y `hwy`

- `hwy` y `cty`

> **Visualizando una variable categórica y una continua**
>
> geom_density_ridges()!

> **Visualizando pares de variables continuas!**
>
> geom_smooth()!





6.3 Ejercicios finales

6.3.1 Ejercicio exploración base nueva

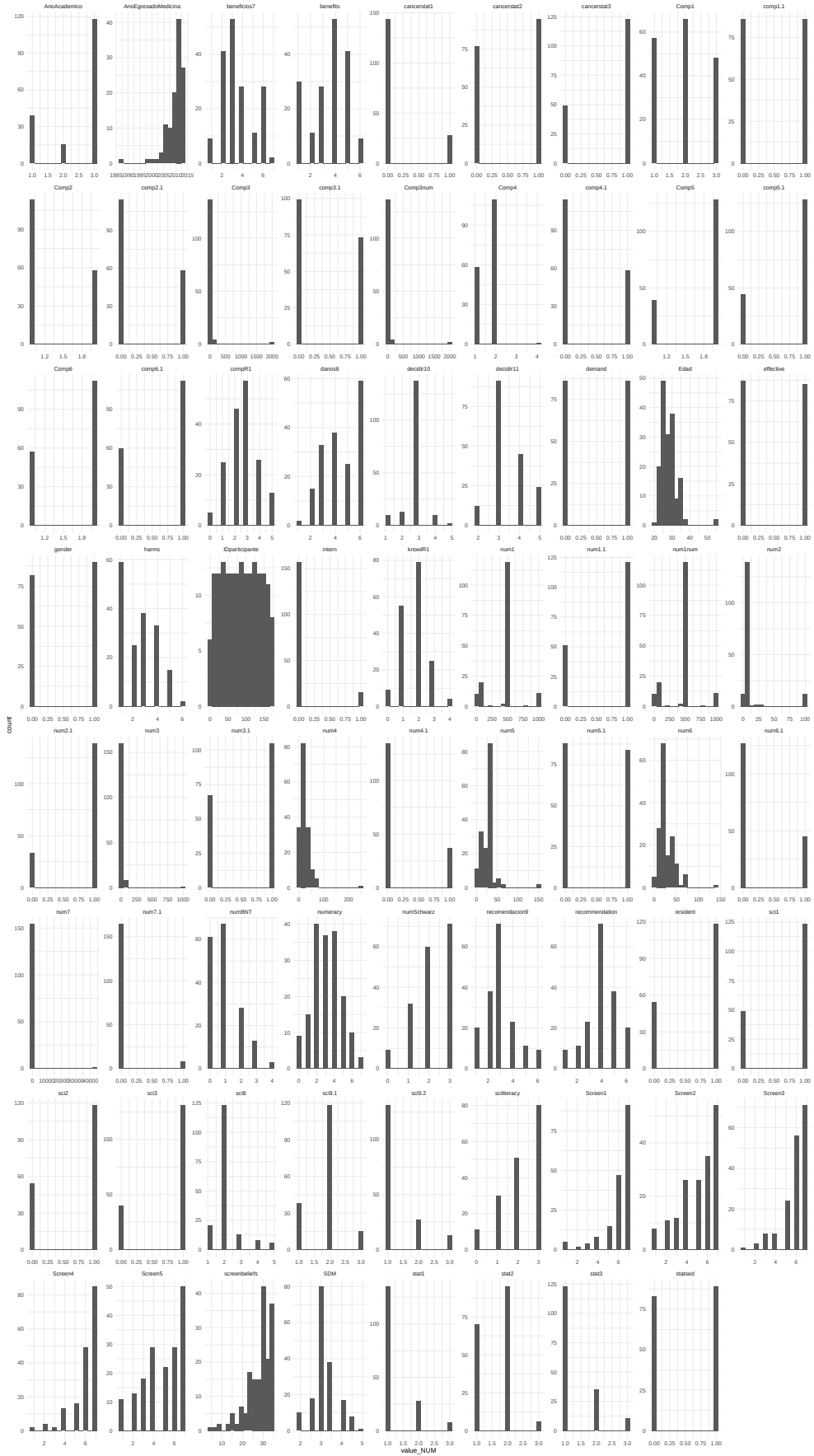
Usando la base del paper Cancer Screening Risk Literacy of Physicians in Training, haz un primer análisis exploratorio que incluya:

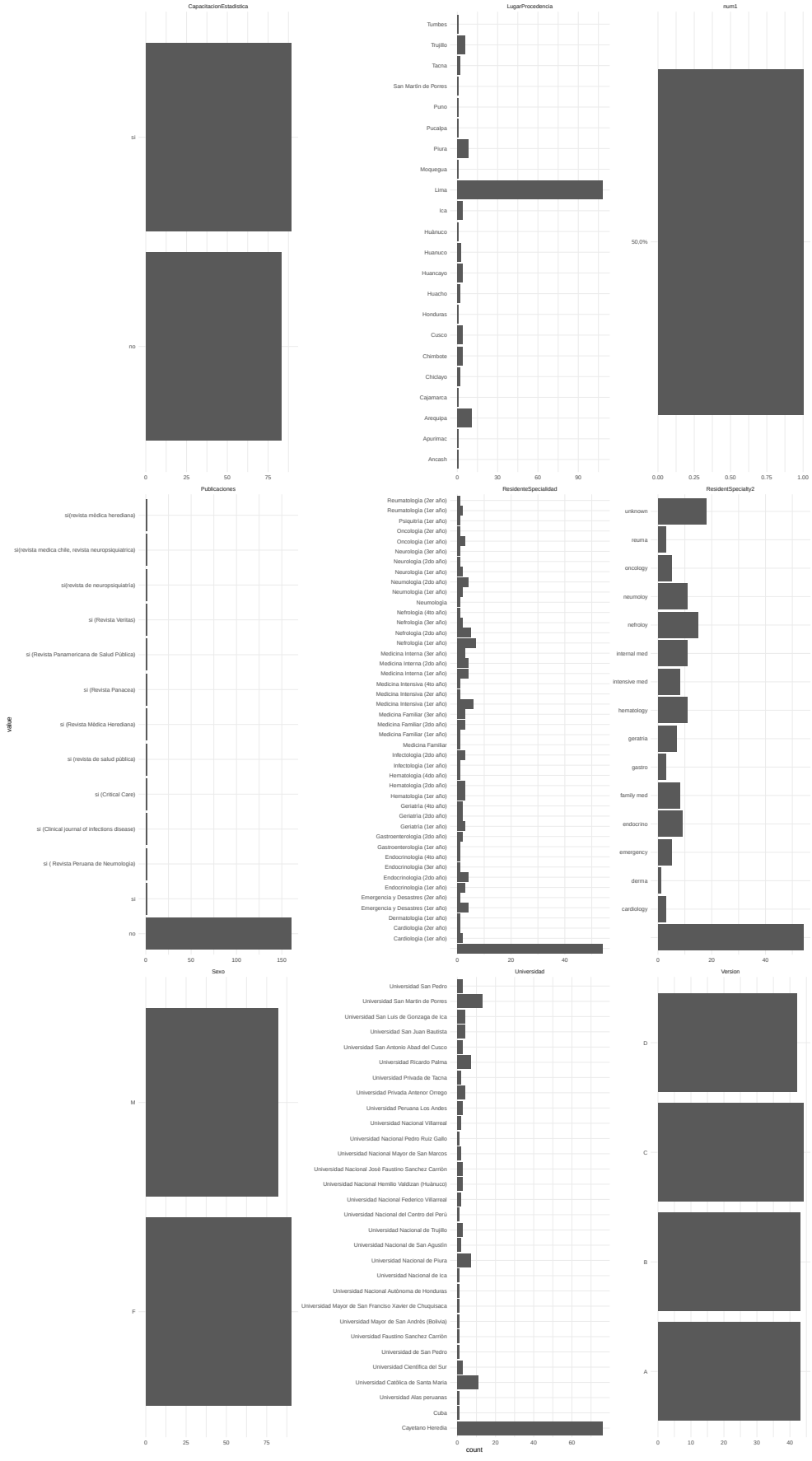
- histogramas de todas las variables numéricas y no-numéricas
- scatterplots de la relación entre comprensión y numeracy, y entre comprensión y screenbeliefs

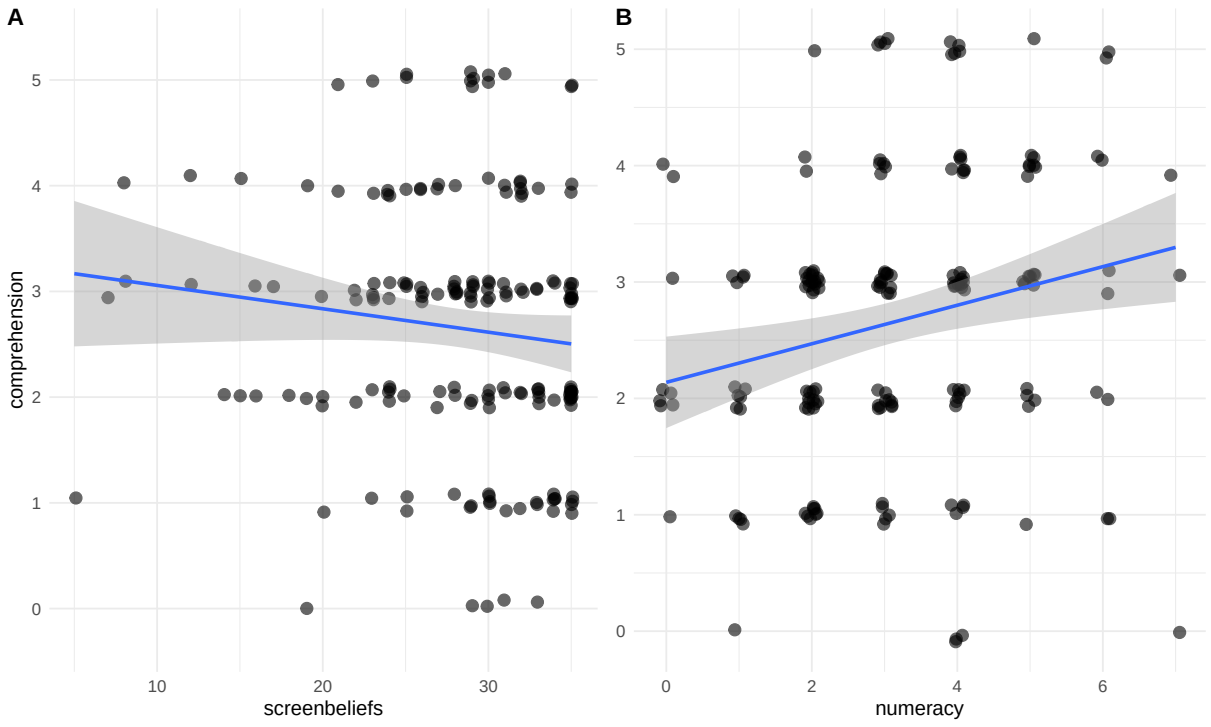
Puedes ir al enlace anterior y descargar el archivo `Cancer screening risk literacy R1.sav` en la carpeta `Data and results`, o directamente usar el código de abajo.

> Como visualizar todas las variables
>
> Ver el apartado visualizando-datasets-completos en este mismo capítulo

```
# Usamos haven::read_sav() para leer los archivos .sav
DF_dafina = haven::read_sav(here::here("data/files/Dafina", "Cancer screening risk literacy
```


AnoAcademico
AnoEgresadoMedicina
beneficios7
benefits
cancerstat1
cancerstat2
cancerstat3
Comp1
comp1.1
Comp2
comp2.1
Comp3
comp3.1
Comp3num
Comp4
comp4.1
Comp5
comp5.1
Comp6
comp6.1
compR1
danos8
decidir10
decidir11
demand
Edad
effective
gender
harms
IDparticipante
intern
knowR1
num1
num1.1
num1num
num2
num2.1
num3
num3.1
num4
num4.1
num5
num5.1
num6
num6.1
num7
num7.1
numBNT
numeracy
numSchwarz
recomendacion9
recommendation
resident
sci1
sci2
sci3
sci8
sci9.1
sci9.2
sciliteracy
Screen1
Screen2
Screen3
Screen4
Screen5
screenbeliefs
SDM
stat1
stat2
stat3
statsed
count
value_NUM


CapacitacionEstadistica
si
no
0
25
50
75
LugarProcedencia
Tumbes
Trujillo
Tacna
San Martín de Porres
Puno
Pucalpa
Piura
Moquegua
Lima
Ica
Huánuco
Huanuco
Huancayo
Huacho
Honduras
Cusco
Chimbote
Chiclayo
Cajamarca
Arequipa
Apurimac
Ancash
0
30
60
90
num1
50,0%
0.00
0.25
0.50
0.75
1.00
Publicaciones
value
si(revista mèdica herediana)
si(revista medica chile, revista neuropsiquiatrica)
si(revista de neuropsiquiatria)
si (Revista Veritas)
si (Revista Panamericana de Salud Pública)
si (Revista Panacea)
si (Revista Médica Herediana)
si (revista de salud pública)
si (Critical Care)
si (Clinical journal of infections disease)
si (Revista Peruana de Neumologia)
si
no
0
50
100
150
ResidenteSpecialidad
Reumatología (2er año)
Reumatología (1er año)
Psiquiatría (1er año)
Oncología (2er año)
Oncología (1er año)
Neurología (3er año)
Neurología (2do año)
Neurología (1er año)
Neumología (2do año)
Neumología (1er año)
Neumología
Nefrología (4to año)
Nefrología (3er año)
Nefrología (2do año)
Nefrología (1er año)
Medicina Interna (3er año)
Medicina Interna (2do año)
Medicina Interna (1er año)
Medicina Intensiva (4to año)
Medicina Intensiva (2er año)
Medicina Intensiva (1er año)
Medicina Familiar (3er año)
Medicina Familiar (2do año)
Medicina Familiar (1er año)
Medicina Familiar
Infectología (2do año)
Infectología (1er año)
Hematología (4do año)
Hematología (2do año)
Hematología (1er año)
Geriatría (4to año)
Geriatría (2do año)
Geriatría (1er año)
Gastroenterología (2do año)
Gastroenterología (1er año)
Endocrinología (4to año)
Endocrinología (3er año)
Endocrinología (2do año)
Endocrinología (1er año)
Emergencia y Desastres (2er año)
Emergencia y Desastres (1er año)
Dermatología (1er año)
Cardiología (2er año)
Cardiología (1er año)
0
20
40
ResidentSpecialty2
unknown
reuma
oncology
neumoloy
nefroloy
internal med
intensive med
hematology
geratria
gastro
family med
endocrino
emergency
derma
cardiology
0
20
40
Sexo
M
F
0
25
50
75
Universidad
Universidad San Pedro
Universidad San Martín de Porres
Universidad San Luis de Gonzaga de Ica
Universidad San Juan Bautista
Universidad San Antonio Abad del Cusco
Universidad Ricardo Palma
Universidad Privada de Tacna
Universidad Privada Antenor Orrego
Universidad Peruana Los Andes
Universidad Nacional Villarreal
Universidad Nacional Pedro Ruiz Gallo
Universidad Nacional Mayor de San Marcos
Universidad Nacional José Faustino Sanchez Carrión
Universidad Nacional Hermilio Valdizan (Huánuco)
Universidad Nacional Federico Villarreal
Universidad Nacional del Centro del Perú
Universidad Nacional de Trujillo
Universidad Nacional de San Agustín
Universidad Nacional de Piura
Universidad Nacional de Ica
Universidad Nacional Autónoma de Honduras
Universidad Mayor de San Francisco Xavier de Chuquisaca
Universidad Mayor de San Andrés (Bolivia)
Universidad Faustino Sanchez Carrión
Universidad de San Pedro
Universidad Científica del Sur
Universidad Católica de Santa Maria
Universidad Alas peruanas
Cuba
Cayetano Heredia
0
20
40
60
count
Version
D
C
B
A
0
10
20
30
40



Bibliografía

Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc. https://r4ds.had.co.nz/

7 Análisis de datos inferencial

Paquetes para este capítulo

```
if (!require('afex')) install.packages('afex'); library('afex')
if (!require('correlation')) install.packages('correlation'); library('correlation')
if (!require('corrr')) install.packages('corrr'); library('corrr')
if (!require('cowplot')) install.packages('cowplot'); library('cowplot')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('gapminder')) install.packages('gapminder'); library('gapminder')
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
if (!require('ggridges')) install.packages('ggridges'); library('ggridges')
if (!require('gtsummary')) install.packages('gtsummary'); library('gtsummary')
if (!require('haven')) install.packages('haven'); library('haven')
if (!require('inspectdf')) install.packages('inspectdf'); library('inspectdf')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('lme4')) install.packages('lme4'); library('lme4')
if (!require('papaja')) install.packages("papaja"); library('papaja')
if (!require('parameters')) install.packages('parameters'); library('parameters')
if (!require('performance')) install.packages('performance'); library('performance')
if (!require('report')) install.packages('report'); library('report')
if (!require('sjPlot')) install.packages('sjPlot'); library('sjPlot')
if (!require('tidyr')) install.packages('tidyr'); library('tidyr')
```

7.1 Análisis de datos y reporte de resultados

R es un lenguaje creado por estadísicos que ha ido evolucionando hacia un lenguaje de programación completo. No obstante, una de sus fortalezas innegables es el análisis de datos, y el reporte de resultados. En esta sección vamos a ver de manera muy general algunas de las herramientas que tenemos a nuestra disposición.

7.1.1 Tablas

Hay numerosos paquetes para crear tablas descriptivas o para facilitar el reporte de resultados en R:

- {gtsummary}
- {stargazer}
- {papaja}
- {flextable}
- {huxtable}

Mostraremos algunos ejemplos usando gtsummary. Una ventaja interesante es que permite de manera sencilla transformar nuestra tabla a otros formatos.

7.2 Tablas descriptivos

Podemos crear tablas con los descriptivos de nuestros datos usando la función `tbl_summary()` de {gtsummary}

```
# Por defecto, usa: mediana (rango inter cuartil)
gapminder |>
  select(-country) |>
  gtsummary::tbl_summary()
```

Characteristic	**N = 1,704**[1]
continent	
Africa	624 (37%)
Americas	300 (18%)
Asia	396 (23%)
Europe	360 (21%)
Oceania	24 (1.4%)
year	1,980 (1,965, 1,995)
lifeExp	61 (48, 71)
pop	7,023,596 (2,792,776, 19,593,661)
gdpPercap	3,532 (1,202, 9,326)

[1]n (%); Median (Q1, Q3)

Usando el parámetro `by` podemos crear columnas para cada valor de una variable:

```r
# Por continente
gapminder |>
  select(-country) |>
  gtsummary::tbl_summary(by = continent)
```

Characteristic	**Africa** N = 624[1]	**Americas** N = 300[1]	**Asi**
year	1,980 (1,965, 1,995)	1,980 (1,965, 1,995)	1,980
lifeExp	48 (42, 54)	67 (58, 72)	6
pop	4,579,311 (1,341,536, 10,822,313)	6,227,510 (2,959,572, 18,555,489)	14,530,831 (3
gdpPercap	1,192 (761, 2,378)	5,466 (3,426, 7,853)	2,647

[1]Median (Q1, Q3)

El parámetro `statistic` nos permite controlar que estadísticos mostrar en función del tipo de variable. En el caso de abajo, usaremos "promedio (desviación estándar)" para las variables continuas y "número de observaciones (% del total)" para las variables categóricas.

```r
gapminder |>
  select(-country) |>
  gtsummary::tbl_summary(by = continent,
                         statistic = list(all_continuous() ~ "{mean} ({sd})",
                                          all_categorical() ~ "{n} ({p}%)"),
                         missing = "ifany") |>
  gtsummary::add_n() |>
  gtsummary::modify_spanning_header(c("stat_1", "stat_2", "stat_3", "stat_4", "stat_5") ~ "*
```

				Continent
Characteristic	**N**	**Africa** N = 624[1]	**Americas** N = 300[1]	**Asia** N = 396[1]
year	1,704	1,980 (17)	1,980 (17)	1,980 (17)
lifeExp	1,704	49 (9)	65 (9)	60 (12)
pop	1,704	9,916,003 (15,490,923)	24,504,795 (50,979,430)	77,038,722 (206,885,205)
gdpPercap	1,704	2,194 (2,828)	7,136 (6,397)	7,902 (14,045)

[1]Mean (SD)

Ejercicio - Descriptivos

Usando la base de datos del apartado anterior:

```r
DF_dafina = haven::read_sav(here::here("data/files/Dafina", "Cancer screening risk literacy
  select(IDparticipante, resident, screenbeliefs, compR1, numeracy) |>
  rename(comprehension = compR1)
```

Intenta reproducir la tabla de abajo. En el manual de gtsummary tienes ejemplos para todo lo que necesitarás. Busca a la función `tbl_summary()`.

> La tabla me muestra demasiado detalle, pero sólo quiero los promedios
>
> Si todas tus variables son continuas, puedes usar `type = list(everything() ~ 'continuous'),` dentro de `tbl_summary()` para forzar el tratamiento de variables con pocos niveles como continuas.

> Error al añadir valor p
>
> En la ayuda de la función: `?add_p.tbl_summary` encontrarás que puedes usar algo como: `add_p(test = everything() ~ "t.test")`

> ¿Cómo consigo poner el titulo `Resident`?
>
> Fíjate cómo usamos `modify_spanning_header()` en el ejemplo anterior. Solo tienes que adaptarlo a los datos actuales, donde tenemos únicamente dos grupos.

		Resident		
Characteristic	**N**	**0** N = 54[1]	**1** N = 118[1]	**p-value**[2]
IDparticipante	172	72 (51)	93 (48)	0.012
Screening beliefs	172	27.4 (5.0)	28.4 (6.8)	0.3
Comprehension of the evidence	172	2.91 (1.28)	2.54 (1.16)	0.077
Numeracy BNT-S	172	3.70 (1.54)	2.87 (1.56)	0.001

[1]Mean (SD)
[2]Welch Two Sample t-test

7.3 Tablas resultados inferenciales

Para tablas con los resultados de nuestros modelos estadísticos, usamos la función `tbl_regression()` de {gtsummary}

Primero preparamos los datos:

```
# Transform variables
DF_gapminder = gapminder |>
  # Log
  mutate(gdpPercap_log = log(gdpPercap),
         pop_log = log(pop)
         ) |>
  # Mean center variables so the 0 values have meaning
```

```
  mutate(year = year - mean(year, na.rm = TRUE),
         gdpPercap_log = gdpPercap_log - mean(gdpPercap_log, na.rm = TRUE),
         pop_log = pop_log - mean(pop_log, na.rm = TRUE)) |>
  # Will use only last year
  filter(year == max(year))
```

Creamos un modelo sencillo y mostramos la tabla de resultados.

```
model1 = lm(lifeExp ~ gdpPercap_log + pop_log, DF_gapminder)

table_model1 = gtsummary::tbl_regression(model1, intercept = TRUE) |>
  add_global_p() |>
  bold_labels() |>
  italicize_levels() |>
  add_glance_table(include = c("nobs", "df.residual", "r.squared", "adj.r.squared"))


table_model1
```

Characteristic	Beta	95% CI[1]	p-value
(Intercept)	63	62, 65	<0.001
gdpPercap_log	7.2	6.4, 8.1	<0.001
pop_log	0.81	0.04, 1.6	0.039
No. Obs.	142		
Residual df	139		
R^2	0.665		
Adjusted R^2	0.660		

[1]CI = Confidence Interval

7.4 Reporte de resultados

Con la función `report()` podemos ver una descripción completa de los resultados de nuestro modelo:

```
  report::report(model1)
#> We fitted a linear model (estimated using OLS) to predict lifeExp with
#> gdpPercap_log and pop_log (formula: lifeExp ~ gdpPercap_log + pop_log). The
#> model explains a statistically significant and substantial proportion of
#> variance (R2 = 0.66, F(2, 139) = 137.93, p < .001, adj. R2 = 0.66). The
#> model's intercept, corresponding to gdpPercap_log = 0 and pop_log = 0, is at
#> 63.28 (95% CI [61.98, 64.58], t(139) = 96.30, p < .001). Within this model:
#>
#>   - The effect of gdpPercap log is statistically significant and positive
```

```
#> (beta = 7.24, 95% CI [6.38, 8.11], t(139) = 16.56, p < .001; Std. beta =
#> 0.81, 95% CI [0.72, 0.91])
#>   - The effect of pop log is statistically significant and positive (beta =
#> 0.81, 95% CI [0.04, 1.58], t(139) = 2.09, p = 0.039; Std. beta = 0.10, 95%
#> CI [5.35e-03, 0.20])
#>
#> Standardized parameters were obtained by fitting the model on a standardized
#> version of the dataset. 95% Confidence Intervals (CIs) and p-values were
#> computed using a Wald t-distribution approximation.
```

7.5 Texto inline

Algo genial de gtsummary, es que podemos usar las propias tablas para extraer detalles de los resultados y usarlos directamente en el texto.

El paquete report tiene también funcionalidades muy potentes que merece la pena explorar.

La ventaja de escribir los resultados de esta manera es que si hacemos algún pequeño cambio en la preparación de datos, podemos volver a correr el script de generación del reporte de resultados, y los valores p, etc. se ajustarán automáticamente. Únicamente tenemos que asegurarnos que la interpretación cualitativa no cambia :)

```
paste0(
  "Life expectancy was significantly associated with GDP per capita (log), beta = ",
  gtsummary::inline_text(table_model1, variable = gdpPercap_log)
)
#> [1] "Life expectancy was significantly associated with GDP per capita (log), beta = 7.2 (
```

Ejercicio - Resultados inferenciales

Usando la misma base de datos del ejercicio anterior:

```
DF_dafina = haven::read_sav(here::here("data/files/Dafina/Cancer screening risk literacy R1.
  as_tibble() |>
  select(IDparticipante, resident, screenbeliefs, compR1, numeracy) |>
  rename(comprehension = compR1)
```

Haz una regresión lineal prediciendo comprensión a partir de las otras variables de la base.

Finalmente, crea una tabla similar a la siguiente para reportar los resultados de tu análisis. Recuerda que en el manual de gtsummary tienes ejemplos para todo lo que necesitarás. En concreto, `tbl_regression()` es tu amiga:

Funciones necesarias

`add_global_p()` para los valores p. `bold_labels()` para poner en negrita los nombres de variables

Como añadir información en el pie de la tabla

Tendrás que usar la función `add_glance_source_note()` o `add_glance_table()`. Para saber que nombres poner en el parámetro `include`, puedes usar la función `broom::glance(model)`, donde model es el nombre que has usado para tu modelo estadístico.

Characteristic	Beta	95% CI[1]	p-value
(Intercept)	2.8	1.9, 3.8	<0.001
Resident	-0.23	-0.62, 0.17	0.3
Screening beliefs	-0.02	-0.05, 0.01	0.2
Numeracy BNT-S	0.14	0.03, 0.26	0.016

[1]CI = Confidence Interval
No. Obs. = 172; Adjusted R^2 = 0.047; Residual df = 168; Statistic = 3.83; p-value = 0.011; df = 3

7.6 Unir tablas

De manera muy sencilla podemos unir varias tablas. Esto es útil, por ejemplo, para mostrar los resultados de regresiones jerárquicas:

```
# Primero creamos un modelo más sencillo, basado en el anterior
model10 = lm(lifeExp ~ gdpPercap_log, DF_gapminder)

# Creamos la tabla
table_model10 = gtsummary::tbl_regression(model10, intercept = TRUE) |>
  add_global_p() |>
  bold_labels() |>
  italicize_levels() |>
  add_glance_table(include = c("nobs", "df.residual", "r.squared", "adj.r.squared"))

# Combinamos ambas tablas
tbl_merge(
  tbls = list(table_model10, table_model1),
  tab_spanner = c("**Baseline**", "**Step 1**")) |>
  # Necesario para que los parámetros globales de los modelos se muestren al final
  modify_table_body(~.x |> arrange(row_type == "glance_statistic")
  )
```

Characteristic	Baseline			Step 1		
	Beta	95% CI[1]	p-value	Beta	95% CI[1]	p-value
(Intercept)	64	62, 65	<0.001	63	62, 65	<0.001
gdpPercap_log	7.2	6.3, 8.1	<0.001	7.2	6.4, 8.1	<0.001
pop_log				0.81	0.04, 1.6	0.039
No. Obs.	142			142		
Residual df	140			139		
R^2	0.654			0.665		
Adjusted R^2	0.652			0.660		

[1]CI = Confidence Interval

7.7 Otros análisis y sus tablas

Que test estadístico debería usar, con código en R

Number of Dependent Variables	Nature of Independent Variables	Nature of Dependent Variable(s)*	Test(s)	How to SAS	How to Stata	How to SPSS	How to R
1	0 IVs (1 population)	interval & normal	one-sample t-test	SAS	Stata	SPSS	R
		ordinal or interval	one-sample median	SAS	Stata	SPSS	R
		categorical (2 categories)	binomial test	SAS	Stata	SPSS	R
		categorical	Chi-square goodness-of-fit	SAS	Stata	SPSS	R
	1 IV with 2 levels (independent groups)	interval & normal	2 independent sample t-test	SAS	Stata	SPSS	R
		ordinal or interval	Wilcoxon-Mann Whitney test	SAS	Stata	SPSS	R

7.7.1 Correlación simple

```
# Data
iris |> as_tibble()
#> # A tibble: 150 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1          5.1         3.5          1.4         0.2 setosa
#> 2          4.9         3            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           3.6          1.4         0.2 setosa
#> 6          5.4         3.9          1.7         0.4 setosa
```

```
#> # i 144 more rows

# Test
simple_corr_test = cor.test(iris$Sepal.Width, iris$Sepal.Length, method = "spearman")

# Report
simple_corr_test |> report::report()
#> Effect sizes were labelled following Funder's (2019) recommendations.
#>
#> The Spearman's rank correlation rho between iris$Sepal.Width and
#> iris$Sepal.Length is negative, statistically significant, and small (rho =
#> -0.17, S = 6.56e+05, p = 0.041)
```

7.7.2 Multiples correlaciones

```
# Multiple correlations
table_correlations = iris |>
  correlation(partial = FALSE, method = "spearman")

# Print table
TABLE_CORR = table_correlations |>
  summary(stars = FALSE, include_significance = TRUE, p_digits = 3)

# Fancy table
TABLE_CORR |>
  parameters::print_md()
```

Table 7.8: Correlation Matrix (spearman-method)

Parameter	Petal.Width	Petal.Length	Sepal.Width
Sepal.Length	0.83 ($p < .001$)	0.88 ($p < .001$)	-0.17 ($p = 0.041$)
Sepal.Width	-0.29 ($p < .001$)	-0.31 ($p < .001$)	
Petal.Length	0.94 ($p < .001$)		

p-value adjustment method: Holm (1979)

7.7.3 Anova

- Ver paquete {afex}

```
data(obk.long, package = "afex")
head(obk.long)
#>   id treatment gender   age phase hour value
#> 1  1   control      M -4.75   pre    1     1
#> 2  1   control      M -4.75   pre    2     2
#> 3  1   control      M -4.75   pre    3     4
#> 4  1   control      M -4.75   pre    4     2
#> 5  1   control      M -4.75   pre    5     1
#> 6  1   control      M -4.75  post    1     3

# estimate mixed ANOVA on the full design:
model = afex::aov_ez(id = "id",
                     dv = "value",
                     data = obk.long, between = c("treatment"),
        within = c("phase", "hour"))


  table_afex = papaja::apa_print(model)$table
  knitr::kable(table_afex)
```

term	estimate	conf.int	statistic	df	df.residual	p.value
Treatment	.211	[.000, .468]	2.91	2	13	.090
Phase	.164	[.000, .356]	19.29	1.74	22.64	$< .001$
Hour	.129	[.000, .237]	18.44	1.95	25.41	$< .001$
Treatment $\times$ Phase	.099	[.000, .212]	5.43	3.48	22.64	.004
Treatment $\times$ Hour	.001	[.000, .000]	0.08	3.91	25.41	.987
Phase $\times$ Hour	.017	[.000, .000]	1.35	4.02	52.29	.265
Treatment $\times$ Phase $\times$ Hour	.008	[.000, .000]	0.33	8.05	52.29	.951

7.7.4 Modelos mixtos

Primero preparamos los datos:

```
# Transform variables
DF_gapminder2 = gapminder |>
  # Log
  mutate(gdpPercap_log = log(gdpPercap),
         pop_log = log(pop)
         ) |>
  # Mean center variables so the 0 values have meaning
  mutate(year = year - mean(year, na.rm = TRUE),
         gdpPercap_log = gdpPercap_log - mean(gdpPercap_log, na.rm = TRUE),
         pop_log = pop_log - mean(pop_log, na.rm = TRUE))

# Reference levels and contrast coding
```

```r
DF_gapminder2 <- within(DF_gapminder2, continent <- relevel(continent, ref = "Oceania"))
contrasts(DF_gapminder2$continent) = car::contr.Sum(levels(DF_gapminder2$continent))
```

Creamos un modelo sencillo:

```r
model2 = lme4::lmer(lifeExp ~ gdpPercap_log + pop_log + year + (1|country), DF_gapminder2)

# Extraemos los R2 del modelo para usar en la tabla
R2_1 = performance::r2(model2)
```

Y mostramos la tabla de resultados. Como se trata de modelos mixtos, tenemos que añadir manualmente los R2's.

```r
table_model2 = gtsummary::tbl_regression(model2) |>  #, intercept = TRUE
  add_global_p() |>
  bold_labels() |>
  italicize_levels() |>
  add_glance_source_note(include = c("nobs", "df.residual"))

# broomExtra::glance_performance(model2)

table_model2 |>
  as_gt() |>
  gt::tab_source_note(gt::md(
    paste0(
      deparse1(model2@call$formula),
      "<BR> ",
      "R2 conditional = ",
      round(R2_1$R2_conditional, 3),
      ", R2 marginal = ",
      round(R2_1$R2_marginal, 3)
    )
  ))
```

Characteristic	Beta	95% CI[1]	p-value
gdpPercap_log	3.3	2.8, 3.8	<0.001
pop_log	6.1	5.4, 6.9	<0.001
year	0.15	0.13, 0.17	<0.001

[1]CI = Confidence Interval
No. Obs. = 1,704; Residual df = 1,698
lifeExp ~ gdpPercap_log + pop_log + year + (1 | country) R2 conditional = 0.964, R2 marginal = 0.49

Bibliografía

Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc. https://r4ds.had.co.nz/

8 Trabajo con Quarto para reportes reproducibles

Paquetes para este capítulo

```
if (!require('afex')) install.packages('afex'); library('afex')
if (!require('correlation')) install.packages("correlation"); library('correlation')
if (!require('corrr')) install.packages('corrr'); library('corrr')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('DT')) install.packages('DT'); library('DT')
if (!require('ggraph')) install.packages('ggraph'); library('ggraph')
if (!require('grateful')) install.packages('grateful'); library('grateful')
if (!require('here')) install.packages('here'); library('here')
if (!require('gtsummary')) install.packages('gtsummary'); library('gtsummary')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('papaja')) install.packages("papaja"); library('papaja')
if (!require('parameters')) install.packages('parameters'); library('parameters')
if (!require('quarto')) install.packages("quarto"); library('quarto')
if (!require('remotes')) install.packages('remotes'); library('remotes')
if (!require('renv')) install.packages("renv"); library('renv')
if (!require('report')) install.packages("report"); library('report')
if (!require('rticles')) install.packages('rticles'); library('rticles')
if (!require('see')) install.packages("see"); library('see')
if (!require('sjPlot')) install.packages('sjPlot'); library('sjPlot')
if (!require('stringi')) install.packages('stringi'); library('stringi')
if (!require('tinytex')) install.packages('tinytex'); library('tinytex')
if (!require('usethis')) install.packages('usethis'); library('usethis')
```

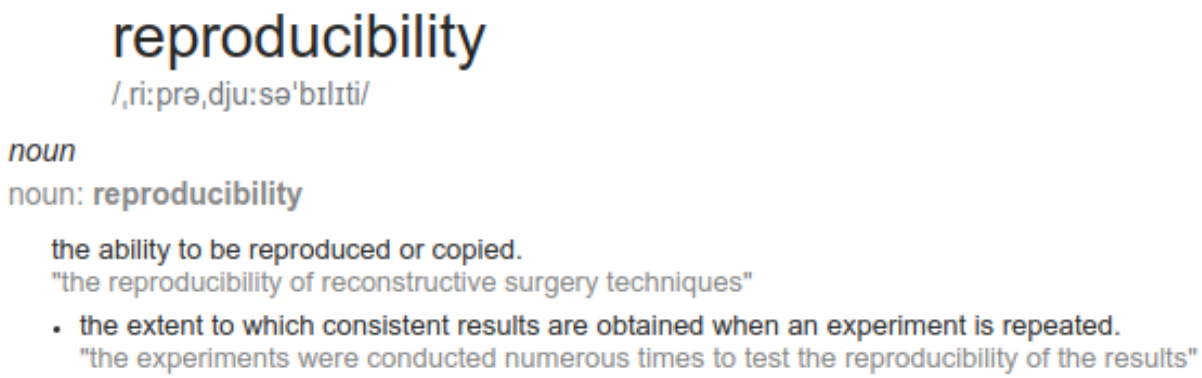
! Dependencias

Instalar Quarto
Quarto es un sistema de publicación de código abierto que funciona con diferentes lenguajes de programación como R o python.
Descarga e instala Quarto

Instalar latex:
Para generar pdf's necesitaremos tener instalado Latex. `tinytex` nos ayudará a simplificar el proceso:

```
tinytex::install_tinytex() # Llevará un rato
```

8.1 Que es la reproducibilidad



La crisis de replicación (*replication crisis*) se inició con un paper que trató de replicar los resultados de 100 investigaciones clásicas. Esta crisis ha generado un movimiento muy interesante dentro de las Ciencias Sociales y la Psicología en particular. Cada vez es más común aplicar algunos principios de buenas prácticas como compartir materiales, datos y scripts de análisis, para que tanto los revisores como otros investigadores puedan entender, reanalizar, etc. nuestras investigaciones.

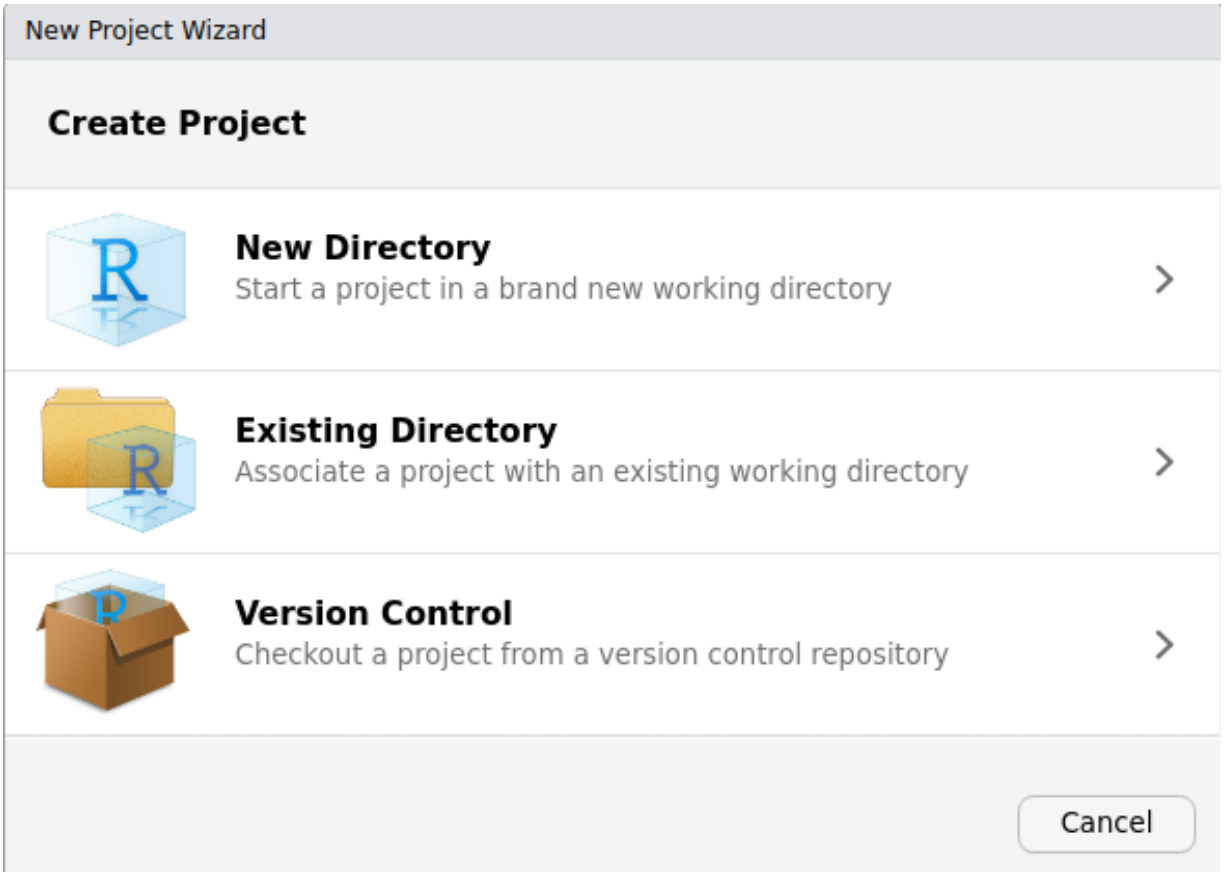
Hay algunas organizaciones que han surgido para tratar de mejorar la colaboración, transparencia, y manera de trabajar, como el Psychological Science Accelerator, la Peer Reviewer's Openness Initiative (PRO), o la Open Science Foundation. Una de las soluciones propuestas para resolver muchos de los problemas actuales pasa por los Registered reports. En estos se da una *restructured submission timeline: Before collecting data, authors submit a study protocol containing their hypotheses, planned methods, and analysis pipeline, which undergoes peer review.* La última evolución de los Registered Reports es la Peer Community in Registered Reports, la cual establece un sistema de revisión de RR global, independiente de las revistas.

Además de los motivos científicos para trabajar de manera más transparente y reproducible, hay también motivos prácticos. Si trabajamos de manera reproducible, las modificaciones en tablas, gráficas, número de participantes o reanálisis son triviales. En este capítulo vamos a ver algunos pasos fundamentales para tender un workflow que permita y ayude a la reproducibilidad.

8.2 Proyectos de R-Studio

El primer paso empieza por crear un proyecto de RStudio. Al usar proyectos, simplificamos varias cosas, haciendo más fácil compartir nuestro trabajo con otras personas, retomar nuestro trabajo, detectar cambios, etc. Podéis leer algo más sobre esto aquí.

Para crear un nuevo proyecto de RStudio, haz click en este icono (o File -> New Project), y sigue las instrucciones paso a paso del `New Project Wizard`:



8.3 Quarto/RMarkdown, openscience y análisis reproducibles

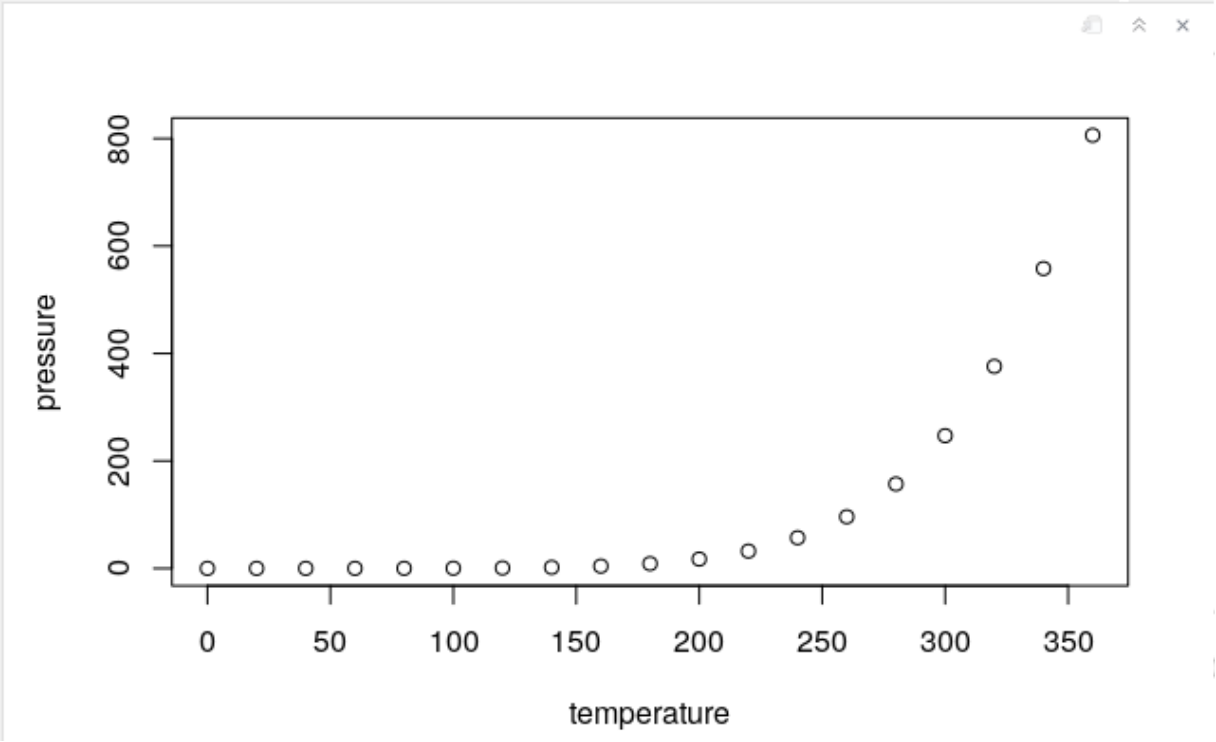
Quarto quarto o RMarkdown son herramientas que nos permiten combinar texto formateado con código y resultados en un mismo documento (html, pdf, docx, ...). Quarto es una evolución de Rmarkdown, y que facilita la interoperabilidad entre R, Python, Julia, etc. La diferencia esencial es que usaremos archivos `.qmd` en lugar de `.Rmd`, y que tendremos que instalar quarto en nuestro ordenador. Pero en general, resulta trivial convertir nuestros archivos Rmd a qmd (solo hay que renombrarlos).

Aprovechando la potencia de estas herramientas, algunas personas han creado paquetes para preparar artículos en formato APA, o con las plantillas de decenas de editoriales.

8.4 Sintaxis, chunks de código, tipos de archivo

La sintaxis básica de Quarto / RMarkdown es sorprendentemente sencilla, como se puede ver más abajo. Eso si, lo que hay detrás es toda la potencia de latex, así que el cielo es el límite.

````
# Soy un titulo grande

## Soy un título algo más pequeño

Texto sencillo, cosas en **negrita**, *itálica*, ~~tachadas~~, etc.

Podemos también incluir pedazos de código que muestren o no el output, plots, etc.

```{r nombre_chunk}

plot(pressure)

```
````



Y como no, tenemos mucha ayuda:

- quarto
- Markdown basics
- R Markdown: The Definitive Guide
- Web oficial de Rmarkdown dentro de RStudio

Resumiendo, tienes tres elementos básicos:

8.4.1 Cabecera YAML

Cuando creas un documento .qmd nuevo verás algo similar a lo siguiente en las primeras lineas:

```
---
title: "Untitled"
---
```

Esta es la cabecera YAML, en la cual se le pueden pasar parámetros para añadir un índice, cambiar formato, y muchas otras cosas.

8.4.2 Markdown

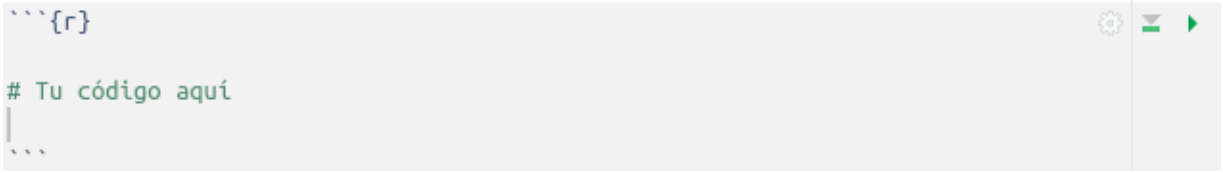
En el resto del documento (con la excepción de los chunks de código), el formato que usaremos será Markdown. Su sintaxis es muy sencilla pero nada tolerante. Podéis ver las bases en la Markdown basics.

IMPORTANTE. Si algo no funciona como esperas:

> **Si algo no funciona**
>
> - Añade saltos de linea entre párrafos.
> - Añade dos espacios al final de las líneas.
> - Añade un espacio después de #:
>   - MAL: `#Título grande`
>   - BIEN: `# Título grande`

8.4.3 Chunks de código

Los chunks de código están delimitados por:



En su interior, puedes usar código R como si estuvieras en un script de R normal.

```r
library(dplyr)

myvariable = c(1, 2, 3)
```

Para insertar un chunk de código, solo tienes que hacer `Control + Alt + I`.

En su cabecera puedes añadir opciones. Hay una cantidad apabullante de opciones. Por ejemplo, en el siguiente chunk:

```
{r nombre_chunk, eval=TRUE, include=TRUE, fig.height=10, fig.width=12, message=FALSE, warning=FALSE, cache=TRUE, results='asis'}
```

> **i** Parametros chunks
>
> - `echo=FALSE`: Esconde el código pero este sigue corriendo
> - `eval=TRUE`: Evalúa el código
> - `include=TRUE`: Incluye el código
> - `fig.height=10`: altura de los plots (en inches)
> - `fig.width=12`: ancho de los plots (en inches)
> - `message=FALSE`: NO muestres mensajes
> - `warning=FALSE`: NO muestres warnings
> - `cache=TRUE`: cachea el output del plot
> - `results='asis'`: muestra el output tal cual (importante cuando el output es en latex/pdf)

TRUCO:

- Un chunk llamado `{r setup, include=FALSE}` al principio de tu documento .qmd / .Rmd, es ideal para poner tus librerías, lectura de datos inicial, etc.

Ejercicio básico Quarto / RMarkdown

Vamos a empezar creando un nuevo proyecto de RStudio. En este proyecto crearemos un documento con la estructura de artículo científico, que acabará siendo la entrega final de el Workshop.

- `File -> New Project -> New directory -> New project`

Crea un nuevo documento .qmd:

- `File -> New File -> Quarto document`
- Selecciona el formato de output `PDF`

Ahora hagamos lo siguiente:

1. Dale formato de artículo científico, creando las siguientes secciones:

- Title
- Abstract

- Introducción
- Materials and Methods
  - Participants
  - Materials
- Results
  - Experiment 1
- Discussion
- Bibliography

2. Pon texto de relleno dentro de cada sección. Para ello puedes usar la función `stringi::stri_rand_lipsum(n_paragraphs = 1)` del paquete {stringi}. Los chunks de código deberán ser similares a este:

“‘{r, echo=FALSE, results=‘asis’}

cat(stringi::stri_rand_lipsum(n_paragraphs = 1))

“‘

---

3. Renderiza tu documento en formato PDF.

El resultado del ejercicio anterior deberá ser un archivo pdf con la estructura general de un artículo científico. Puedes ver el archivo qmd y su pdf resultante en `data/files/07-markdown/`.

Si el botón `Render` no funciona, puedes renderizar el pdf usando: `quarto::quarto_render("data/files/07-rm`
`output_format = "pdf")`

Ejercicio avanzado

Usando los datos de `gapminder`, incluye en el apartado Resultados del documento .qmd que has creado:

1. Una tabla de descriptivos.
2. Una tabla con los resultados de un análisis sencillo (e.g. una regresión lineal).
3. Un gráfico mostrando la relación entre dos de las variables.

Podéis ver ejemplos de tablas de descriptivos o tablas de resultados inferenciales en el capítulo anterior, y de gráficas en los capítulos de Introducción a la visualización y Visualización avanzada.

Recuerda empezar añadiendo el chunk de setup `{r setup, include=FALSE}` donde incluiras las llamadas a librerias necesarias (al menos `ggplot2`, `gtsummary`).

El resultado de este ejercicio deberá ser un archivo pdf como el que se puede ver en `data/files/07-markdown/`.

De nuevo, si el botón `Render` no funciona, puedes renderizar el pdf usando: `quarto::quarto_render("data/fil output_format = "pdf")`

8.5 Avanzado

Puedes crear artículos en formato APA, añadir bibliografía a tus documentos fácilmente, citar los paquetes de R que usas, etc.

8.5.1 Artículos APA con Papaja

- Preparar artículos en formato APA

```
install.packages("papaja")

# Create new R Markdown file
rmarkdown::draft(
  here::here("data", "output", "mymanuscript.Rmd"),
  "apa6",
  package = "papaja",
```

```
  create_dir = FALSE,
  edit = FALSE)

# Render manuscript
rmarkdown::render(
  here::here("data", "output", "mymanuscript.Rmd"),
  quiet = TRUE,
  clean = TRUE)
```

Y no olvidemos el paquete {rticles}, que contiene plantillas de decenas de editoriales

8.5.2 Usar bibliografía

Puedes incluir citas en tu documento fácilmente con Rmarkdown o Quarto.

Necesitaras un archivo .bib e incluirlo en el yaml inicial, por ejemplo: `bibliography: name_file.bib`.

A partir de ahí, puedes citar artículos simplemente incluyendo `Blah Blah [@wickham2015; @knuth1984].` o `@knuth1984 says blah.`.

Para saber más:

- https://quarto.org/docs/authoring/footnotes-and-citations.html
- https://blog.rstudio.com/2020/11/09/rstudio-1-4-preview-citations/
- https://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html
- https://www.r-bloggers.com/bibliography-with-knitr-cite-your-references-and-packages/

8.5.3 Citar los paquetes que usamos

¿Debemos citar los paquetes que usamos?

- Respuesta corta, si
- Respuesta larga, la mayoría de los paquetes

Una manera muy sencilla de hacer esto es usando {grateful}.

```
grateful::cite_packages(
  pkgs = "All",
  output = "file",
  out.format = "Rmd",
  include.RStudio = TRUE,
  out.dir = "~/Downloads"
)
```

8.5.4 Manejo de dependencias

Usando un sistema de manejo de dependencias renv creamos un snapshot de las librerías usadas actualmente. Es muy importante para garantizar que nuestros scripts correrán en el futuro.

> **Funcionamiento básico `renv`**
>
> 0. Si no lo hemos hecho, Instalamos renv: `install.packages("renv")`
> 1. Inicializamos el entorno local de un nuevo proyecto, con una librería privada de R `renv::init()`
> 2. Trabajamos en el proyecto, instalando los paquetes que necesitemos
> 3. Guardamos el estado de las librerías usadas en el proyecto en un lockfile (llamado renv.lock), `renv::snapshot()`
> 4. Restauramos el estado de las librerías a partir del lockfile generado por renv::snapshot(). `renv::restore()`

Ejercicio renv

Usando el proyecto RStudio del artículo científico que estamos creando:

1. Inicializa `renv`
2. Intenta usar una librería nueva (e.g. lee un archivo con `readr::read_csv`)
3. Si el paso anterior da un error. ¿Cómo lo puedes solucionar?
4. Borra la carpeta `renv` y restaura el estado de las librerías a partir del lockfile

8.5.1 Shortcuts!

- Alt+SHIFT+K: Ver shortcuts!
- CTRL+SHIFT+M: Pipe
- CTRL+SHIFT+A: Reformat code
- CTRL+I: Reindent lines

8.5.2 Estilo

Es recomendable ser consistente en la manera de escribir código. Habitualmente se recomienda seguir una guía de estilo. Por ejemplo, Hadley Wickham's Style guide o la guia de estilo del tidyverse.

Bibliografía

Guia de estilo del tidyverse

Hadley Wickham's Style guide

targets

Scheel, A. M., Schijen, M., & Lakens, D. (in press). An excess of positive results: Comparing the standard Psychology literature with Registered Reports. Advances in Methods and Practices in Psychological Science.

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). R Markdown: The Definitive Guide. CRC Press. https://bookdown.org/yihui/rmarkdown/

Yihui Xie (2018). bookdown: Authoring Books and Technical Documents with R Markdown https://bookdown.org/yihui/bookdown/markdown-syntax.html

- Mas cosas sobre reproducibilidad:
  - Reproducibility project: Psychology
  - Many labs 2

9 Control de cambios con Git y Github

Paquetes para este capítulo

```
if (!require('afex')) install.packages('afex'); library('afex')
if (!require('correlation')) install.packages("correlation"); library('correlation')
if (!require('corrr')) install.packages('corrr'); library('corrr')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('DT')) install.packages('DT'); library('DT')
if (!require('ggraph')) install.packages('ggraph'); library('ggraph')
if (!require('here')) install.packages('here'); library('here')
if (!require('gtsummary')) install.packages('gtsummary'); library('gtsummary')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('papaja')) install.packages("papaja"); library('papaja')
if (!require('parameters')) install.packages('parameters'); library('parameters')
if (!require('remotes')) install.packages('remotes'); library('remotes')
if (!require('renv')) install.packages("renv"); library('renv')
if (!require('report')) install.packages("report"); library('report')
if (!require('rticles')) install.packages('rticles'); library('rticles')
if (!require('see')) install.packages("see"); library('see')
if (!require('sjPlot')) install.packages('sjPlot'); library('sjPlot')
if (!require('stringi')) install.packages('stringi'); library('stringi')
if (!require('tinytex')) install.packages('tinytex'); library('tinytex')
if (!require('usethis')) install.packages('usethis'); library('usethis')
```

! Dependencias

Vamos a necesitar Git para poder trabajar:
Instalar Git
Ver instrucciones para Windows, Mac y Linux.
Windows: en el paso *Adjusting your PATH environment*, selecciona `Git from the command line and also from 3rd-party software`

9.1 Git

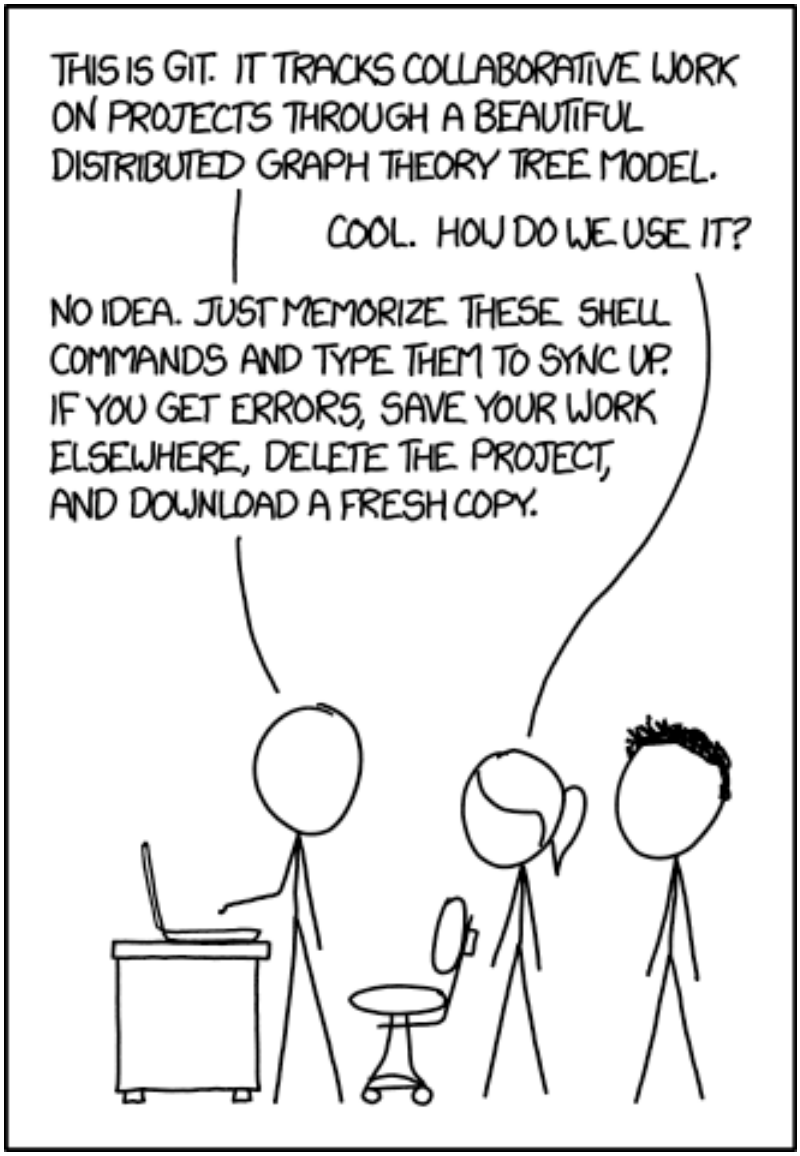


Figure 9.1: SOURCE: https://xkcd.com/1597/

Un segundo elemento que nos va a ayudar a trabajar en equipo, y a evitar problemas en proyectos relativamente complejos es el uso de un sistema de control de versiones como Git. Los proyectos de RStudio hacen especialmente sencillo usar algunas funcionalidades básicas de Git.

Algunas referencias útiles:

- OhshitGit website
- Git in practice
- **happygitwithr**

9.2 Github

Figure 9.2: SOURCE: github.githubassets.com

Github es una plataforma web muy popular donde almacenar proyectos de programación que usa como motor. Muchos de los paquetes de R, el mismo RStudio, etc, tienen repositorios abiertos en Github. Una de las ventajas fundamentales de usar Github es que esta plataforma integra algunas herramientas para hacer más sencillo el control de versiones, como el pull request, que nos permite combinar ramas de proyectos sin apenas problemas.

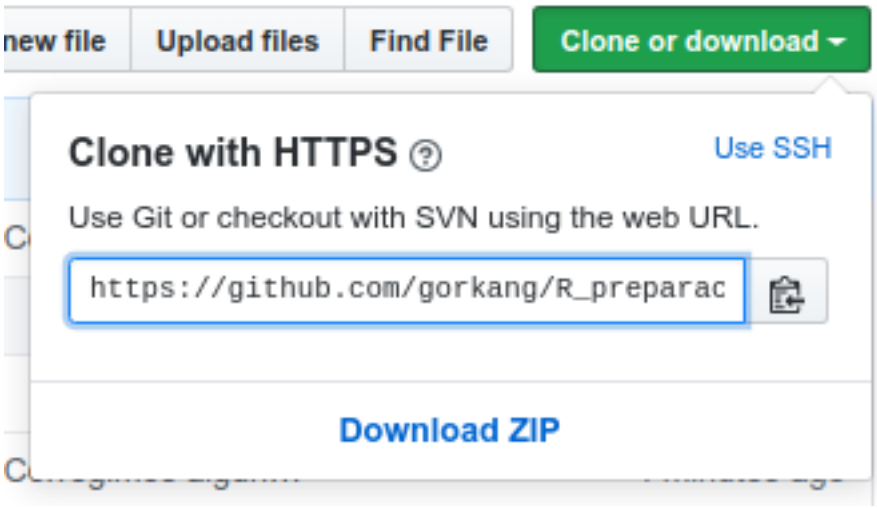
Github tiene un programa especial para estudiantes: https://education.github.com/

9.3 Clonar un repositorio existente

Algo que podemos hacer con todos los repositorios de Github es clonarlos localmente:

Clonar repositorio Github

Primero, copiamos la `repository URL` del repo de Github (ver imagen de abajo). Será algo similar a `https://github.com/VUESTRO_NOMBRE_DE_USUARIO/NOMBRE_REPOSITORIO.git`



Segundo, en RStudio: `File > New Project > Version Control > Git`

9.4 Crear un proyecto en RStudio asociado a Github

Tener nuestros proyectos de RStudio asociados a repositorios en Github es muy útil para poder compartir el código y datos asociados a nuestras investigaciones, disponer de una copia de seguridad online, y, si lo usamos adecuadamente, detectar de manera más rápida cual de los últimos cambios es responsable de los errores que nos aparezcan.

! Creando personal access token

La primera vez que usemos Github asociado a RStudio tendremos que crear un personal access token.
Método automático:
`usethis::create_github_token()`
Método manual, como los animales:
En tu página de Github, haz click en tu icono (arriba a la derecha) -> Settings -> Developer settings -> Personal access tokens -> [Generate new token] -> Give gist, repo and workflow permissions.

La manera más sencilla de tener un proyecto de RStudio vinculado a repositorio de Github es empezar creando un repositorio en Github.

i Empezando desde 0 [recomendado]

Podemos empezar creando un repositorio en Github, para después clonarlo localmente. Para eso, en Github:

1. Creamos repositorio nuevo
2. `Initialize this repository with a README`
3. Clonar repositorio

Alternativamente, si ya tenemos un proyecto de RStudio y hemos avanzado en nuestra preparación, análisis de datos, etc. podemos usar `usethis::use_github()` para que nos cree y asocie automáticamente un repositorio de Github.

Si ya tenemos el proyecto de RStudio creado

Preparación

1. Crear un repositorio local de git (solo si no lo tenemos aún): `usethis::use_git()` (se crea una carpeta oculta llamada .git)
2. Insertar token en archivo `.Renviron` (si no lo tienes, ver arriba, `Creando personal access token`): `usethis::edit_r_environ()`

Asociando a repositorio en Github

1. Crear Github repo: `usethis::use_github()`
2. Empujar el repositorio local a Github: `git push --set-upstream origin master`

Ejercicios

Ejercicio 1: Github-RStudio

Vamos a poner a prueba lo anterior, creando un repositorio en Github llamado `Github_Rstudio`, y clonándolo para tener un proyecto de RStudio asociado.

1. Abre una cuenta en Github y/o haz login
2. Crea un repositorio en Github (después podrás borrarlo si quieres)
3. Sigue los pasos de arriba para clonar el repositorio de Github

---

Ejercicio 2: RStudio-Github

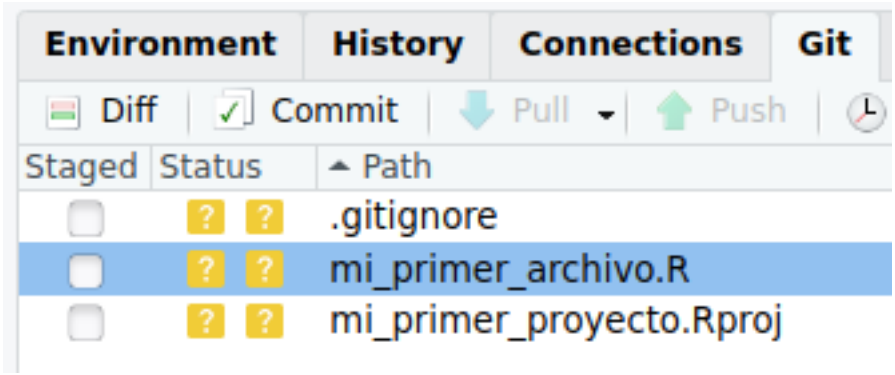
Ahora podemos usar el método inverso. Si tenemos un proyecto de RStudio, vamos a crear un repositorio de Github asociado.

1. Crea un nuevo repositorio de RStudio. Si marcas la pestaña `Create a git repository` te ahorrarás uno de los pasos (`usethis::use_git()`)
2. Crea un script donde hagas algo muy sencillo (puedes copiar algún fragmento de código de temas anteriores)
3. Sigue los pasos de arriba para crear automáticamente un repositorio de Github asociado

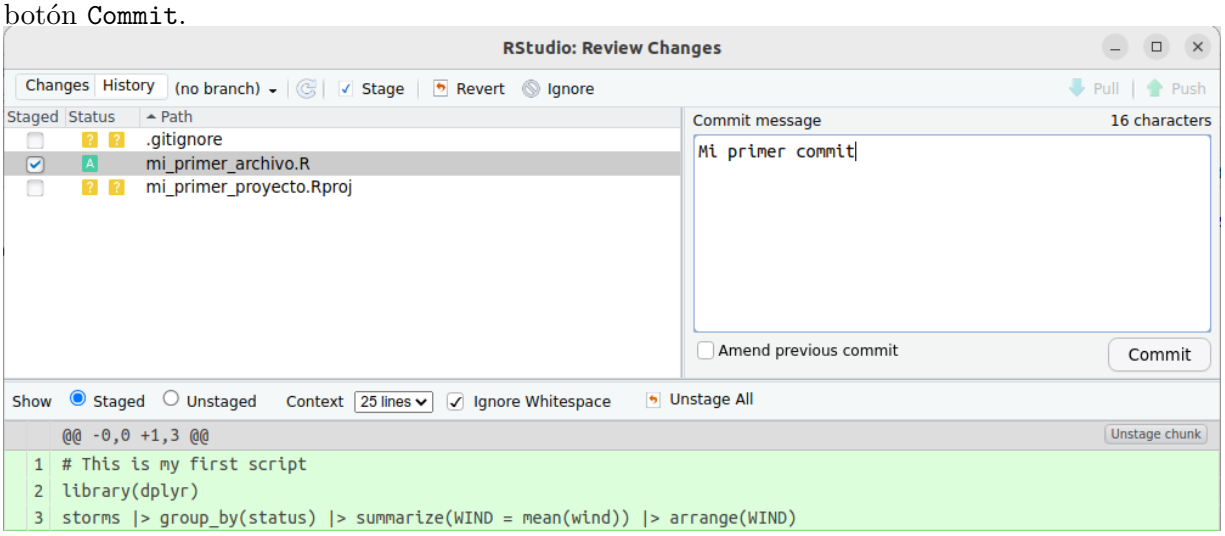
9.5 Commits

Git es extraordinariamente potente, pero vamos a empezar por la funcionalidad básica, `commit`(enviar/encomendar) archivos.

Si partimos de un nuevo proyecto de RStudio llamado `mi_primer_proyecto` en el que hemos añadido un repositorio de Git (`usethis::use_git()`), y creado un nuevo archivo llamado `mi_primer_archivo.R`, en el panel `Git` veremos algo similar a esto:

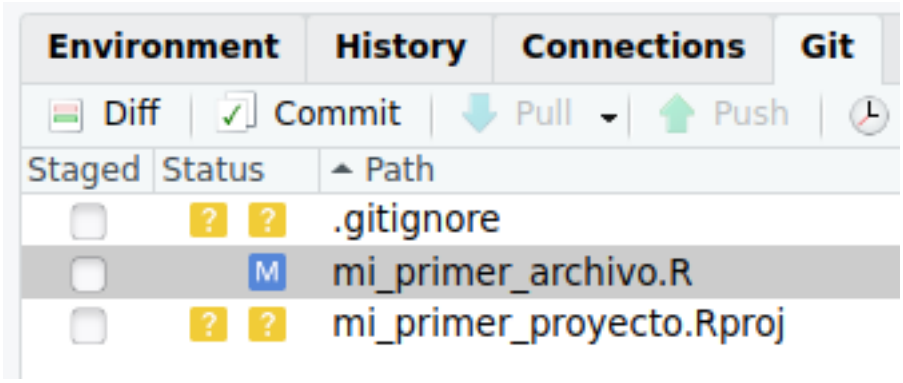


Si hacemos cambios en `mi_primer_archivo.R` y hacemos click en Commit  , aparecerá una ventana para revisar los cambios. Hacemos doble click en `mi_primer_archivo.R`, y veremos que se pone en verde. Añadimos un `Commit message`, y estamos listos para hacer click en el botón `Commit`.

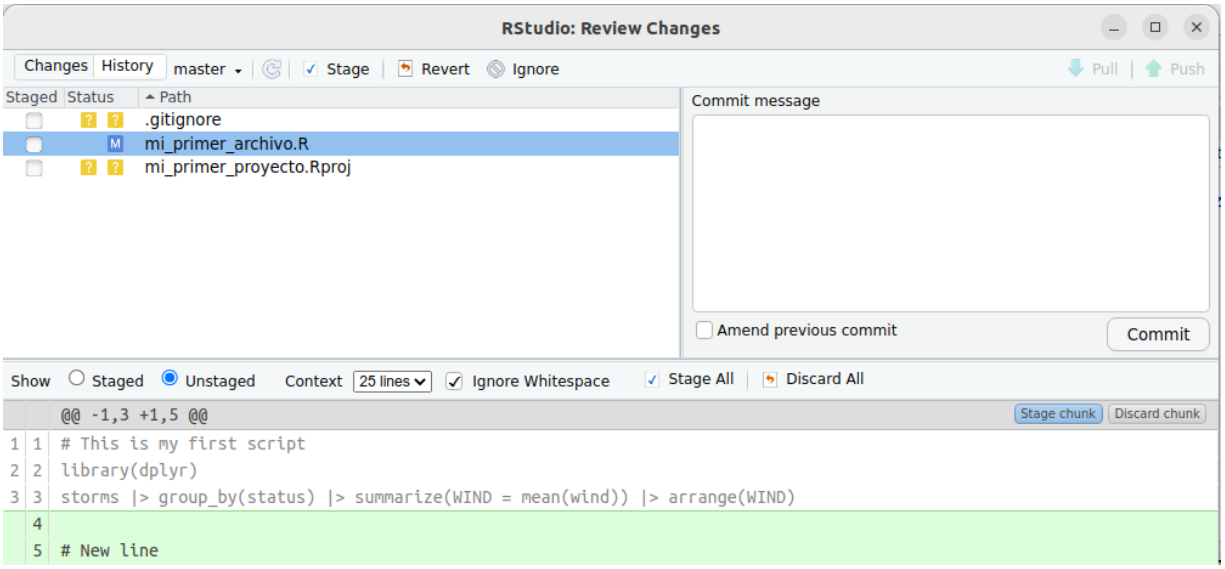


A partir de este momento, cualquier cambio en `mi_primer_archivo.R` será detectado por Git.

Por ejemplo, si añadimos una línea, veremos que el icono junto al archivo se convierte en una M en un recuadro azul.



Si hacemos de nuevo click en Commit  , podemos ver los cambios en `mi_primer_archivo.R`.



9.6 Pull, Push

Con los comandos pull y push:

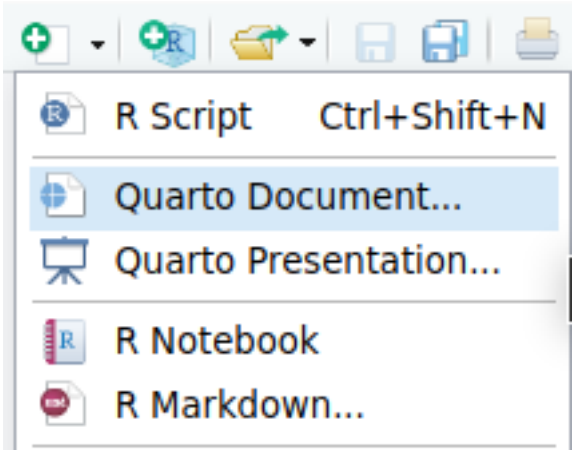
Pull: nos aseguramos que nuestro repositorio local esta actualizado

Push: subimos los cambios commiteados de la rama a Github

Ejercicio

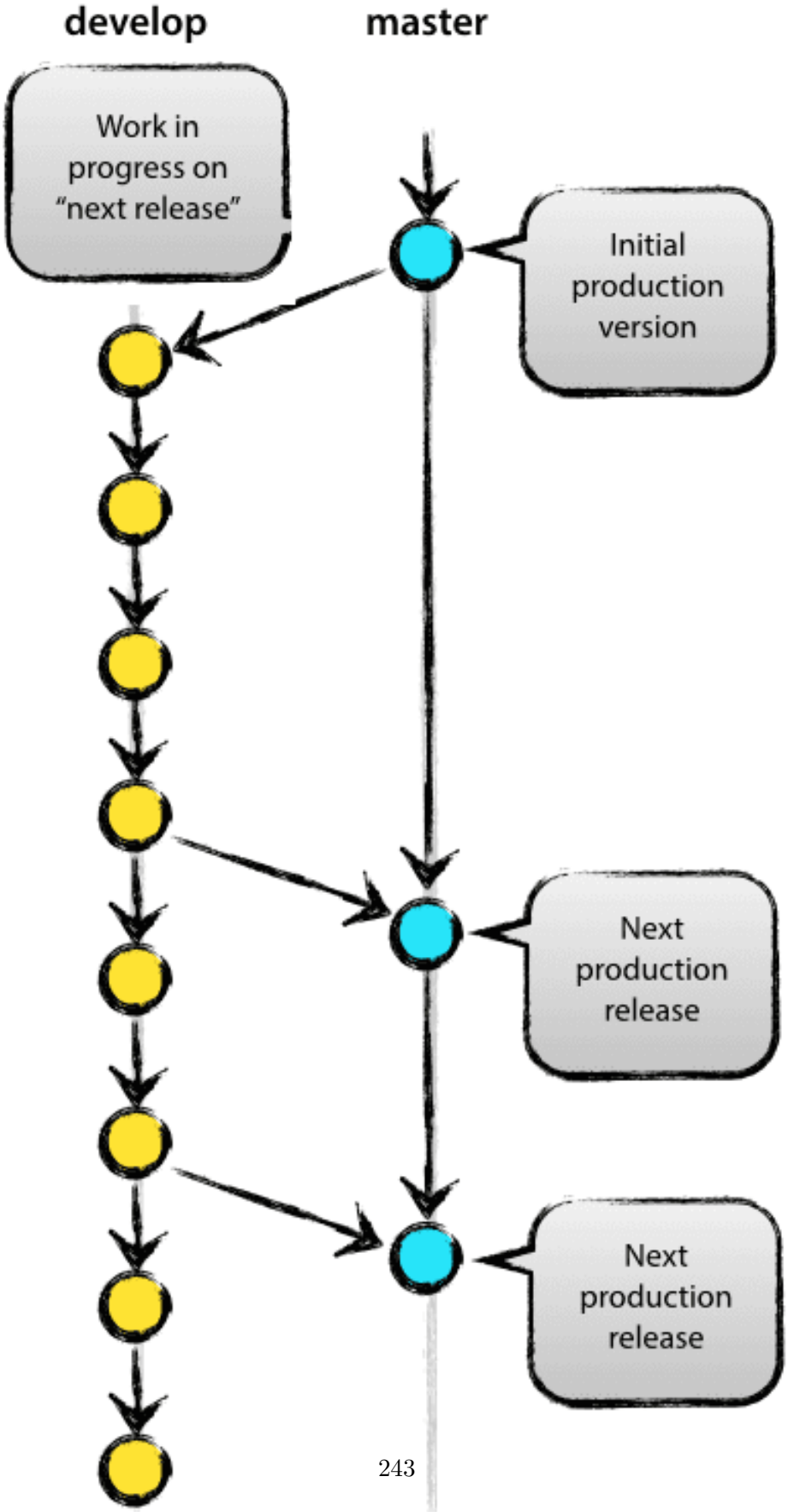
Nuestro primer commit

1. Usando el proyecto de RStudio de antes, crea un nuevo documento de Quarto (.qmd):



2. Haz un commit de ese archivo y súbelo (push) a Github (asegúrate que esta allá!). No olvides hacer un pull!
3. Ahora haz cambios en el archivo localmente. Una vez hechos:
    - Commitealos
    - Súbelos (Pull & Push)
    - Sincroniza tu repo local (Pull final)

9.7 Workflow


develop
master
Work in progress on "next release"
Initial production version
Next production release
Next production release

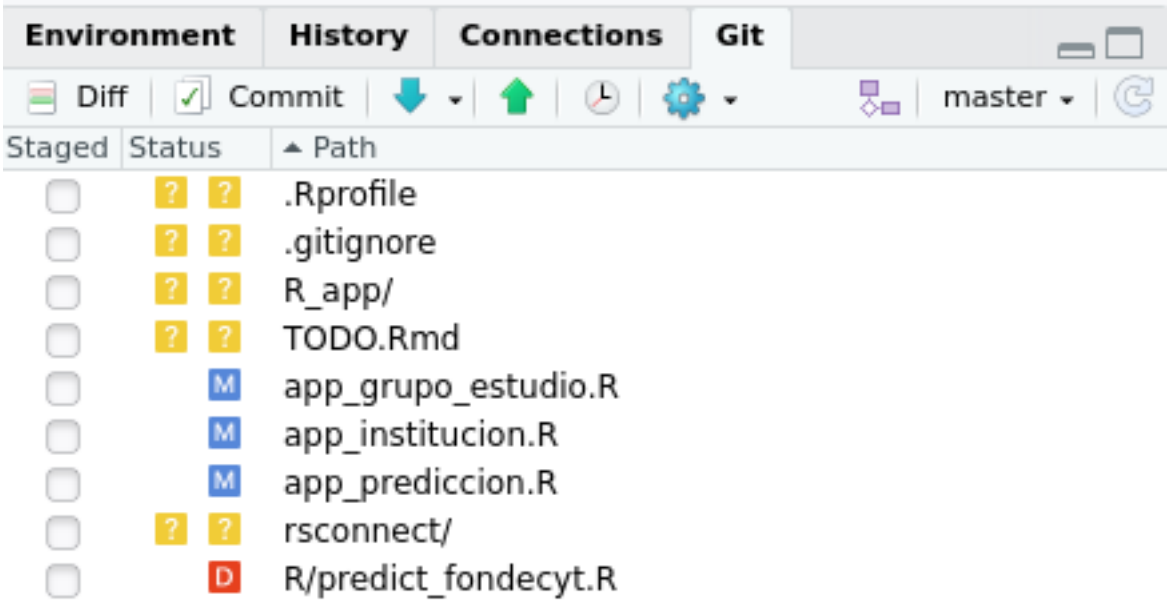
Hay diferentes filosofías sobre cual es la mejor manera de trabajar con Git.

En este post por Vincent Driessen podéis ver una explicación bien detallada, complementada con imágenes como la que se ve a continuación.

El modelo básico implica la existencia de dos ramas. Una `main` o master ("producción"), donde tenemos código que siempre debe funcionar, y una `development` (para desarrollo), donde experimentamos, rompemos cosas, etc.

Podéis ver un manual super completo llamado Happy Git and GitHub for the useR elaborado por Jenny Bryan, Jim Hester, entre otros.

En RStudio podemos trabajar gráficamente, Usando el panel `Git`.



Usando el entorno gráfico

Empezamos en la rama master:

0. **Pull** : nos aseguramos que nuestro repositorio local esta actualizado
1. **Branch** : Creamos nueva rama llamada **development**
2. Hacemos cambios en nuestros scripts
3. **Commit** : Commiteamos los cambios

4. **Push**  : subimos la rama a Github

5. **Pull request** (En Github):

- Compare & Pull request

6. **Pull**  : nos aseguramos que nuestro repositorio local esta actualizado

Como hacerlo usando el `terminal`

0. **Pull**: nos aseguramos que nuestro repositorio local esta actualizado: `git pull`
1. **Branch**: Creamos nueva rama llamada **development**: `git checkout -b development`

2. Hacemos cambios en nuestros scripts

3. **Commit**: Commiteamos los cambios

- Añadimos archivos: `git add foo.txt`
- Hacemos el commit: `git commit --message "A commit message"`

4. **Push**: subimos la rama a Github: `git push origin development`

5. **Pull request** (En Github):

- Compare & Pull request

6. **Pull**: nos aseguramos que nuestro repositorio local esta actualizado: `git pull`

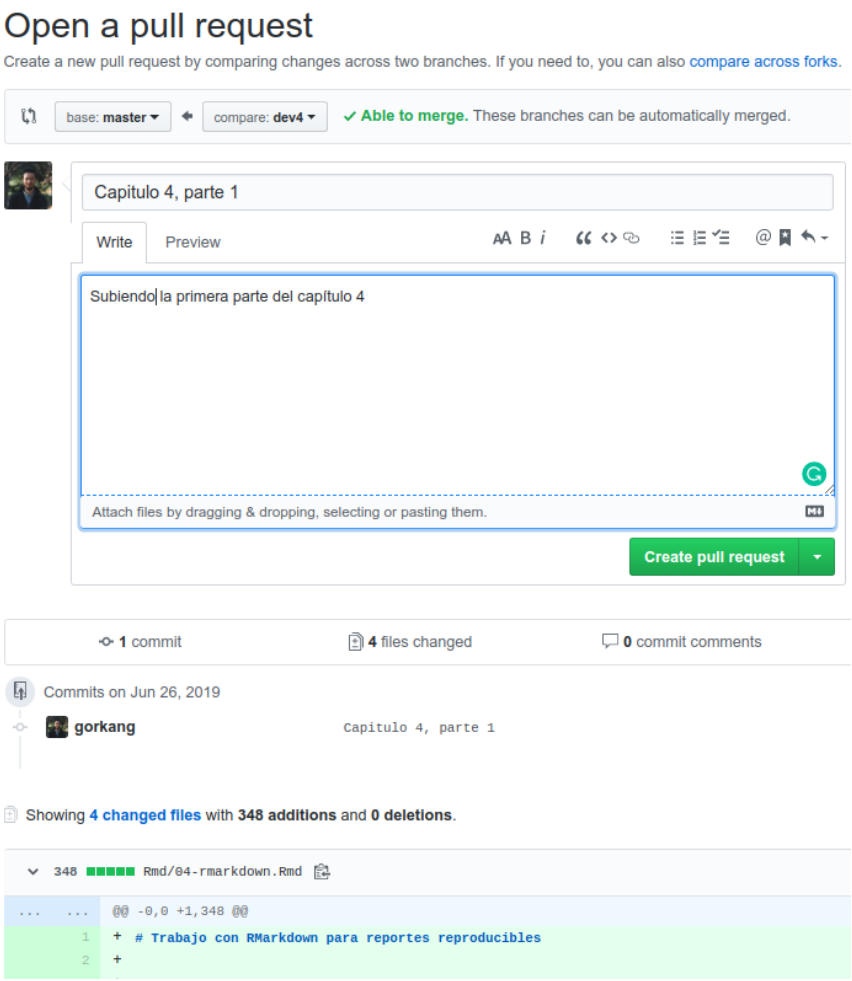
9.7.1 Pull request en 3 sencillos pasos

Los `Pull request` son una funcionalidad de Github que facilita colaborar con otras personas, contribuir a proyectos, etc. En esencia, automatizan la comprobación de cambios y, si no hay conflictos, permiten combinar el código nuevo con unos pocos clicks.

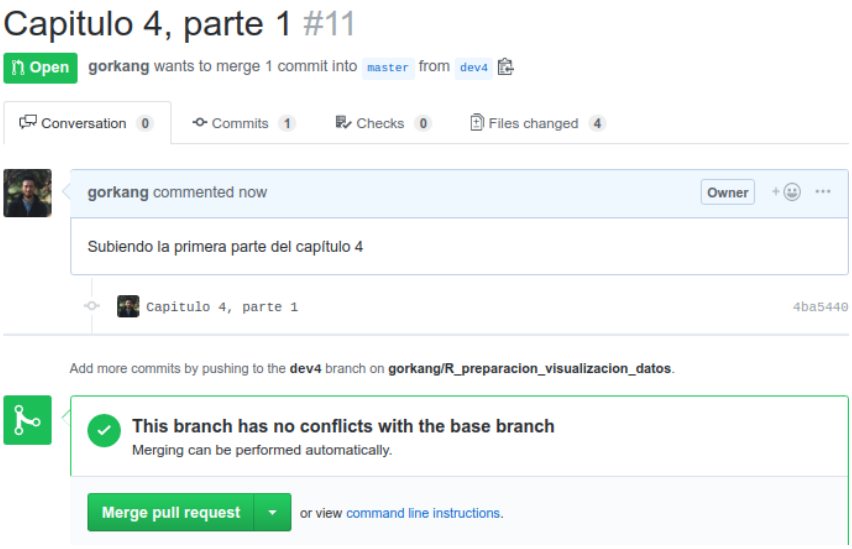
Después de hacer el push de arriba (paso 4), al entrar en nuestro repositorio de Github deberíamos ver algo parecido a lo siguiente (si no lo vemos, ir a **branches**). La única dificultad es saber cual de los botones verdes apretar:

Paso 1. Compare & pull request

Paso 2. Create pull request



Paso 3. Merge pull request



- Borrar rama antigua

Bibliografía

Guia de estilo del tidyverse

Hadley Wickham's Style guide

Happy Git and GitHub for the useR

targets

Scheel, A. M., Schijen, M., & Lakens, D. (in press). An excess of positive results: Comparing the standard Psychology literature with Registered Reports. Advances in Methods and Practices in Psychological Science.

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). R Markdown: The Definitive Guide. CRC Press. https://bookdown.org/yihui/rmarkdown/

Yihui Xie (2018). bookdown: Authoring Books and Technical Documents with R Markdown https://bookdown.org/yihui/bookdown/markdown-syntax.html

- Mas cosas sobre reproducibilidad:
  - Reproducibility project: Psychology
  - Many labs 2

10 Experimentos reproducibles

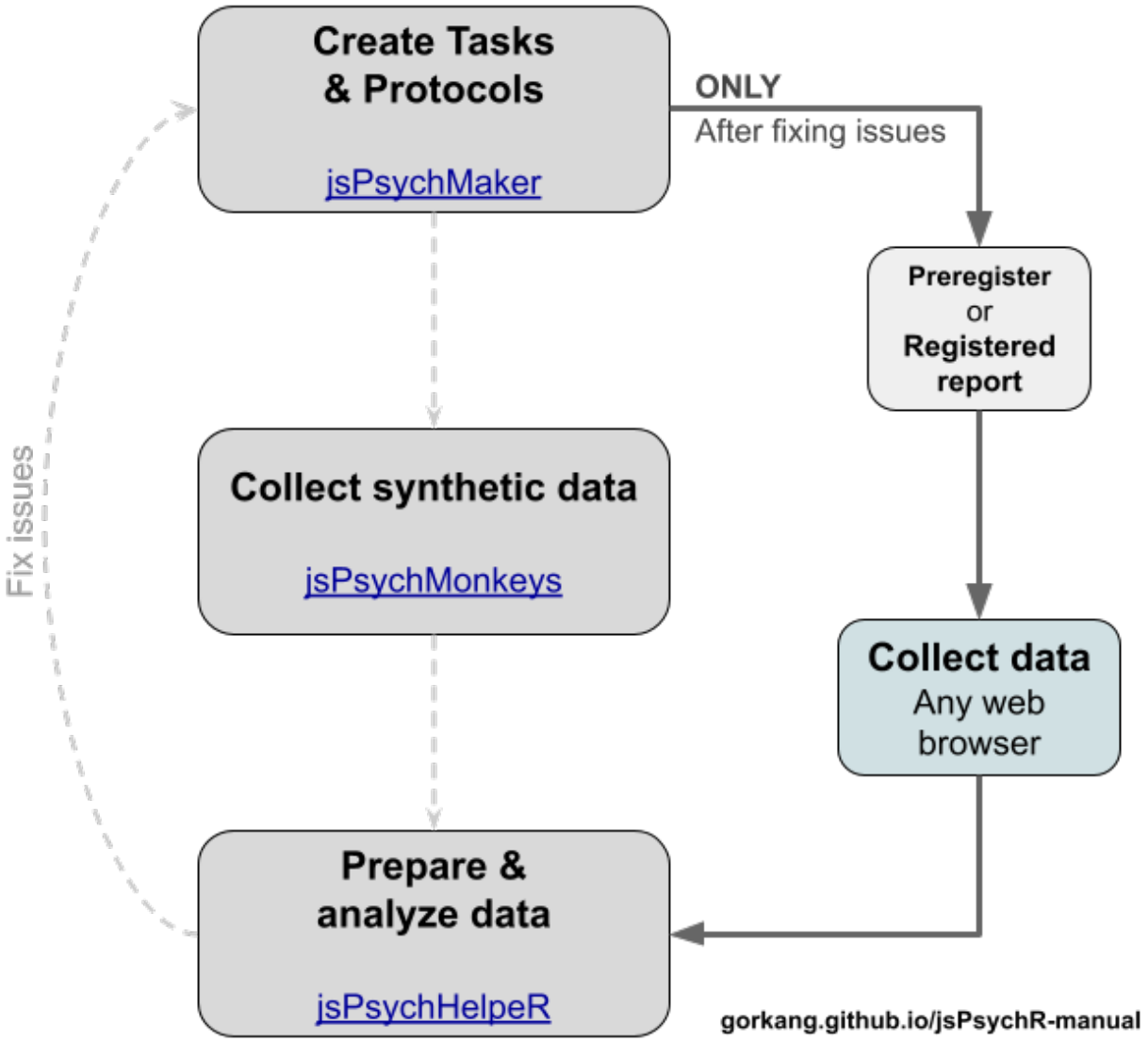
Paquetes para este capítulo

```
if (!require('jsPsychMaker')) remotes::install_github("gorkang/jsPsychMaker"); library('jsPs
if (!require('jsPsychMonkeys')) remotes::install_github("gorkang/jsPsychMonkeys"); library('
if (!require('jsPsychHelpeR')) remotes::install_github("gorkang/jsPsychHelpeR"); library('js
```

En el CSCN usamos distintas tecnologías para desarrollar experimentos. Algunos ejemplos son Psychopy, Qualtrics, Limesurvey, jsPsych, Gorilla, etc. Cada una de estas tiene ventajas y desventajas, y en general es importante tener en cuenta aspectos pragmáticos a la hora de adoptar una u otra tecnología (costo económico, tipo de experimento [EEG/conductual, laboratorio/online]).

Algunos de nosotros hemos optado principalmente por jsPsych para experimentos conductuales por tratarse de una librería javascript de **código abierto**, basada en tecnologías web standard, y que puede ser usada online y offline. Dado que en el CSCN disponemos de servidor propio, los costos habituales de hosting no se aplican.

En los últimos años, hemos empezado a trabajar en un conjunto de herramientas (jsPsychR) para crear experimentos usando la librería jsPsych con jsPsychMaker, simular participantes con jspsychMonkeys y estandarizar y automatizar la preparación de datos con jsPsychHelpeR.



Nuestro objetivo final es tener un gran número de tareas disponibles para ser usadas en el repositorio de jsPsychMaker. Cada una de estas tareas funcionará en jspsychMonkeys para crear participantes virtuales. Cada tarea tendrá un script hermano en jsPsychHelpeR para automatizar la preparación de datos.

Puedes consultar las tareas disponibles en el manual de jsPsychR para más detalles.

10.1 Pipeline experimental abierto y reproducible

Replicar el experimento de una publicación no es trivial. Una de las fortalezas fundamentales de nuestro sistema es que compartir y reproducir un experimento y los análisis asociados se convierte en algo muy sencillo.

Además, todos los componentes del proceso son código abierto, lo que permite que revisores, colaboradores, etc. puedan verificar que no hay errores en el código.

Con este sistema podremos crear fácilmente el código del experimento, simular datos y preparar datos de manera casi automática (incluyendo anonimización).

El output del sistema es estandarizado, lo que implica que los nombres de las variables y la estructura de datos son predecibles. Finalmente, la generación de gráficas, tablas, reportes y los análisis son reproducibles.

10.2 jsPsychMaker: Como crear un protocolo experimental

En el manual de jsPsychR puedes ver las tareas disponibles junto con una breve descripción de cada una de ellas. Alternativamente, puedes ver el documento con todos los detalles de las tareas disponibles, o simplemente ejecutar `jsPsychMaker::list_available_tasks()`.

Si quieres consultar los scripts de las tareas puedes hacerlo en la carpeta `canonical_protocol/` del repositorio de jsPsychMaker. Si quieres crear una nueva tarea para añadir a tu protocolo, puedes seguir las instrucciones de más abajo.

Para crear un protocolo con las tareas `AIM`, `EAR` e `IRI`, y abrirlo en un navegador:

```
if (!require('pak')) utils::install.packages('pak'); pak::pkg_install("gorkang/jsPsychMaker"

jsPsychMaker::create_protocol(
  # Pruebas a incluir
  canonical_tasks = c("AIM", "EAR", "IRI"),
   # El directorio tiene que incluir un número (se usará como pid)
  folder_output = "~/Downloads/protocol999",
  # Abre el navegador con el protocolo
  launch_browser = TRUE
  )
```

- Podemos editar la configuración del protocolo en la carpeta que hemos indicado en `folder_output`, abriendo el archivo `config.js`. Puedes consultar la ayuda sobre la configuración de experimentos.
- El experimento esta listo para ser utilizado localmente. Si `launch_browser = TRUE` se abrirá el navegador. En cualquier caso, podemos iniciar el experimento abriendo `index.html` en tu navegador preferido.

Ejercicio 1

Diseña un **sencillo** protocolo:

- Debes usar alguna de las tareas que aparecen en en manual (máximo 2)
- Opcionalmente, puedes hacer primero el Ejercicio 2 de abajo, para añadir o adaptar una nueva tarea/escala **muy sencilla**
- La duración total del "experimento" no debería superar los 5 minutos

Tendrás que hacer una breve presentación contándonos el diseño experimental.

Notas

(tareas jsPsychMaker): Usa un máximo de 2 tareas

10.3 jsPsychMonkeys: Como simular datos

El sistema para simular participantes utiliza Selenium dentro de un contenedor de Docker. En Linux es trivial su uso, pero en Windows su configuración puede ser más compleja.

Puedes seguir los siguientes pasos para preparar tu sistema:

Completa el setup para tu sistema operativo

Si no funciona, te, quedan las siguientes opciones:

- Correr un par de participantes manualmente
- Usar un ordenador con Linux o crear una partición Linux
- Crear una máquina virtual linux desde la que simular participantes. Puedes usar Virtualbox para instalar Ubuntu. Una vez dentro, tendrás que seguir los pasos del manual para prepara el sistema para correr R y RStudio

Errores comunes:

- Error sobre `elevated privileges`: abre Docker desktop antes de empezar

Para lanzar monos localmente:

```
if (!require('pak')) utils::install.packages('pak'); pak::pkg_install("gorkang/jsPsychMonkey

# Un solo mono viendo su progreso
jsPsychMonkeys::release_the_monkeys(
  # Lanza un mono con el user id 5
  uid = "5",
  local_folder_tasks = "~/Downloads/protocol999",
  open_VNC = TRUE)


#  Monos del 1 al 4 simultáneamente
```

```
jsPsychMonkeys::release_the_monkeys(
  uid = "1:4",
  local_folder_tasks = "~/Downloads/protocol999",
  # Lanza los monos en paralelo
  sequential_parallel = "parallel",
  # Usando este número de CPUs
  number_of_cores = 4
)
```

Puedes ver de los parámetros disponibles en el Manual de jsPsychMonkeys. Por ejemplo, con `open_VNC = TRUE` puedes ver a los monos hacer su trabajo (siempre y cuando hayas instalado realvnc).

10.4 jsPsychHelpeR: Como preparar datos

Cada tarea de jsPsychMaker debería tener un script hermano en jsPsychHelpeR para automatizar la preparación de datos. Una vez tengamos nuestro protocolo listo para el pilotaje, con una función de jsPsychHelpeR crearemos todo lo necesario para que la preparación de datos corra automáticamente.

Para crear y abrir un nuevo proyecto de RStudio con todo listo para correr la preparación de datos de tu protocolo:

```
if (!require('pak')) utils::install.packages('pak'); pak::pkg_install("gorkang/jsPsychHelpeR

jsPsychHelpeR::run_initial_setup(pid = '999',
                                 data_location = "~/Downloads/protocol999/.data",
                                 folder = "~/Downloads/jsPsychHelpeR999")
```

En el nuevo proyecto, tendremos que correr la preparación de datos. Puedes abrir el archivo `run.R`, donde encontrarás algunas instrucciones básicas.

```
# Corremos el pipeline de preparación de datos
targets::tar_make()
```

Para ver el data frame final listo para el análisis

```
# List available objects
targets::tar_objects()

# Load DF_analysis file
targets::tar_load(DF_analysis)

# See DF_analysis data frame
DF_analysis
```

10.4.1 Como crear un reporte dentro del jsPsychHelpeR

Dentro del proyecto en el que has preparado los datos, simplemente tienes que:

---

1) Abre la plantilla report_analysis.Rmd:

```
rstudioapi::navigateToFile("Rmd/report_analysis.Rmd")
```

---

2) En el archivo `_targets.R`, en la sección análisis, descomenta las dos líneas de abajo

```
# tar_render(report_analysis, "Rmd/report_analysis.Rmd",
#            output_file = paste0("../outputs/reports/report_analysis.html")),
```

---

3) Finalmente, puedes trabajar en report_analysis.Rmd tal y como hiciste en el capítulo anterior. Cuando acabes, o quieras probar si todo funciona bien, solo tienes que correr `targets::tar_make()` desde la Consola.

Ejercicio FINAL

Ya estáis listas/os para enfrentaros al ejercicio FINAL

10.5 Avanzado

10.5.1 Como crear una nueva tarea

Tenemos un buen número de tareas disponibles para usar (puedes verlas en el manual de jsPsychR). Si la tarea que necesitas no está disponible, puedes crearla de distintas maneras:

- Modificando alguna de las tareas que ya existen: tareas en jsPsychMaker
- Usando las plantillas disponibles: `jsPsychMaker::copy_example_tasks(destination_folder = "~/Downloads/TEST")`

Veamos como crear una nueva tarea a partir de documentos excel usando las plantillas disponibles. Ver ayuda:

---

1) Copia las plantillas de tareas de ejemplo:

```
jsPsychMaker::copy_example_tasks(destination_folder = "~/Downloads/TEST")
```

2) Ve a la carpeta indicada en `destination_folder`, en este ejemplo `~/Downloads/TEST`, y borra todas las carpetas menos aquellas que correspondan al plugin que quieras usar. Por ejemplo, `Slider`.

3) Adapta `Slider.csv` a tu nueva tarea:

   - Adapta `min`, `max`, `slider_start`
   - Copia las filas existentes tantas veces como ítems necesites
   - Asegurate que los valores en la columna ID son correlativos
   - Adapta `stimulus`, `labels` a tus items

4) Adapta los .html con tus instrucciones:

   - Si necesitas más páginas de instrucciones, simplemente haz copias de las existentes
   - Edita el contenido de los archivos html

5) Ejecuta `create_protocol()` con los parámetros de abajo, se creará un nuevo protocolo con tu/tus tareas.

```
jsPsychMaker::create_protocol(
  # Incluye la tarea EAR
  canonical_tasks = "EAR",
  # Crea e incluye las tareas que estan en esta carpeta
  folder_tasks = "~/Downloads/TEST/",
  # Crea el protocolo aquí
  folder_output = "~/Downloads/TEST/new_protocol",
  # Lanza un navegador
```

```
    launch_browser = TRUE
    )
```

Ejercicio Optativo: Crear nueva tarea

Crea la siguiente tarea en jsPsychMaker:

- The Brief Resilience Scale (PDF versión inglesa), o en su versión española

Notas

(tareas jsPsychMaker): Ver instrucciones en experimentos-reproducibles - como crear una nueva tarea

Si prefieres puedes implementar una tarea distinta a The Brief Resilience Scale. Los únicos requisitos son que sea breve y sencilla.

10.5.1.1 Corrección de la tarea

Para cada tarea en jsPsychMaker, aspiramos a tener un script de corrección en jsPsychHelpeR. Si has creado una nueva tarea, por favor, completa la información de NUEVAS Tareas jsPsychR para que podamos integrar tu tarea en el repositorio común.

10.5.1.2 Como preparar datos para una tarea nueva

Tendremos que crear primero el script de preparación para la nueva tarea. En jsPsychHelpeR tienes una tarea que te ayudará con esto. Si has completado los datos en NUEVAS Tareas jsPsychR, el proceso será muy sencillo.

1) Instalamos jsPsychHelper:

```
if (!require('jsPsychHelpeR')) remotes::install_github("gorkang/jsPsychHelpeR"); library('js
```

2) Creamos el nuevo archivo `prepare_NOMBRETAREA()`:

```
jsPsychHelpeR::create_new_task(
  short_name_task = "NAMETASK",
  get_info_googledoc = TRUE
  )
```

Esta función:

1. Creará un nuevo archivo de corrección a partir de la plantilla
2. Lo adaptará para que funciones con el nombre que le has asignado a la tarea

3. Abrirá el archivo para que lo puedas editar

Si hay información en todas las pestañas de NUEVAS Tareas jsPsychR, en la consola se mostrará información lista para copiar y pegar en tu script sobre:

- nombres de dimensiones
- ítems para cada dimensión
- cálculo de dimensiones
- ítems invertidos
- conversión numérica

11 Ejercicios

11.1 Ejercicio FINAL

Para el ejercicio final (evaluable), usaremos el protocolo que has creado en el ejercicio anterior. Usando datos simulados, con la ayuda de jsPsychHelpeR vamos a crear un proyecto de RStudio que procese los datos automáticamente, y adaptar un archivo Rmarkdown para generar un reporte automatizado con tablas, gráficas y una descripción semi-automática de nuestros resultados simulados.

1) El primer paso consiste en obtener datos de participantes virtuales.

- Podéis intentarlo simulando datos
- Si no os funciona jsPsychMonkeys en vuestro sistema, el profesor os ayudará a simular datos para vuestro experimento.
- Alternativamente, podéis correr el experimento un par de veces (abriendo el index.html)

2) Una vez tengamos los datos:

- Crearemos un proyecto que procesará los datos automáticamente usando jsPsychHelpeR - como preparar datos
- Crearemos un reporte en Rmd como parte del pipeline de jsPsychHelpeR donde incluiremos:
  - Tabla/s con descriptivos ver ayuda
  - Gráfico/s con resultado ver ayuda y ayuda
  - Tabla/s con resultados de un análisis sencillo ver ayuda
  - Una frase reportando resultados del análisis (usando Texto inline de gtsummary)

El proyecto tiene que correr en cualquier computador.

IMPORTANTE

La nota del workshop estará basada en el resultado de esta tarea.

Tendréis que compartir el proyecto completo con el profesor, y él deberá poder correrlo y ver como resultado el reporte en pdf incluyendo los elementos detallados arriba.

Notas

(simulación datos): Sigue las instrucciones de simulando datos

(preparando datos): Ver como preparar datos

Para más información, ver el manual de jsPsychR

Paquetes usados

En la documentación y ejercicios de este workshop se usaron los paquetes que se pueden ver abajo. Este listado se creó automáticamente usando {grateful}:

```
grateful::cite_packages(pkgs = "All", output = "paragraph", out.format = "Rmd", include.RStu
```

We used R version 4.3.3 (R Core Team 2024) and the following R packages: afex v. 1.3.1 (Singmann et al. 2024), car v. 3.1.2 (Fox and Weisberg 2019), correlation v. 0.8.5 [@], corrplot v. 0.92 (Wei and Simko 2021), corrr v. 0.4.4 (Kuhn, Jackson, and Cimentada 2022), cowplot v. 1.1.3 (Wilke 2024a), DT v. 0.33 (Xie, Cheng, and Tan 2024), esquisse v. 2.0.0 (Meyer and Perrier 2024), gapminder v. 1.0.0 (Bryan 2023), geomtextpath v. 0.1.4 (Cameron and van den Brand 2024), gghighlight v. 0.4.1 (Yutani 2023), ggrain v. 0.0.4 (Allen et al. 2021), ggraph v. 2.2.1 (Pedersen 2024), ggridges v. 0.5.6 (Wilke 2024b), ggtext v. 0.1.2 (Wilke and Wiernik 2022), ggthemes v. 5.1.0 (Arnold 2024), gt v. 0.11.0 (Iannone et al. 2024), gtsummary v. 1.7.2 (Sjoberg et al. 2021), here v. 1.0.1 (Müller 2020), inspectdf v. 0.0.12 (Rushworth 2022), janitor v. 2.2.0 (Firke 2023), jsPsychHelpeR v. 0.3.5.903 (Navarrete 2024a), jsPsychMaker v. 0.3.5.901 (Navarrete and Valencia 2024), jsPsychMonkeys v. 0.3.5.901 (Navarrete 2024b), knitr v. 1.48 (Xie 2014, 2015, 2024a), lme4 v. 1.1.35.5 (Bates et al. 2015), pak v. 0.7.2 (Csárdi and Hester 2024), papaja v. 0.1.2 (Aust and Barth 2023), parameters v. 0.22.1 (Lüdecke et al. 2020), performance v. 0.12.2 (Lüdecke, Ben-Shachar, et al. 2021), plotly v. 4.10.4 (Sievert 2020), quarto v. 1.4.4 (Allaire and Dervieux 2024), RColorBrewer v. 1.1.3 (Neuwirth 2022), readODS v. 2.3.0 (Schutten et al. 2024), regexplain v. 0.2.2.9000 (Aden-Buie 2024), remotes v. 2.5.0 (Csárdi et al. 2024), renv v. 1.0.7 (Ushey and Wickham 2024), report v. 0.5.9 (Makowski et al. 2023), rmarkdown v. 2.27 (Xie, Allaire, and Grolemund 2018; Xie, Dervieux, and Riederer 2020; Allaire, Xie, Dervieux, McPherson, et al. 2024), rticles v. 0.27 (Allaire, Xie, Dervieux, R Foundation, et al. 2024), scales v. 1.3.0 (Wickham, Pedersen, and Seidel 2023), see v. 0.8.5 (Lüdecke, Patil, et al. 2021), sjPlot v. 2.8.16 (Lüdecke 2024), skimr v. 2.1.5 (Waring et al. 2022), stringi v. 1.8.4 (Gagolewski 2022), tidyverse v. 2.0.0 (Wickham et al. 2019), tinytex v. 0.52 (Xie 2019, 2024b), usethis v. 2.2.3 (Wickham et al. 2024), waldo v. 0.5.2 (Wickham 2023), writexl v. 1.5.0 (Ooms 2024).

References

Aden-Buie, Garrick. 2024. *regexplain: Rstudio Addin to Explain, Test and Build Regular Expressions*. https://github.com/gadenbuie/regexplain.

Allaire, JJ, and Christophe Dervieux. 2024. *quarto: R Interface to "Quarto" Markdown Publishing System*. https://github.com/quarto-dev/quarto-r.

Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2024. *rmarkdown: Dynamic Documents for r*. https://github.com/rstudio/rmarkdown.

Allaire, JJ, Yihui Xie, Christophe Dervieux, R Foundation, Hadley Wickham, Journal of Statistical Software, Ramnath Vaidyanathan, et al. 2024. *rticles: Article Formats for r Markdown*. https://CRAN.R-project.org/package=rticles.

Allen, Micah, Davide Poggiali, Kirstie Whitaker, Tom Rhys Marshall, Jordy van Langen, and Rogier A. Kievit. 2021. "Raincloud Plots: A Multi-Platform Tool for Robust Data Visualization [Version 2; Peer Review: 2 Approved]." *Wellcome Open Research* 4 (63). https://doi.org/10.12688/wellcomeopenres.15191.2.

Arnold, Jeffrey B. 2024. *ggthemes: Extra Themes, Scales and Geoms for "ggplot2"*. https://CRAN.R-project.org/package=ggthemes.

Aust, Frederik, and Marius Barth. 2023. *papaja: Prepare Reproducible APA Journal Articles with R Markdown*. https://github.com/crsh/papaja.

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. https://doi.org/10.18637/jss.v067.i01.

Bryan, Jennifer. 2023. *gapminder: Data from Gapminder*. https://github.com/jennybc/gapminder.

Cameron, Allan, and Teun van den Brand. 2024. *geomtextpath: Curved Text in "ggplot2"*. https://allancameron.github.io/geomtextpath/.

Csárdi, Gábor, and Jim Hester. 2024. *pak: Another Approach to Package Installation*. https://pak.r-lib.org/.

Csárdi, Gábor, Jim Hester, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. 2024. *remotes: R Package Installation from Remote Repositories, Including "GitHub"*. https://remotes.r-lib.org.

Firke, Sam. 2023. *janitor: Simple Tools for Examining and Cleaning Dirty Data*. https://github.com/sfirke/janitor.

Fox, John, and Sanford Weisberg. 2019. *An R Companion to Applied Regression*. Third. Thousand Oaks CA: Sage. https://socialsciences.mcmaster.ca/jfox/Books/Companion/.

Gagolewski, Marek. 2022. "stringi: Fast and Portable Character String Processing in R." *Journal of Statistical Software* 103 (2): 1–59. https://doi.org/10.18637/jss.v103.i02.

Iannone, Richard, Joe Cheng, Barret Schloerke, Ellis Hughes, Alexandra Lauer, JooYoung Seo, Ken Brevoort, and Olivier Roy. 2024. *gt: Easily Create Presentation-Ready Display Tables*. https://gt.rstudio.com.

Kuhn, Max, Simon Jackson, and Jorge Cimentada. 2022. *corrr: Correlations in r*. https://CRAN.R-project.org/package=corrr.

Lüdecke, Daniel. 2024. *sjPlot: Data Visualization for Statistics in Social Science*. https://CRAN.R-project.org/package=sjPlot.

Lüdecke, Daniel, Mattan S. Ben-Shachar, Indrajeet Patil, and Dominique Makowski. 2020. "Extracting, Computing and Exploring the Parameters of Statistical Models Using R." *Journal of Open Source Software* 5 (53): 2445. https://doi.org/10.21105/joss.02445.

Lüdecke, Daniel, Mattan S. Ben-Shachar, Indrajeet Patil, Philip Waggoner, and Dominique Makowski. 2021. "performance: An R Package for Assessment, Comparison and Testing of Statistical Models." *Journal of Open Source Software* 6 (60): 3139. https://doi.org/10.21105/joss.03139.

Lüdecke, Daniel, Indrajeet Patil, Mattan S. Ben-Shachar, Brenton M. Wiernik, Philip Waggoner, and Dominique Makowski. 2021. "see: An R Package for Visualizing Statistical Models." *Journal of Open Source Software* 6 (64): 3393. https://doi.org/10.21105/joss.03393.

Makowski, Dominique, Daniel Lüdecke, Indrajeet Patil, Rémi Thériault, Mattan S. Ben-Shachar, and Brenton M. Wiernik. 2023. "Automated Results Reporting as a Practical Tool to Improve Reproducibility and Methodological Best Practices Adoption." *CRAN*. https://easystats.

github.io/report/.
Meyer, Fanny, and Victor Perrier. 2024. *esquisse: Explore and Visualize Your Data Interactively*. https://dreamrs.github.io/esquisse/.
Müller, Kirill. 2020. *here: A Simpler Way to Find Your Files*. https://CRAN.R-project.org/package=here.
Navarrete, Gorka. 2024a. *jsPsychHelpeR: Standardize and Automatize Data Preparation and Analysis of jsPsych Experiments Created with jsPsychMaker*. https://github.com/gorkang/jsPsychHelpeR.
———. 2024b. *jsPsychMonkeys: Release Monkeys to a jsPsych Experiment Using the r Package targets, Docker and RSelenium*. https://github.com/gorkang/jsPsychMonkeys.
Navarrete, Gorka, and Herman Valencia. 2024. *jsPsychMaker: Create Behavioral Experiments and Surveys Using jsPsych and r*. https://github.com/gorkang/jsPsychMaker.
Neuwirth, Erich. 2022. *RColorBrewer: ColorBrewer Palettes*.
Ooms, Jeroen. 2024. *writexl: Export Data Frames to Excel "xlsx" Format*. https://CRAN.R-project.org/package=writexl.
Pedersen, Thomas Lin. 2024. *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*. https://CRAN.R-project.org/package=ggraph.
R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.
Rushworth, Alastair. 2022. *inspectdf: Inspection, Comparison and Visualisation of Data Frames*. https://CRAN.R-project.org/package=inspectdf.
Schutten, Gerrit-Jan, Chung-hong Chan, Peter Brohan, Detlef Steuer, and Thomas J. Leeper. 2024. *readODS: Read and Write ODS Files*. https://docs.ropensci.org/readODS/.
Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with r, Plotly, and Shiny*. Chapman; Hall/CRC. https://plotly-r.com.
Singmann, Henrik, Ben Bolker, Jake Westfall, Frederik Aust, and Mattan S. Ben-Shachar. 2024. *afex: Analysis of Factorial Experiments*. https://afex.singmann.science/.
Sjoberg, Daniel D., Karissa Whiting, Michael Curry, Jessica A. Lavery, and Joseph Larmarange. 2021. "Reproducible Summary Tables with the Gtsummary Package." *The R Journal* 13: 570–80. https://doi.org/10.32614/RJ-2021-053.
Ushey, Kevin, and Hadley Wickham. 2024. *renv: Project Environments*. https://CRAN.R-project.org/package=renv.
Waring, Elin, Michael Quinn, Amelia McNamara, Eduardo Arino de la Rubia, Hao Zhu, and Shannon Ellis. 2022. *skimr: Compact and Flexible Summaries of Data*. https://docs.ropensci.org/skimr/ (website).
Wei, Taiyun, and Viliam Simko. 2021. *R Package "corrplot": Visualization of a Correlation Matrix*. https://github.com/taiyun/corrplot.
Wickham, Hadley. 2023. *waldo: Find Differences Between r Objects*. https://waldo.r-lib.org.
Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.
Wickham, Hadley, Jennifer Bryan, Malcolm Barrett, and Andy Teucher. 2024. *usethis: Automate Package and Project Setup*. https://usethis.r-lib.org.
Wickham, Hadley, Thomas Lin Pedersen, and Dana Seidel. 2023. *scales: Scale Functions for Visualization*. https://scales.r-lib.org.
Wilke, Claus O. 2024a. *cowplot: Streamlined Plot Theme and Plot Annotations for "ggplot2"*. https://CRAN.R-project.org/package=cowplot.
———. 2024b. *ggridges: Ridgeline Plots in "ggplot2"*. https://CRAN.R-project.org/package=ggridges.

Wilke, Claus O., and Brenton M. Wiernik. 2022. *ggtext: Improved Text Rendering Support for "ggplot2"*. https://wilkelab.org/ggtext/.

Xie, Yihui. 2014. "knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

———. 2019. "TinyTeX: A Lightweight, Cross-Platform, and Easy-to-Maintain LaTeX Distribution Based on TeX Live." *TUGboat* 40 (1): 30–32. https://tug.org/TUGboat/Contents/contents40-1.html.

———. 2024a. *knitr: A General-Purpose Package for Dynamic Report Generation in r*. https://yihui.org/knitr/.

———. 2024b. *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. https://github.com/rstudio/tinytex.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Xie, Yihui, Joe Cheng, and Xianying Tan. 2024. *DT: A Wrapper of the JavaScript Library "DataTables"*. https://CRAN.R-project.org/package=DT.

Xie, Yihui, Christophe Dervieux, and Emily Riederer. 2020. *R Markdown Cookbook*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown-cookbook.

Yutani, Hiroaki. 2023. *gghighlight: Highlight Lines and Points in "ggplot2"*. https://CRAN.R-project.org/package=gghighlight.