Herramientas del BIG DATA y analytics

Empezando a visualizar datos con R y *ggplot2*

Segunda Edición

Autores:
Julio César Alonso
Maria Fernanda Largo

Editorial Universidad Icesi
UNIVERSIDAD ICESI

Empezando a visualizar datos con R y ggplot2

Segunda Edición

Julio César Alonso[1] Maria Fernanda Largo[2]

31 de agosto de 2023

[1]CIENFI - Universidad Icesi, jcalonso@icesi.edu.co
[2]CIENFI - Universidad Icesi, mflargo@icesi.edu.co

Julio César Alonso C. y Maria Fernanda Largo L.

Colección «Herramientas del Big Data y Analytics», vol. 3

Cali. Universidad Icesi, 2023.

155 páginas.

Incluye referencias bibliográficas.

ISBN: 978-628-7630-14-7 (eBook).

DOI: https://doi.org/10.18046/EUI/bda.h.3.2

Palabras Clave: 1. R | 2. Analítica | 3. ggplot | 4. Visualizaciones | 5. Big Data Analytics

Clasificación Dewey: 545 ddc 21

CIENFI - Centro de Investigación en Economía y Finanzas

www.icesi.edu.co/centros-academicos/cienfi

Rector: Esteban Piedrahita Uribe

Secretaria General: Olga Patricia Ramírez Restrepo

Director Académico: José Hernando Bahamón

Coordinador editorial: Adolfo A. Abadía

Corrección de estilo: Claudia L. González G.

Diseño de portada: Sandra Moreno

Fotos tomadas por: Julio César Alonso

Editorial Universidad Icesi

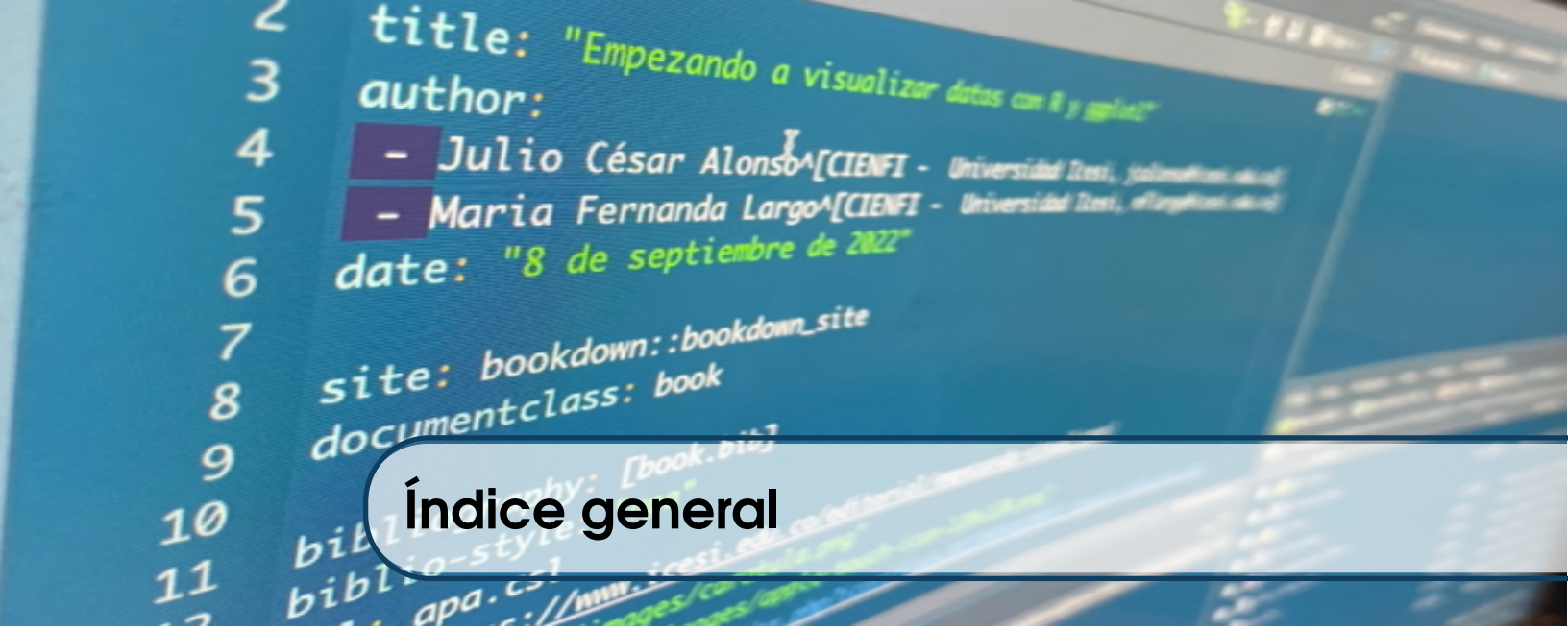
Calle 18 No. 122-135 (Pance), Cali – Colombia

Teléfono: +57 (2) 555 2334 | E-mail: editorial@icesi.edu.co

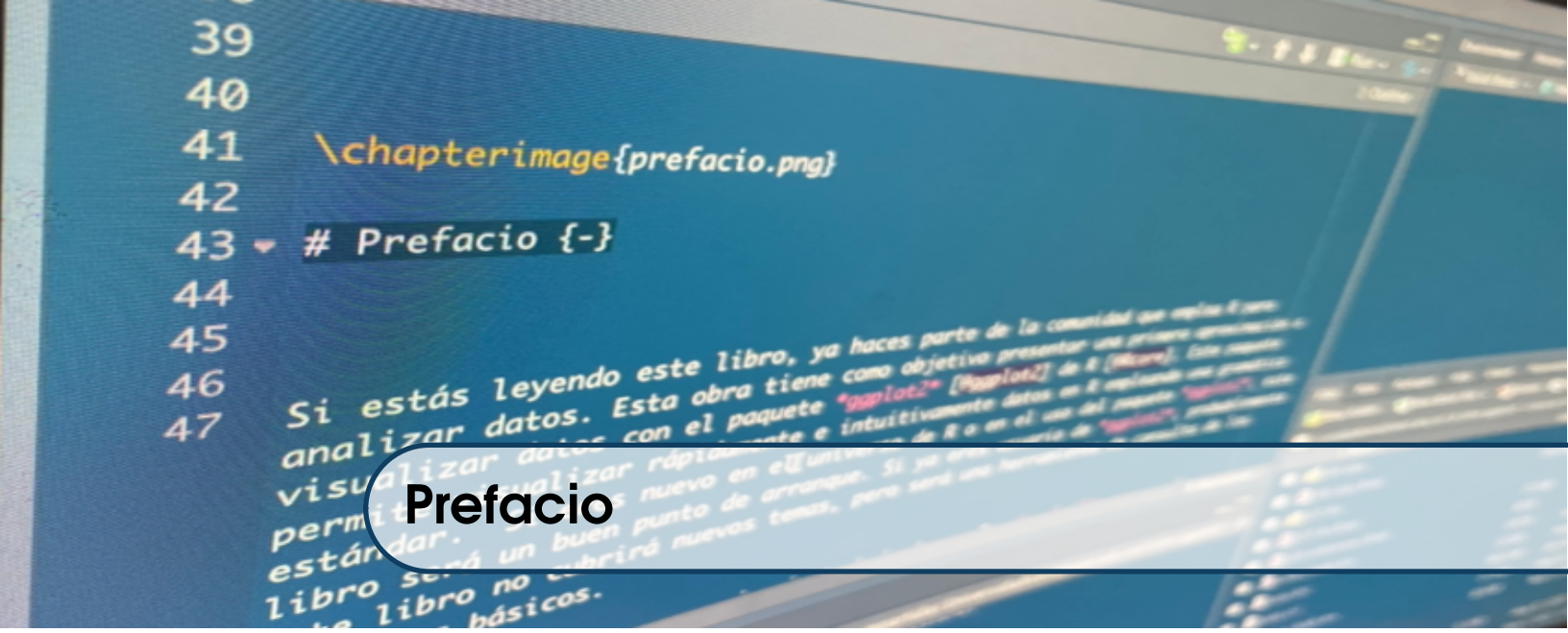
http://www.icesi.edu.co/editorial

Publicado en Colombia – *Published in Colombia*

La publicación de este libro se aprobó luego de superar un proceso de evaluación doble ciego.



Índice general



Prefacio

Si estás leyendo este libro, ya haces parte de la comunidad que emplea R para analizar datos. Esta obra tiene como objetivo presentar una primera aproximación a visualizar datos con el paquete *ggplot2* (Wickham, 2016) de R (R Core Team, 2018). Este paquete permite visualizar rápidamente e intuitivamente datos en R empleando una gramática estándar. Si eres nuevo en el universo de R o en el uso del paquete *ggplot2*, este libro será un buen punto de arranque. Si ya eres usuario de *ggplot2*, te recomendamos revisar los Capítulos 8 y 9, donde encontrarás material más avanzado.

Aquí se recoge nuestra experiencia trabajando con R y *ggplot2* para resolver problemas con datos desde el CIENFI (Centro de Investigación en Economía y Finanzas) de la Universidad Icesi. En el CIENFI, empleamos R para la transformación de datos en conclusiones que faciliten la toma de decisiones en organizaciones privadas y públicas. Toda esta experiencia la queremos plasmar en esta obra para asegurar que nuevas generaciones de profesionales continúen fortaleciendo la comunidad de R alrededor del mundo.

Este libro también refleja la evolución del paquete y conocimiento que hemos recogido en el CIENFI desde la primera guía introductoria al paquete publicada en 2012 (Alonso y González, 2012). Los comentarios de sus lectores y estudiantes nos han motivado para escribir este libro.

Este es el tercero de una serie de libros introductorios al uso de R. El primer libro (Alonso y Ocampo, 2022) presenta una breve introducción para iniciar a usar R. En él se enseña cómo instalar R, la interfaz RStudio y paquetes, cómo cargar diferentes bases de datos y cómo realizar operaciones aritméticas y lógicas con objetos. También se discuten las clases esenciales de objetos sencillos y compuestos. No dudes en consultar ese primer libro si aún no has iniciado tu camino por el universo de R. Lo

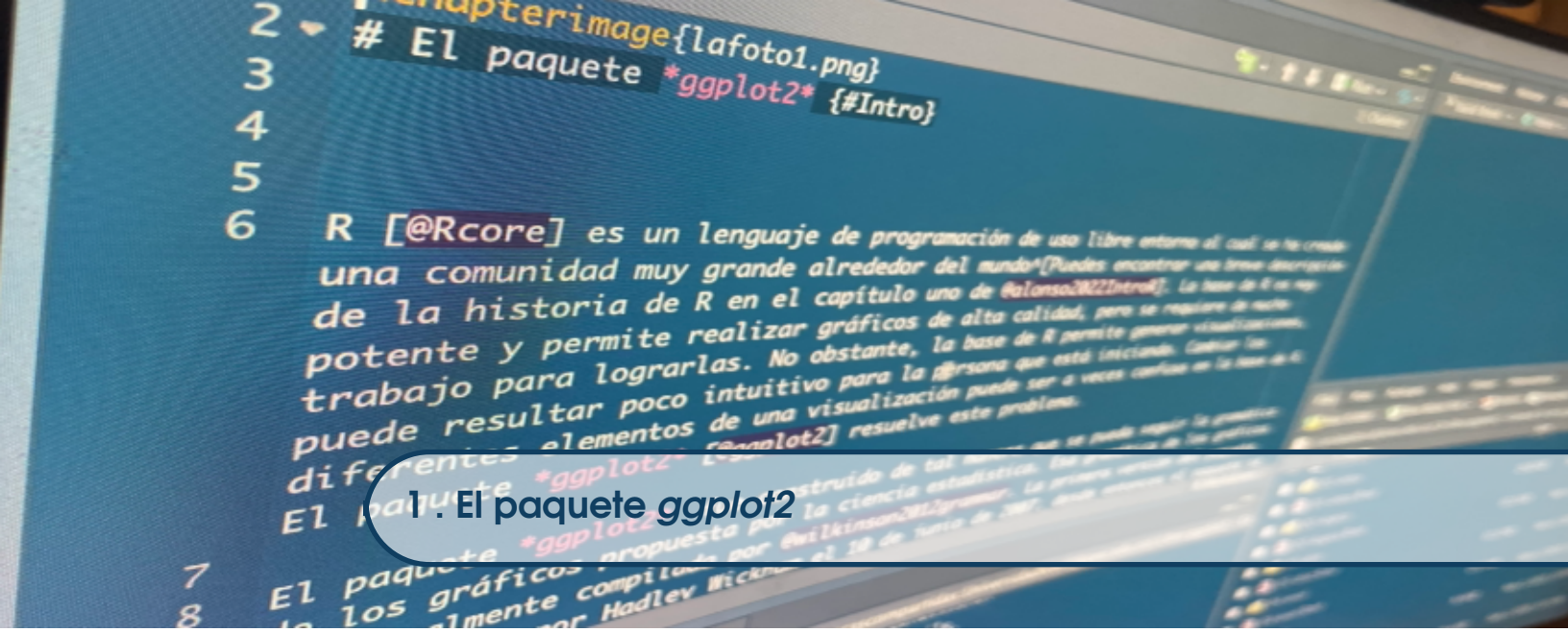
puedes encontrar en: https://www.icesi.edu.co/editorial/empezando-usar/ .

El segundo libro de la serie (Alonso, 2022) corresponde a una introducción al paquete *dplyr* (Wickham et al., 2023a), el cual permite manipular objetos que contengan datos. En esa obra mostramos cómo filtrar observaciones, crear nuevas variables y combinar objetos con datos. Es recomendable tener un conocimiento de ese paquete antes de leer esta obra. Consulta ese segundo libro si aún no has tenido alguna experiencia manipulando objetos con datos con *dplyr*. Lo puedes encontrar en el siguiente enlace: http://www.icesi.edu.co/editorial/empezando-transformar/ .

En este tercer libro de la serie nos concentraremos en cómo visualizar datos empleando el paquete *ggplot2*. En el flujo de trabajo que nos lleva de los datos a la toma de decisiones, las visualizaciones son importantes en la exploración de los datos y en la comunicación de los resultados. Es ahí dónde nos centraremos en este libro; en escoger la mejor visualización de los datos que tengamos.

Esta es la segunda edición de este libro. Esta edición la hemos construido gracias a los comentarios recibidos. Hemos adicionado dos capítulos con temas un poco más avanzados que el resto del libro. En el Capítulo 8 presentamos una discusión de cómo personalizar nuestras visualizaciones empleando la capa de ***Tema***. Y en el Capítulo 9 discutimos más geometrías para visualizar datos.

¡Esperamos encuentres esta obra útil y la compartas con otros futuros usuarios interesados! Si tienes alguna sugerencia del libro o corrección, no dudes en escribirme. Esta es una obra en constante construcción.



1 El paquete *ggplot2*

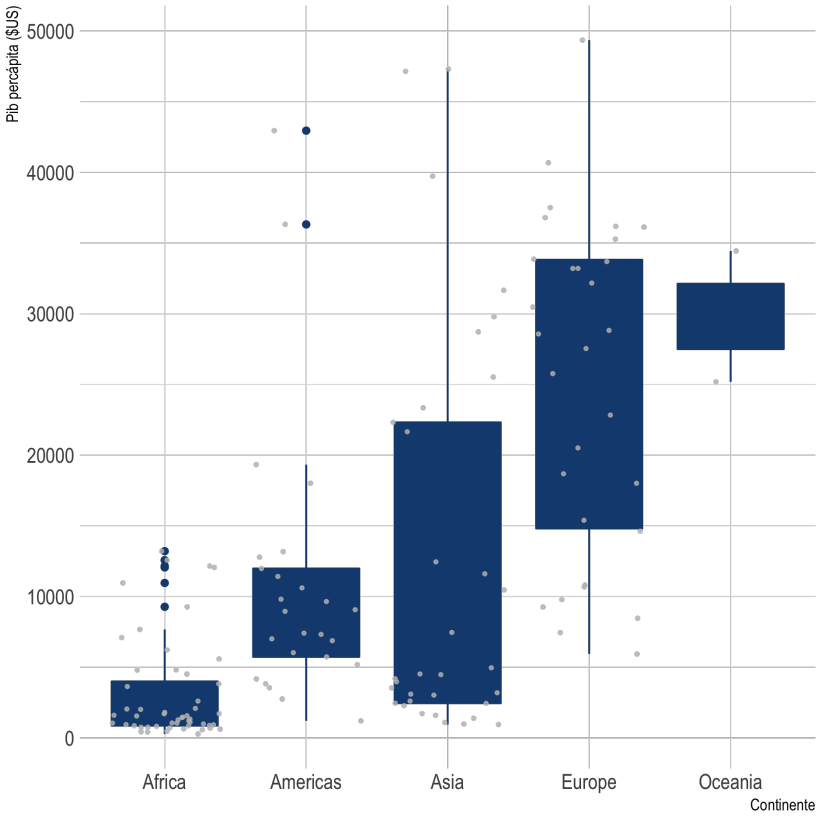
R (R Core Team, 2018) es un lenguaje de programación de uso libre entorno al cual se ha creado una comunidad muy grande alrededor del mundo[1]. La base de R es muy potente y permite realizar gráficos de alta calidad, pero se requiere de mucho trabajo para lograrlas. No obstante, la base de R permite generar visualizaciones, puede resultar poco intuitivo para la persona que está iniciando. Cambiar los diferentes elementos de una visualización puede ser a veces confuso en la base de R. El paquete *ggplot2* (Wickham, 2016) resuelve este problema.

El paquete *ggplot2* fue construido de tal manera que se pueda seguir la gramática de los gráficos propuesta por la ciencia estadística. Esa gramática de los gráficos fue finalmente compilada por Wilkinson (2012). La primera versión del paquete fue liberada por Hadley Wickham el 10 de junio de 2007. Desde entonces el paquete se ha enriquecido con diferentes elementos. En los últimos años, *ggplot2* se ha convertido en el paquete de creación de visualizaciones más popular en el universo R por permitir de manera sencilla obtener gráficos de alta calidad. Por ejemplo, en las Figuras 1.1, 1.2 y 1.3 vemos unas visualizaciones creadas con *ggplot2* empleando un par de líneas.

La Figura 1.1 presenta un diagrama de cajas (o también conocido como *Boxplot*) que además incluye cada una de las observaciones por grupo; en el Capítulo 3 explicaremos cómo interpretar y construir estas visualizaciones.

[1] Puedes encontrar una breve descripción de la historia de R en el Capítulo 1 de Alonso y Ocampo (2022)

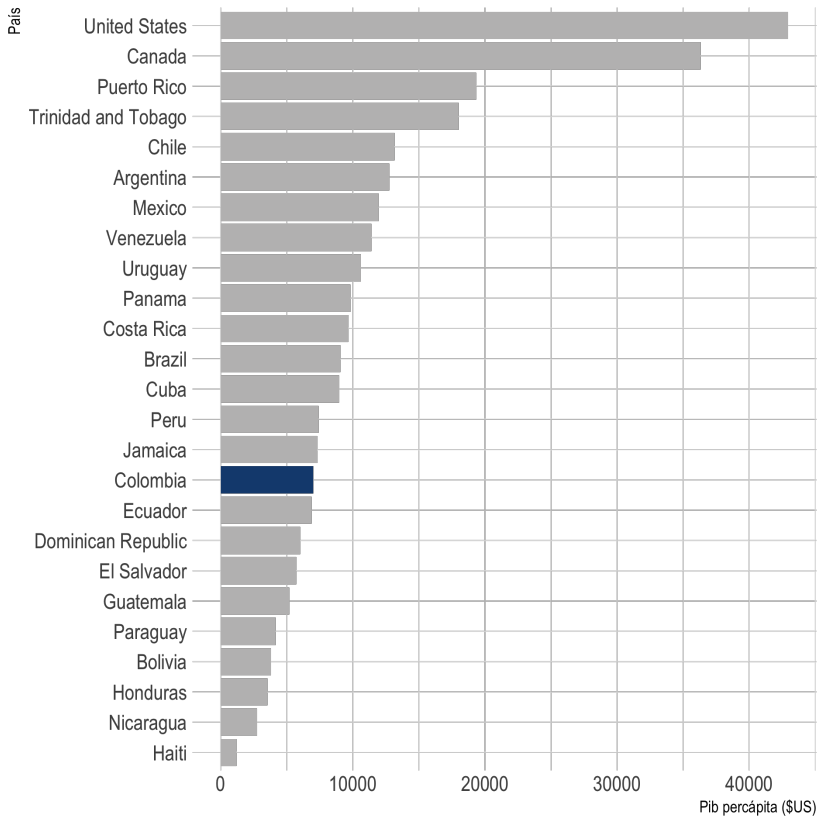
Figura 1.1. Distribución del PIB percápita por continente (2007)



Fuente: Datos de gapminder y cálculos propios.

La Figura 1.2 presenta un gráfico de barras tradicional que aprenderemos a construir en el Capítulo 3. Ese gráfico lo hemos "embellecido" empleando unos trucos que estudiaremos en el Capítulo 6.

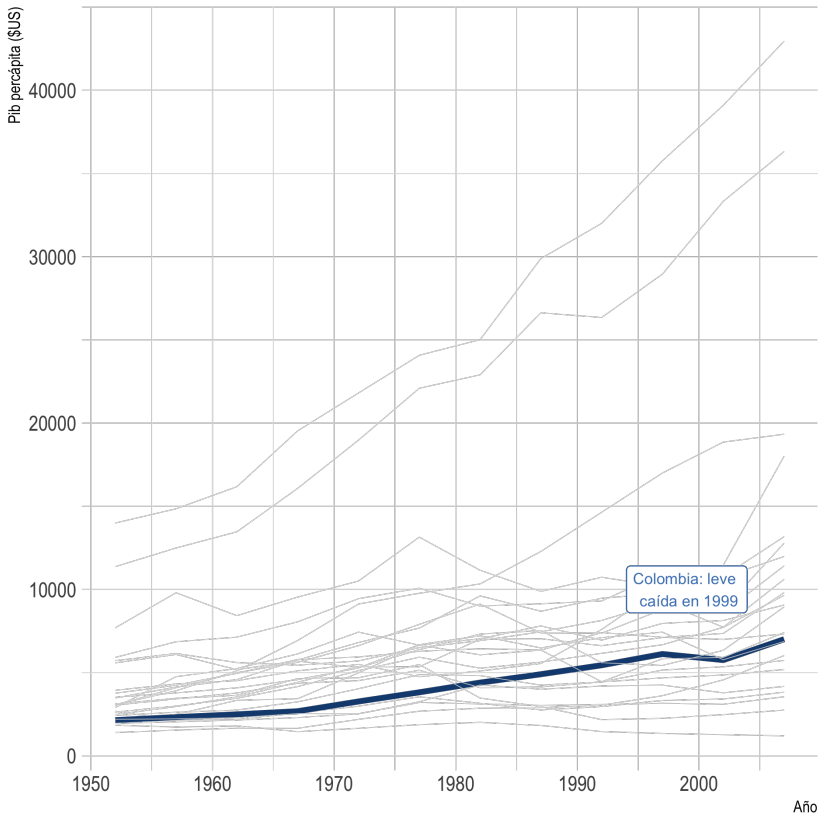
Figura 1.2. PIB percápita de los países del continente Americano (2007)



Fuente: Datos de gapminder y cálculos propios.

La Figura 1.3 presenta un gráfico de líneas con una anotación que permite resaltar uno de los casos. El gráfico de líneas lo estudiaremos en el Capítulo 4, y en el Capítulo 6 veremos como hacer las anotaciones y resaltar una sola línea.

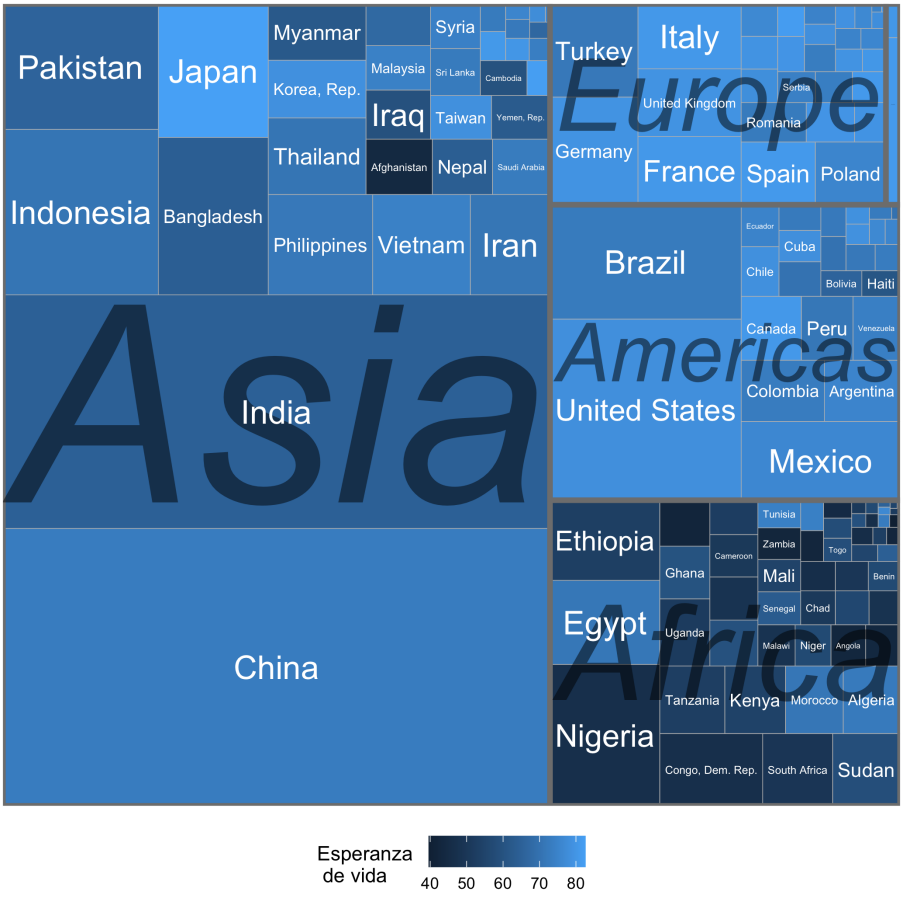
Figura 1.3. Evolución del PIB percápita de Colombia y otros países de las Américas



Fuente: Datos de gapminder y cálculos propios.

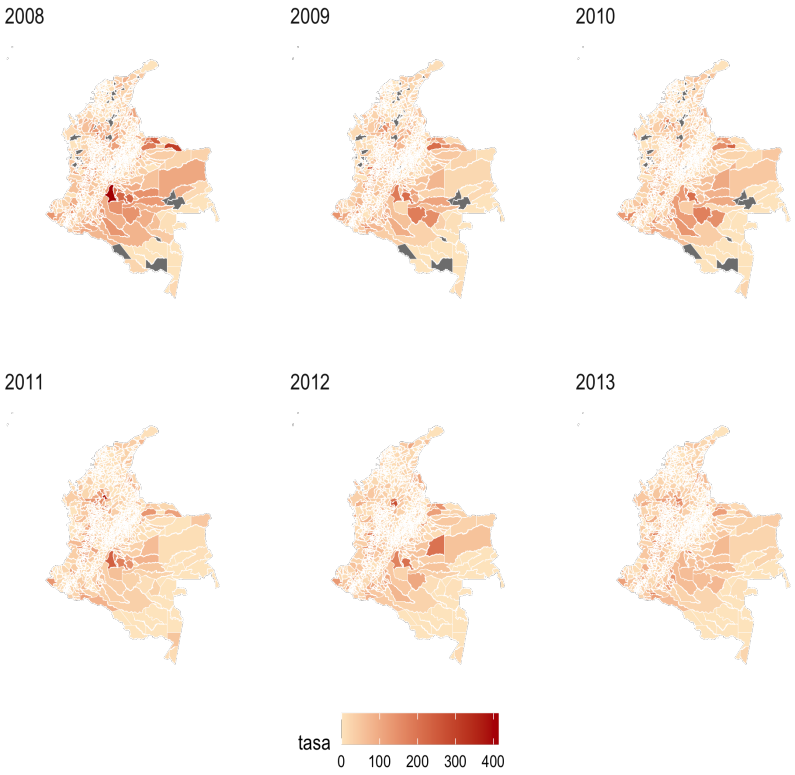
También existen numerosos paquetes que emplean la lógica de *ggplot2* para ampliar sus funcionalidades, permitiendo nuevos tipos de visualizaciones. Por ejemplo, el paquete *treemapify* (Wilkins, 2021) con el que se construyó el *treemap*, reportado en la Figura 1.4. Otro ejemplo se presenta en al Figura 1.5, que fue creada con los paquetes *colmaps* (Moreno, 2015a) y *homicidios* (Moreno, 2015b) . En el Capítulo 2 estudiaremos la lógica detrás de la gramática de las visualizaciones que emplea *ggplot2* y que podrás aplicar para otros paquetes como *treemapify* y *colmaps*.

Figura 1.4. Composición de la población mundial y esperanza de vida al nacer por país (2007)



Fuente: Datos de gapminder y cálculos propios empleando el paquete treemapify.

Figura 1.5. Evolución de la tasa municipal de homicidios (por mil habitantes)



Fuente: Datos del paquete homicidios y cálculos propios empleando el paquete colmaps.

Todas las visualizaciones, en últimas, obedecen a una "gramática", como lo demuestra Wilkinson (2012). El paquete *ggplot2* facilita la construcción de las visualizaciones permitiendo implementar dicha gramática en R. Precisamente, al usar esta gramática, el paquete *ggplot2* es más intuitivo para el usuario que apenas está empezando en el universo de R.

En el Capítulo 2 analizaremos los elementos básicos de la gramática de los gráficos. En los Capítulos 3, 4 y 5 veremos los diferentes tipos de gráficos (que en este mundo se conoce como la geometría) y en el Capítulo 6 discutiremos unos trucos para mejorar la comunicación de nuestras visualizaciones.

Antes de entrar en los detalles de cómo realizar visualizaciones es importante entender la lógica que siguió el diseñador del paquete *ggplot2*. De hecho, este paquete hace parte de un conjunto de paquetes que se conocen como *tidyverse*[2]. Estos fueron diseñados para facilitar operaciones comunes de la ciencia de datos, permitiendo un flujo de trabajo continuo entre las diferentes tareas de carga, transformación, modelado y visualización de datos.

1.1 El universo *tidyverse*

Los paquetes que hacen parte de *tidyverse*[3] tienen funciones que no solo permiten realizar tareas como la carga, transformación y visualización de datos, sino también la conexión entre dichas tareas[4] (ver Sección 1.2).

Estos paquetes permiten optimizar el flujo de trabajo cuando empleamos datos para sacar conclusiones (ver Figura 1.6). El flujo de trabajo inicia[5] desde la preparación y limpieza de los datos que previamente han sido recolectados y almacenados. Posteriormente se exploran los datos de manera gráfica y con estadísticas descriptivas, para pasar al modelado. Finalmente se comunican los resultados empleando visualizaciones y tablas.

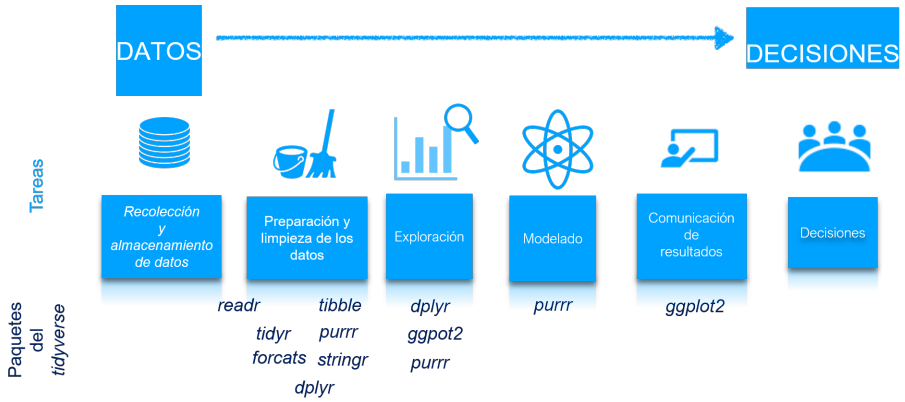
[2] *tidyverse* (Wickham et al., 2019) es un paquete que contiene 8 paquetes.

[3] Observa que *tidy* significa en español ordenado y *verse* es la parte final de *universe* (universo); de ahí que también se emplee la expresión universo *Tidyverse*.

[4] Si deseas conocer mucho más del detalle de cómo fue diseñado este universo puedes consultar Wickham y Grolemund (2016) o la versión en línea del libro en el siguiente enlace: https://r4ds.had.co.nz.

[5] Para una explicación breve de las actividades en el proceso de análisis de datos puedes ver el video en el siguiente enlace: https://youtu.be/rhLWa-vOxyU .

Figura 1.6. Actividades y paquetes del Tidyverse en el flujo de trabajo del análisis de datos



Fuente: Adaptación de Alonso y Ocampo (Alonso Cifuentes y Ocampo, 2022).

Los paquetes del *tidyverse*[6] son:

- *readr* (Wickham et al., 2018): lee datos de muchas fuentes (incluyendo formatos como .tsv y .fwf). Este paquete es útil para pasar de la actividad de recolección y almacenamiento de datos a la de preparación y limpieza (ver Figura 1.6).
- *tibble* (Müller y Wickham, 2021): guarda bases de datos de la clase *tibble* (ver Capítulo 1 de Alonso (2022) para una discusión de esta clase de objetos). Las funciones de este paquete son útiles en la actividad de preparación y limpieza de datos (ver Figura 1.6).
- *tidyr* (Wickham et al., 2023b): permite organizar un objeto con datos de clase **data.frame** o **tibble**. Este paquete es de especial utilidad en la actividad de preparación de datos (ver Figura 1.6).
- *stringr* (Wickham, 2019): facilita el trabajo con datos que contienen carácteres (texto), lo cuál es conveniente en la limpieza de datos (ver Figura 1.6).
- *forcats* (Wickham, 2020): facilita la preparación y limpieza de variables de clase **factor** (ver Figura 1.6).
- *purrr* (Henry y Wickham, 2020): facilita la programación con funciones y vectores. Estas herramientas permiten eliminar los bucles (también conocidos como *loops*[7]), que son útiles en la ejecución de tareas repetitivas en la actividad de preparación, limpieza, exploración y modelado de datos (ver Figura 1.6).
- *dplyr* (Wickham et al., 2023a): facilita el filtrado de observaciones,

[6]Para una descripción más detallada de cada uno de los paquetes se puede consultar Alonso (2022) .

[7]Si quieres conocer sobre los *loops* en R, puedes consultar Alonso (2021) .

creación de variables y unión de objetos con datos por medio de una gramática de manipulación de datos. Para una discusión de este paquete puedes ver Alonso (2022). Este paquete es conveniente en la preparación, limpieza y exploración de los datos (ver Figura 1.6).
- *ggplot2* (Wickham, 2016): crea visualizaciones de datos siguiendo un gramática de capas, las cuales son útiles en las actividades de limpieza y exploración de datos y en la de comunicación de los resultados (ver Figura 1.6). Este libro se centra en este paquete.

Podemos emplear todas las funcionalidades del paquete *ggplot2* sin necesidad de emplear los otros paquetes del *tidyverse*. Pero, en algunas ocasiones, emplear los otros paquetes de universo *tidyverse* simplifica el flujo de trabajo desde los datos originales hasta obtener la visualización. En este libro mantendremos al mínimo el uso de los otros paquetes del *tidyverse*; sin embargo, nos será difícil no emplearlos. En otras palabras, si bien evitaremos emplear el operador `pipe` (`%>%`), en algunas ocasiones será difícil no emplearlo para agilizar nuestro flujo de trabajo. Así mismo ocurrirá con los verbos de los otros 7 paquetes del *tidyverse*.

1.2 El operador `pipe`

Los paquetes del universo *tidyverse* emplean el operador `pipe` (`%>%`). Este operador permite poner en una "tubería" un objeto con datos y ejecutar diferentes operaciones sobre las observaciones (filas) o variables (columnas) sin tener que guardar los resultados intermedios. En otras palabras, el operador `%>%` permite encadenar resultados rápidamente. En Alonso (2022) encontrarás una presentación de este operador más detallada.

Veamos un ejemplo que emplea los datos del paquete *gapminder* (Bryan, 2017) (ver Capítulo 6 de Alonso y Ocampo (2022)). Este paquete tiene un objeto con datos con el mismo nombre, que contiene las siguientes variables: país (`country`), continente (`continent`), año (`year`), esperanza de vida al nacer (`lifeExp`), población (`pop`) y PIB percápita (`gdpPercap`). Los datos van desde 1952 a 2007 de a quinquenios. Carguemos los datos y explorémoslos.

```
# cargar paquete
# install.packages("gapminder")
library(gapminder)
# cargar datos
data("gapminder")
# cargar paquete
library(dplyr)
```

```
# Mirando los datos
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan"~
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, A~
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1~
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.~
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079~
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739~
```

Supongamos que solo queremos trabajar con los promedios por continentes de la esperanza de vida al nacer para el 2007. Hacer esto con la base de R nos tomará bastantes pasos, pero con el paquete *dplyr* y el operador %>%, la tarea se simplifica mucho. Por ejemplo:

```
library(dplyr)
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarise(EV_prom = mean(lifeExp))
```

```
## # A tibble: 5 x 2
##   continent EV_prom
##   <fct>       <dbl>
## 1 Africa       54.8
## 2 Americas     73.6
## 3 Asia         70.7
## 4 Europe       77.6
## 5 Oceania      80.7
```

El operador %>% pasa el resultado del último cálculo al primer argumento de la siguiente función, para que no sea necesario reescribirlo o guardarlo. Con este operador, los datos entran en una tubería (`pipe`), donde en cada paso se realiza un cálculo y al final solo se recibe el resultado. Recuerda, es necesario cargar el paquete *dplyr* o cualquiera del conjunto que hace parte de *tidyverse* (diferente a *ggplot2*) para usar el operador %>%.

1.3 Comentarios finales

Por su gran cantidad de usuarios, hay muchos ejemplos de visualizaciones creadas con *ggplot2* disponibles en internet. Un buen ejemplo es

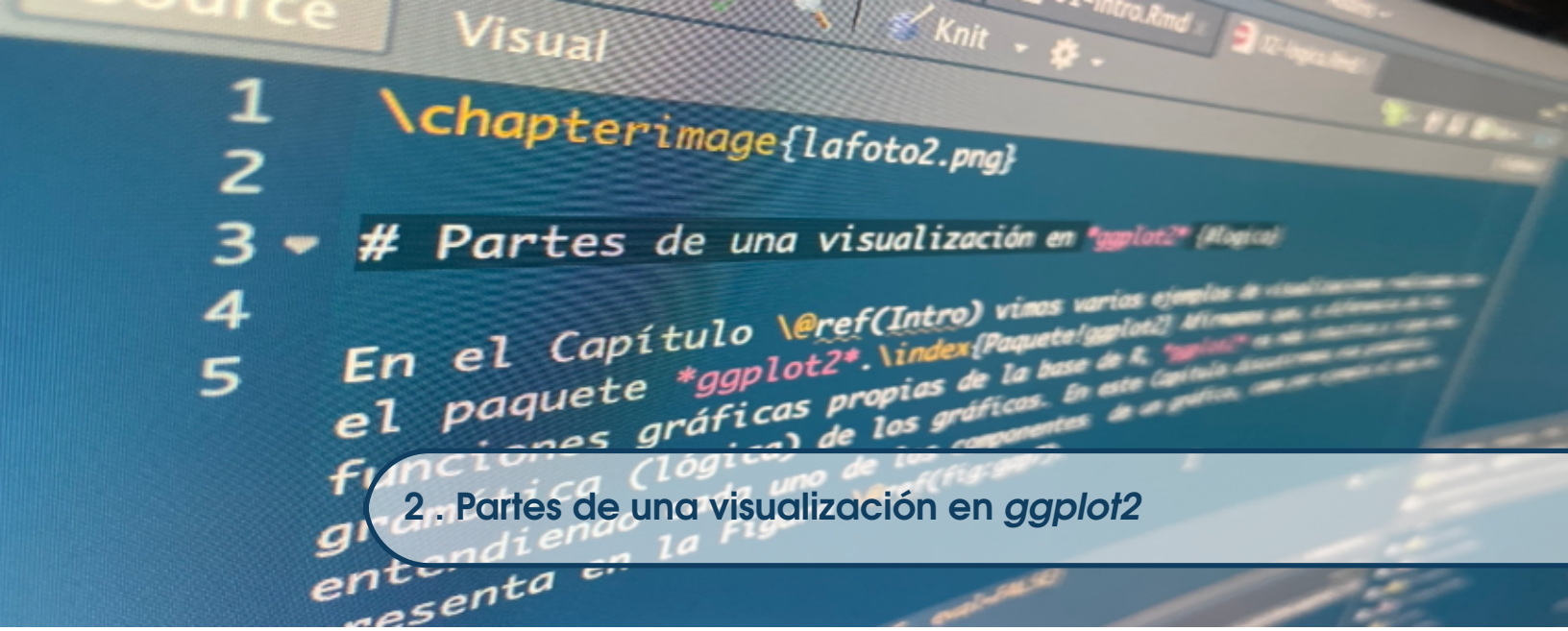
la galería que se presenta en la *The R Graph Gallery*. Estas comunidades no solo presentan los ejemplos de las visualizaciones que podemos crear con este paquete, sino que es común que se provea el código que generó las visualizaciones. Después de finalizar la lectura de este libro, podrás entender esos códigos y adaptarlos para tu caso particular. También encontrarás en línea numerosos *blogs*[8] y foros[9] dedicados a responder dudas sobre errores en este paquete.

No podemos terminar este primer capítulo sin recordar que *ggplot2* no es el único paquete para hacer gráficos. Dentro de la base (o *core*) de R hay otras opciones para "mapear" datos a una visualización, sin tener necesidad de recurrir a instalaciones externas. Lo que hace a este paquete tan especial es que obedece una gramática preestablecida en la estadística y lo traduce a programación de R. Esto hace que su implementación no sea tan compleja como puede llegar a ser la construcción de una visualización en la base de R. El siguiente capítulo te enseñará los elementos esenciales de la gramática de las visualizaciones.

Finalmente, al tener muchos usuarios de uso, existen numerosas comunidades de usuarios que se reflejan en diversas discusiones sobre la realización de gráficos de manera eficiente en los diferentes *blogs* de R.

[8] Por ejemplo, *R-Blogger* es un sitió que se dedica a recoger entradas de diferentes *blogs* de la comunidad de R (https://www.r-bloggers.com).

[9] Por ejemplo, *Stackoverflow* es un foro muy común para usuarios en R que tiene una sección especial para ggplot (https://stackoverflow.com/questions/tagged/ggplot2).



2 . Partes de una visualización en *ggplot2*

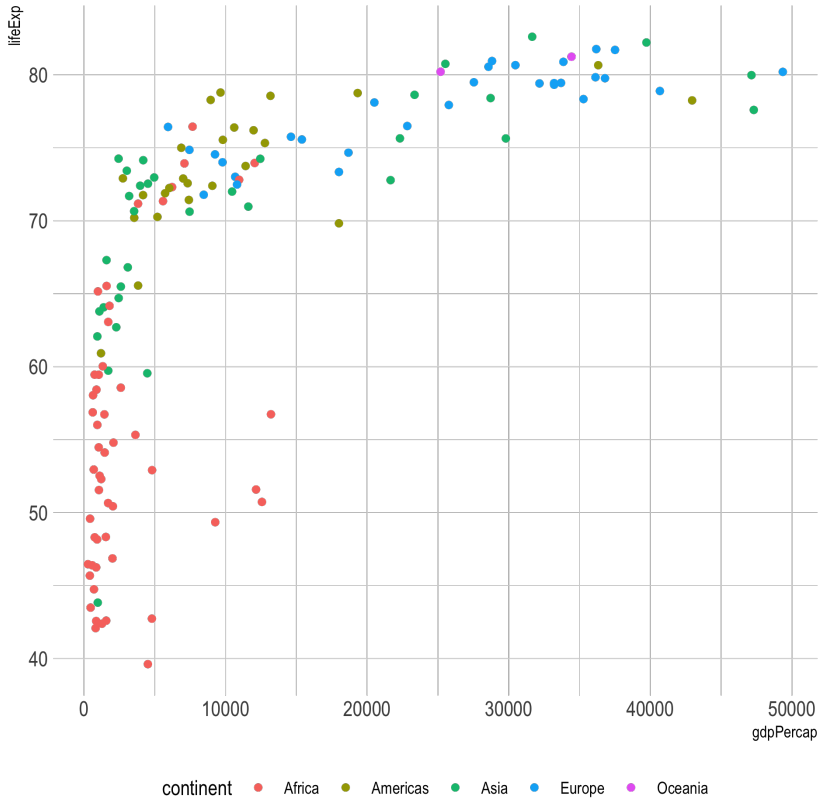
En el Capítulo 1 vimos varios ejemplos de visualizaciones realizadas con el paquete *ggplot2*. Afirmamos que, a diferencia de las funciones gráficas propias de la base de R, *ggplot2* es más intuitivo y sigue una gramática (lógica) de los gráficos. En este Capítulo discutiremos esa gramática entendiendo cada uno de los componentes de un gráfico, como por ejemplo el que se presenta en la Figura 2.1.

La Figura 2.1 parte de unos datos (que salieron del paquete *gapminder* (Bryan, 2017)) y unas decisiones de cómo visualizar esos datos. Reflexionemos un poco sobre todas las decisiones que se tomaron antes de construir el gráfico. Después de tener los datos, se parte de una idea de qué variables "mapear" (graficar) en cada uno de los ejes. En este caso, el PIB percápita (`gdpPercap`) y la esperanza de vida al nacer (`lifeExp`). Se eligió cómo representar cada observación (en este caso, con puntos) y cómo el color de estos puntos tendría una variable "mapeada". Se asignó el continente (`continent`) al color. También, se decidió de manera implícita que el tamaño del punto no tendría ningún significado. Adicionalmente, se seleccionó en qué unidades medir los ejes (las coordenadas). Finalmente, se decidió cómo sería el diseño del gráfico, al escoger la posición de los nombres de las variables en los ejes y si el cuerpo del gráfico tendría grilla[1] y el color del fondo.

En este capítulo discutiremos cada uno de esos componentes y veremos la *jerga* que se emplea en el paquete *ggplot2* para referirse a ellos.

[1] En otras palabras, las líneas horizontales y verticales al interior de la visualización. A estas líneas también se les conoce como retícula (en inglés *grid lines*).

Figura 2.1. Relación entre el PIB percápita y la experiencia de vida al nacer por país (2007)



Fuente: Datos del paquete gapminder.

2.1 Componentes

Veamos formalmente cómo *ggplot2* emplea la gramática de los gráficos de Wilkinson (2012). Es decir, identifiquemos cuáles son los componentes principales y cómo estos se sobreponen para obtener un gráfico. En este contexto, un gráfico está compuesto por diferentes capas que se sobreponen de tal manera que el resultado final es una visualización. En la Figura 2.2 se presentan las diferentes capas que componen un objeto de clase **ggplot**.

Figura 2.2. Capas de una visualización construida con ggplot2

Fuente: Elaboración propia a partir de Wickham (2016) y Eilkinson (2012).

No todas las capas son necesarias para construir una visualización en *ggplot2*, pero cada una aportará al producto final un elemento diferente. Veamos cada uno de estas capas (*layers* en inglés):

- ***Datos*** o data: es la primera capa necesaria para realizar visualizaciones (ver Figura 2.2). Es la base fundamental para construir la

visualización y por tanto es una parte obligatoria. Esta capa llama a los datos, que deben estar en un objeto de clase **data.frame**[2] o **tibble**[3]. En la Figura 2.1 esta capa corresponde al objeto `gapminder` (**data** = gapminder).

- ***Aesthetics***: en esta capa se indican las variables que se van a "mapear" (graficar) en los diferentes elementos del gráfico, como por ejemplo los ejes, el color y el tamaño de los puntos. Una traducción literal de esta capa sería "estética", pero de pronto no es una buena traducción. Esta capa indica qué variables del *data.frame* se mapean en el eje horizontal (denotado por **x**) y cuál al eje vertical (**y**). También podemos mapear en esta capa el color (**col**), como en la Figura 2.1, donde el continente (que es una variable cualitativa) se mapeó al color de cada punto (**col = continent**). En dicha figura, la variable cuantitativa PIB percápita se representa en el eje horizontal (**x = gdpPercap**) y la esperanza de vida al nacer (variable cuantitativa) en el eje vertical (**y = lifeExp**). Adicionalmente, podemos mapear variables (si tiene sentido) al tamaño (**size**) y otros elementos como la forma del punto. Al igual que la primera, esta capa es obligatoria; constituyéndose, con los datos, en uno de los tres pilares de cualquier visualización que armemos con *ggplot2*. Esta capa es conocida en el paquete como **aes**.
- ***Geometría***: esta capa le indica a R qué tipo de visualización se desea obtener. Es decir, cuál es la geometría que se desea adoptar para mostrar los datos. Por ejemplo, la geometría podría ser emplear barras, líneas, puntos, histogramas, entre otros. En el caso particular de la Figura 2.1, la geometría seleccionada fue los puntos (**geom_point()**). Hay distintas opciones que se explorarán más adelante en los Capítulos 3, 4 y 5[4]. Esta capa es obligatoria y constituye el tercer pilar indispensable en toda visualización de *ggplot2*.

- ***Facets***: esta capa permite descomponer un gráfico en sub-gráficos (cuadrículas o facetas), según una variable cualitativa. Por ejemplo, en vez de usar colores para distinguir los continentes, como se hace en la Figura 2.1, podríamos construir 5 sub-gráficos, cada uno para un continente como lo muestra la Figura 2.6. Las facetas sirven para comparar grupos, separándolos y así facilitando la identificación de cada uno. No es una capa obligatoria.

- ***Estadísticas***: esta capa permite adicionar información estadística que permita resumir los datos. Por ejemplo, se puede adicionar una línea de tendencia para los datos. No es obligatoria.

[2] En el Capítulo 5 de Alonso y Ocampo (2022) puedes encontrar una introducción a esta clase de objetos.

[3] En el Capítulo 1 de Alonso (2022) se presenta una breve introducción a esta clase de objetos.

[4] En el Capítulo 9 se discutirán geometrías avanzadas.

- ***Escalas***: esta capa permite indicar en qué escala se presentan los datos en cada uno de los ejes. Por ejemplo, en algunas ocasiones se podría desear presentar uno de los dos ejes en escala logarítmica. No es obligatoria.
- ***Coordenadas***: esta capa determina cómo se combinan las variables seleccionas para los ejes `x` (horizontal) y `y` (vertical) en la capa de ***Aesthetics***. En últimas, esta capa define el espacio en el que se mapearán los datos. Por ejemplo, en algunas ocasiones será útil poner límites a los ejes, intercambiar lo que se presenta en un eje con el otro, o asegurarnos que los dos ejes tienen la misma magnitud. No es una capa obligatoria.
- ***Tema***: es la capa destinada a la apariencia final de la gráfica. Podemos encontrar unos temas predeterminados cargados con el paquete *ggplot2* y también se puede crear un tema para que se adapte a la imagen institucional o al tipo de diseño de todo el documento. También existen algunos paquetes que incluyen temas adicionales. Aquí se modifica el color del fondo, ejes, tamaños, grilla, posición de los nombres de los ejes, entre otros. En el 8 podrás encontrar una extensa discusión sobre cómo modificar esta capa. No es obligatoria.

Veamos un ejemplo para entender mejor cada capa y sus argumentos.

2.2 Primeros pasos con *ggplot2*

Para empezar a trabajar con visualizaciones en R debemos instalar el paquete *ggplot2* (Wickham, 2016), en caso de que no esté instalado en el equipo. Los **paquetes** incluyen diversas funciones, para tareas generales o específicas[5]. En especial, este paquete contiene herramientas de visualización. Para proceder a la instalación, puedes emplear la función **install.packages()**.

```
install.packages("ggplot2")
```

La instalación de un paquete solo es necesaria una vez. Recuerda que tenemos que pedirle a R que cargue el paquete cada vez que iniciemos una nueva sesión. Para esto, emplea la función **library()** . Carguemos el paquete *ggplot2*.

[5]El Capítulo 7 de Alonso y Ocampo (2022) presenta una breve introducción a los paquetes en R.

```
library(ggplot2)
```

Para nuestro ejemplo usaremos el objeto de datos `gapminder` del paquete **gapminder** (Bryan, 2017) que ya empleamos en la Sección 1.2. Carga el objeto `gapminder`.

```
# cargar paquete
# install.packages("gapminder")
library(gapminder)
# cargar datos
data("gapminder")
```

Veamos cómo materializar lo descrito en la sección anterior (Sección 2.1). Veíamos que, de las ocho capas descritas, las capas de ***Datos***, ***Aesthetics*** y ***Geometría*** son esenciales. Las dos primeras capas se especifican empleando la función **ggplot()** . El primer argumento de esta función corresponde a la primera capa: los datos[6]. El argumento **data** puede ser un objeto de clase *data.frame* o `tibble`.

```
ggplot(data = gapminder)
```

Correr solo esta parte de la función tendrá como resultado un gráfico en blanco (ver Figura 2.3), porque aún nos faltan los otros dos pilares necesarios para una visualización: Las capas ***Aesthetics*** y ***Geometría***.

La segunda capa, ***Aesthetics***, se especifica como otro argumento de la función **ggplot()**. La función **aes()** permite especificar cómo se mapearán las variables a los diferentes elementos de la visualización. Para este primer ejemplo, se quiere mostrar la relación entre dos variables numéricas (o cuantitativas): la expectativa de vida al nacer (`lifeExp`) y el PIB percápita (`gdpPercap`). Entonces, se representará en el eje horizontal (**x**) la variable `lifeExp` y al eje vertical (*y*) la variable `gdpPerCap`, dentro del argumento **aes()**.

Es decir, la primera y segunda capa de la figura corresponderá a la siguiente línea de código:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))
```

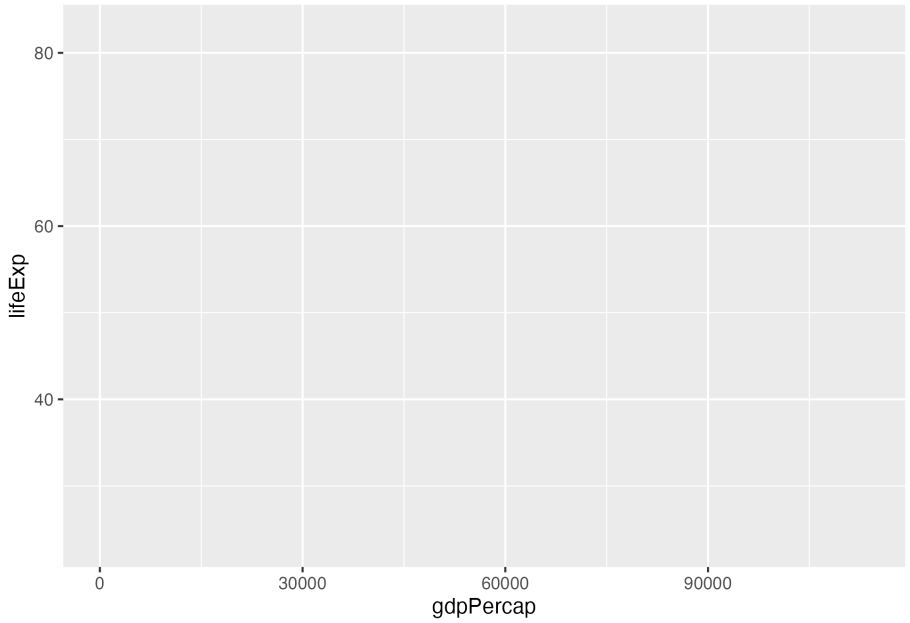
Nota que al correr esta línea, aparece el plano cartesiano en el que se ubicarán las variables, pero no los datos en sí. Esto sucede porque aún nos falta el último pilar: la ***Geometría*** (ver Figura 2.4). Es decir, aún

[6] Si se emplea el operador %>% este argumento se puede pasar desde la línea anterior.

Figura 2.3. Primera capa (datos) de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007)

Fuente: Datos del paquete gapminder.

Figura 2.4. Primera y segunda capa de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007)



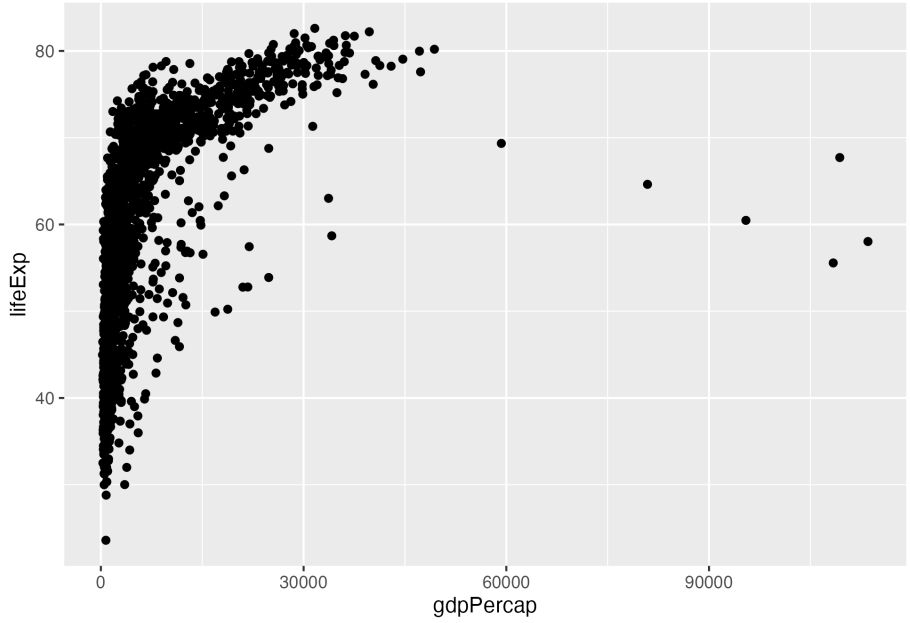
Fuente: Datos del paquete gapminder.

no hemos especificado cómo se mapearán los datos. En otras palabras, no se ha especificado el tipo de visualización que vamos a emplear.

La capa de ***Geometría*** (**geom** en la jerga de *ggplot2*) se especifica mediante una segunda función. Existen diferentes tipos de gráficas, como por ejemplo: histogramas, líneas, barras y puntos. Dado que tenemos diferentes posibilidades de geometría, tendremos diferentes funciones para cada una de ellas (en los próximos capítulos discutiremos sobre estos temas). Estas funciones se caracterizan por empezar con el prefijo **geom_**. Por ejemplo, para la visualización que estamos construyendo se desea un diagrama de dispersión en el que cada dato se representa por puntos. Así, la función correspondiente será: **geom_point()**. Esta capa se suma a las dos capas anteriores de la siguiente manera:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()
```

Figura 2.5. Primeras tres capas de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007).



Fuente: Datos del paquete gapminder.

Observa que esta capa se agrega a las dos anteriores con el signo

de suma (+). En la Figura 2.5 podemos observar las tres primeras capas esenciales: ***Datos***, ***Aesthetics*** y ***Geometría***.

Hasta aquí tenemos las capas necesarias para crear una visualización; pero podemos adicionar otras para mejorarla. Las siguientes capas se pueden agregar, si se desea, en cualquier orden. Por ejemplo, en este caso especial de esta visualización, tendría sentido incluir la capa de ***Facets***.

El objeto `gapminder` también contiene la variable continente (`continent`). Si queremos entender la distribución de las dos variables bajo análisis en los diferentes continentes, podemos emplear la capa de ***Facets*** para dividir los datos de tal manera que grafiquemos para cada continente la relación entre ellas (ver Figura 2.6).

Esta capa se puede adicionar con la función **facet_grid()**. Esta función crea una grilla (*grid* en inglés) o parrilla de gráficos, separando los datos de acuerdo a la variable que se emplee como argumento. Siguiendo la tradición del lenguaje de R, la variable por medio de la cuál se quieran dividir los gráficos debe estar precedida del operador virgulilla[7] (~) . Regresando a la figura que estábamos construyendo, la capa de ***Facets*** se puede agregar para que los datos sean visualizados por continente, como se muestra en el siguiente código:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point() +
  facet_grid(~ continent)
```

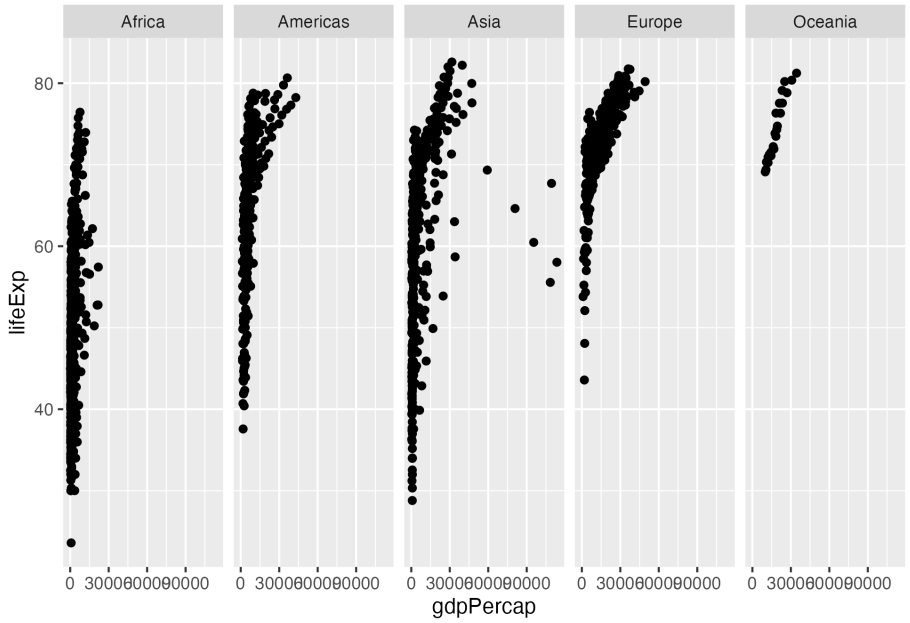
Nota que otra forma de visualizar las diferencias entre continentes podría ser mapear al color la variable continente y no usar la capa de ***Facets***. ¡Inténtalo! El resultado sería algo muy similar a la Figura 2.1. La diferencia es que en la Figura 2.1 se presentan solo datos para el año 2007 y tu gráfica debe tener datos para toda la muestra. En ese caso parece mejor transmitir la idea de las diferencias entre continentes la capa de ***Facets*** que mapear los continentes al color del punto. Las decisiones sobre qué comunica mejor tu idea siempre estarán presentes al momento de hacer visualizaciones. Lo mejor será **siempre ensayar muchas opciones antes de decidir cuál visualización será la mejor para comunicar tus ideas**.

Ahora podemos jugar un poco con la capa de ***Escalas***[8] para ayu-

[7] Virgulilla es el "palito curvo" que se emplea en el español encima de la letra ene (n) para convertirla en eñe (ñ). En inglés este operador (~) se conoce como *tilde*. Este operador se emplea en la base de R para definir la relación entre la variable dependiente y las variables independientes en la fórmula de un modelo estadístico. La variable a la izquierda del operador ~ es la variable dependiente y la o las variables a la derecha de éste son las variables independientes. Siguiendo esa tradición, en *ggplot2* se emplea ~ para determinar que la capa de ***Facets*** dependerá de una variable **x**. Es decir, **~ x**.

[8] Estas capas que no son pilares de la visualización se pueden adicionar en cualquier orden, pero en algunas ocasiones el resultado puede cambiar. En un momento te darás cuenta que por razones pedagógicas cambiamos el orden que se presentó inicialmente en el la Figura 2.2.

Figura 2.6. Primeras tres capas y capa Facets de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007)



Fuente: Datos del paquete gapminder.

darnos a comunicar un mensaje. La capa de **Escalas** permite cambiar la escala en la que se mide un eje y algunos detalles de cada uno de los ejes como los límites o las etiquetas de cada eje. En otras palabras, esta capa controla los detalles de cómo se traducen los valores de los datos a las propiedades visuales.

Por ejemplo, podríamos cambiar la escala en la que se presentan los datos del PIB percápita (`gdpPercap`) de dólares por persona a una escala logarítmica (logaritmo base 10), empleando la función **scale_x_log10()**. Regresando a nuestro ejemplo, cambiemos la escala del eje horizontal[9] para obtener la Figura 2.7 empleando el siguiente código:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10()
```

El paquete *ggplot2* tiene a disposición diferentes transformaciones y operaciones que podemos hacer con las escalas de los ejes, donde todas esas funciones empiezan por el prefijo **scale_**[10].

La capa **Escalas** también puede ser modificada con funciones que no empiezan con el prefijo **scale_**. Por ejemplo, la función **labs()** permite agregar un título (argumento **title**), subtítulo (argumento **subtitle**) y la fuente (argumento **caption**). También tenemos las funciones **xlab()** y **ylab()** que permiten cambiar las leyendas de los ejes que por defecto es el nombre de la variable mapeada. Esta capa también tiene funciones que permiten cambiar los limites de las escalas que se presentan en cada eje (**xlim()** y **ylim()**). Tienes que tener cuidado con estas dos últimas funciones, pues cualquier dato fuera de los límites se desechará.

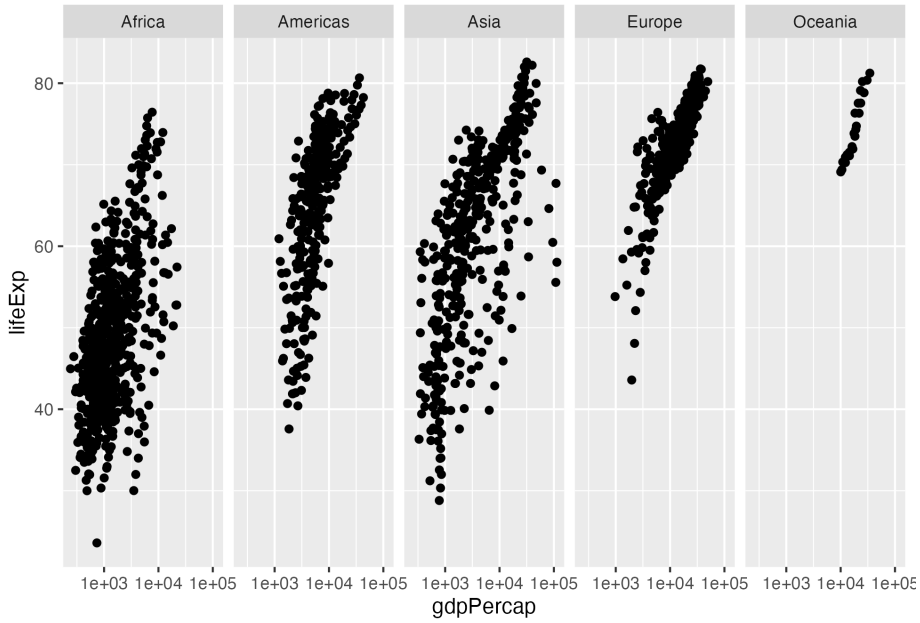
Juega un poco con estas funciones para entender cómo funcionan. Intenta correr el siguiente código y ver el efecto que tiene sobre tu visualización:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10() +
  labs(
  title = "Relación entre  PIB percápita y
```

[9]Nota que este cambio mejora la estética del eje **x** de la visualización, ya que en la Figura 2.6 los valores aparecen sobrepuestos.

[10]Por ejemplo, están las funciones **scale_y_continuous()** y , **scale_y_reverse()**, que permiten convertir la escala del eje vertical (`y`) a una variable continua y cambiar a un orden descendente los valores del eje, respectivamente. ¡Intenta jugar con todas estas funciones!

Figura 2.7. Capa Facets y Escalas de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007)



Fuente: Datos del paquete gapminder.

```
          Expectativa de vida al nacer por país",
  subtitle = "(1952 - 2007)",
  caption = "Fuente: Datos de gapminder") +
  xlab("PIB percápita en logaritmos") +
  ylab("Esperanza de vida al nacer en años")
```

La capa de **Coordenadas** es la que le dice a *ggplot2* cómo combinar la variable en la capa de ***Aesthetics*** mapeada en `x` y `y` para construir la visualización en dos dimensiones. Comúnmente, estamos acostumbrados a emplear coordenadas cartesianas (el plano cartesiano) para mapear los datos. Por defecto *ggplot2* emplea coordenadas cartesianas (función **coord_cartesian()**[11]). Una función de esta capa que puede ser útil es **coord_flip()**, que invierte los ejes. Para un ejemplo de la utilidad de esta función puedes ver las Figuras 6.1 y 6.2 del Capítulo 6. Por ahora, intenta el siguiente código:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10() +
  coord_flip()
```

Otra función útil de esta capa es **coord_fixed()**, la cual fija la relación entre el espacio que ocupa el eje x y el eje y. Con el argumento **ratio** podemos establecer el número de unidades en el eje vertical que equivalen a una unidad en el eje horizontal. El valor por defecto es uno (`ratio = 1`). Una razón mayor a uno, implica que las unidades del eje horizontal serán más largas y viceversa. Por ejemplo, intenta el siguiente código y juega con el argumento **ratio**:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10() +
  coord_fixed(ratio = 0.05)
```

[11]Esta función permite asignar límites en los ejes para los que se presentará los datos. Esto genera un *zoom* en una parte de los datos. Intenta adicionar al código que generó la Figura 2.7 la siguiente línea de código `coord\_cartesian(ylim = c(40,60))`. Recuerda añadir el signo + antes de incluir esta nueva capa. Esto genera una visualización con solo los datos que están entre los límites. La diferencia entre expresar los límites de los ejes en la capa de **Coordenadas** y de **Escalas**, es que en el segundo caso, los datos que estén por fuera se omiten y no se tienen en cuenta para ningún cálculo, mientras que en el segundo caso solo no se visualizan. Esto tiene un impacto sobre la gráfica, en especial cuando se adicione la capa de **Estadística** como se mostrará en el siguiente pie de página.

La capa de **Coordenadas** tiene más funciones para hacer mapas (por ejemplo, **coord_map()**, **coord_quickmap()** y **coord_sf()**) y coordenadas polares (**coord_polar()**). Como con cualquier otra función de este paquete, si necesitas alguna de ellas podrás encontrar una documentación muy completa en línea.

La capa de ***Estadísticas*** nos puede servir para reforzar una idea que queremos transmitir con una visualización. Por ejemplo, parece evidente que existe una relación positiva y lineal entre la expectativa de vida al nacer (`lifeExp`) y el PIB percápita (`gdpPercap`) (en logaritmos) (ver Figura 2.7). Para mostrar esto visualmente, podríamos incluir una línea de regresión que muestre la relación lineal entre las dos variables. Existen diferentes tipos de estadísticas que nos podrían ayudar a construir nuestra visualización. Estas funciones se caracterizan por empezar con el prefijo **stat_**. Por ejemplo, la función **stat_smooth()** ayuda a trazar una línea "suavizada" que se ajuste a los datos. Esta tiene como argumento el método que queremos emplear para trazar la linea de suavización. Por ejemplo si fijamos **method = "lm"** como método, se está indicando que busque una relación lineal (*Lineal Model*, por sus siglas en inglés) (ver Figura 2.8).

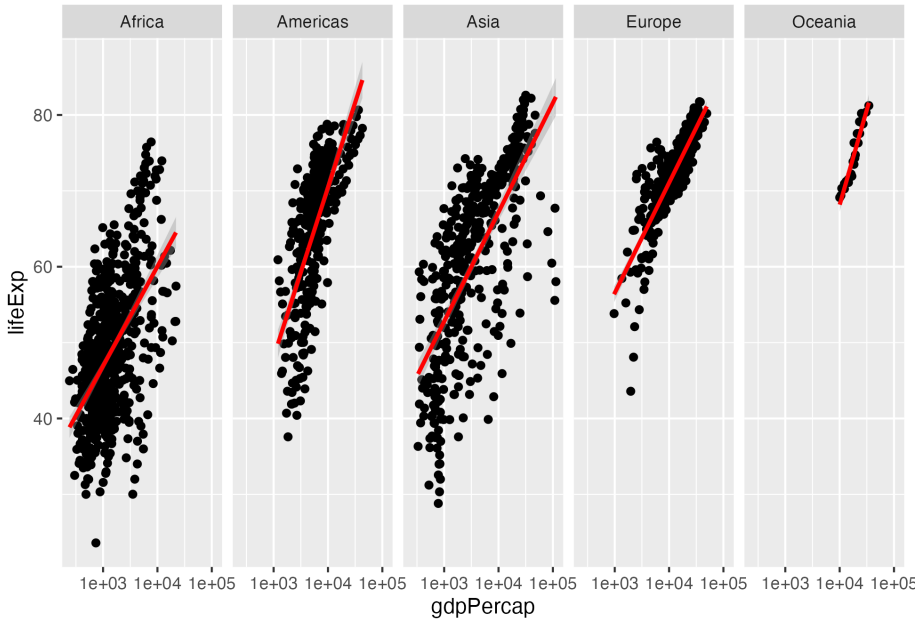
También podemos modificar el color de la línea de suavización (en este caso la recta de regresión) con el argumento **col**. Si no se especifica un color, el color será azul por defecto. En nuestro ejemplo, tendremos lo siguiente:

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10() +
  stat_smooth(method="lm", col="red")
```

Observa que en la Figura 2.8 se ha trazado una línea de regresión para cada continente empleando todos los datos del respectivo continente[12]. Esto transmite mejor la idea de que existe una relación líneal positiva entre el logaritmo del PIB percápita y la esperanza de vida al

[12]Es interesante ver el efecto que tiene cambiar los límites de los ejes en la capa de **Escalas** o en la de **Coordenadas**. Adiciona primero al código que generó la Figura 2.8 una nueva línea modificando la capa de **Coordenadas** (recuerda adicionar el signo + al final de la línea anterior): `coord_cartesian(ylim = c(40,60))`. Esto muestra exactamente las mismas líneas de regresión de la Figura 2.8, pero haciendo un acercamiento a los datos que presentan una expectativa de vida al nacer entre 40 y 60 años. Ahora, en vez de modificar la capa de **Coordenadas**, modifiquemos la de **Escalas** incluyendo el siguiente código: `ylim(40,60)`. En esta oportunidad, la línea de regresión para cada continente se calculó incluyendo solo los datos con una esperanza de vida al nacer entre 40 y 60. Ahí está la diferencia de emplear una de las dos capas para definir el límite de los datos. Todo dependerá de lo que quieras comunicar con tu visualización.

Figura 2.8. Capa Facets, Escalas y Estadísticas de la Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007)



Fuente: Datos del paquete gapminder.

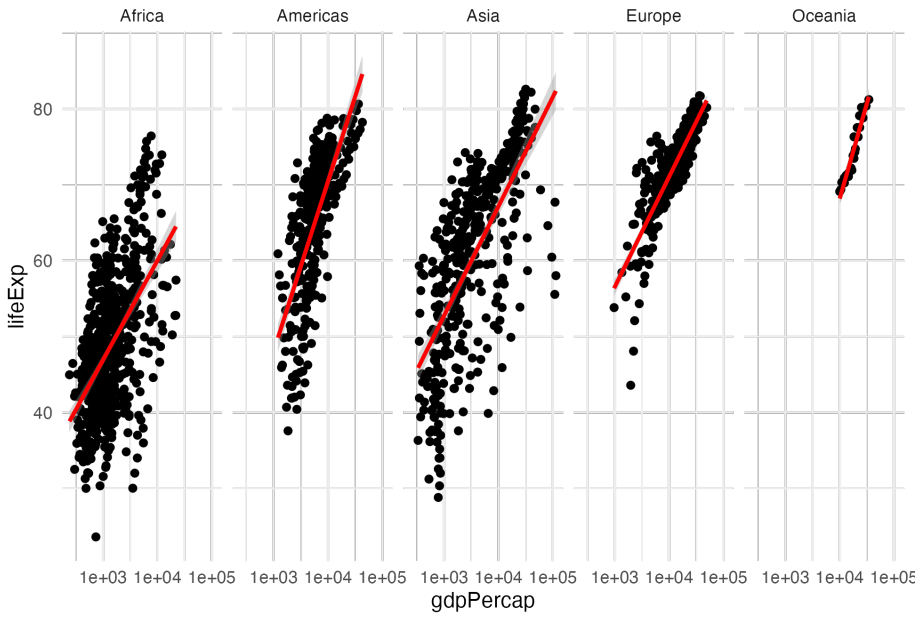
nacer. Típicamente, esta capa se emplea para transmitir algún hallazgo en la exploración o el modelado de los datos.

Nuestra visualización ha cambiado sustancialmente desde la que teníamos en la Figura 2.5. Y podemos continuar modificando nuestra visualización con la capa de ***Tema***. Como se mencionó en la sección 2.1, el paquete *ggplot2* viene con temas predeterminados que determinan cómo es el fondo de la figura, la posición y el tamaño de los nombres de los ejes. Pero este paquete también brinda la opción de crear nuevos temas, para poder agregar logos de empresas, tamaños específicos de los elementos, entre otras opciones. En general, el tema ayuda a que la visualización sea estéticamente más acorde al documento donde se va a mostrar (ver Figuras del Capítulo 1).

Dentro de los temas disponibles directamente en este paquete están **theme_minimal()**, **theme_classic()**, **theme_bw()**, entre otros. Siguiendo la misma gramática de las anteriores capas, esta se agrega sumándola a las anteriores. En este caso las funciones de tema (todas inician con el prefijo **theme_**) no necesitan argumentos para funcionar. Por ejemplo, el siguiente código incorpora el tema minimalista (**theme_minimal()**) (ver Figura 2.9). En el Capítulo 8 podrás encontrar una discusión en detalle de esta capa.

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp))+
  geom_point()+
  facet_grid(~continent)+
  scale_x_log10() +
  stat_smooth(method="lm", col="red") +
    theme_minimal()
```

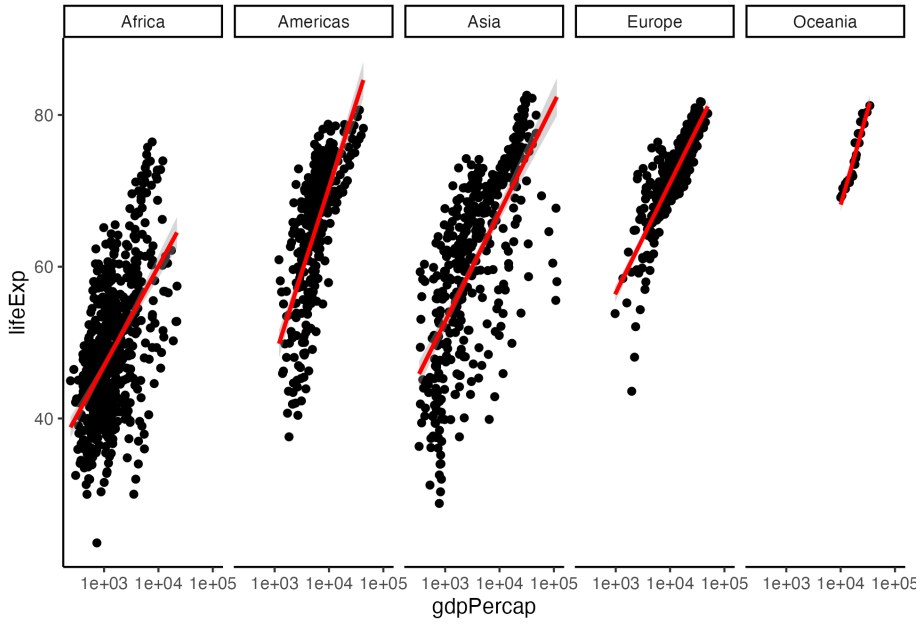
Figura 2.9. Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007) con tema minimalista



Fuente: Datos del paquete gapminder.

Ahora puedes intentar jugar con las diferentes opciones de temas. Por ejemplo, la Figura 2.10 emplea la función **theme_classic()**. Con estas ocho capas, el paquete *ggplot2* nos brinda una gran flexibilidad para hacer visualizaciones rápidamente.

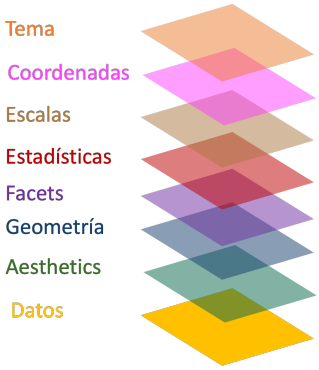
Figura 2.10. Figura de relación entre PIB percápita y Expectativa de vida al nacer por país (1952 - 2007) Con tema clásico



Fuente: Datos del paquete gapminder.

Para resumir, en la Figura 2.11 se muestra de manera esquemática cada una de las capas y el correspondiente código empleado para generar la Figura 2.9.

Figura 2.11. Capas de la Figura 2.9. y su respectivo código




```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
         geom_point() +
        facet_grid (.~continent) +
        stat_smooth(method ="lm", col="red") +
        scale_x_log10() +
        theme_minimal()
```

Fuente: Elaboración propia a partir de Wickham (2016) y Eilkinson (2012).

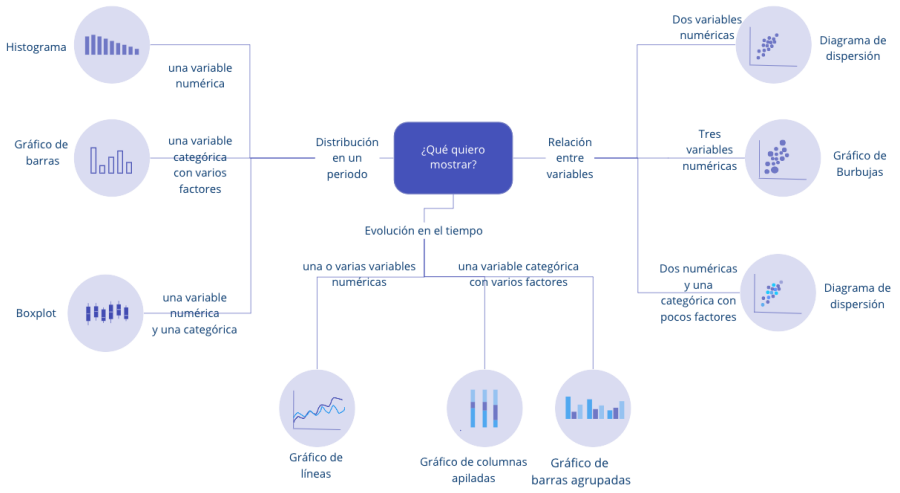
2.3 Comentarios finales

Antes de continuar, es importante mencionar que este paquete y otros adicionales, permiten incluir más capas y funciones de las que vimos en este capítulo. Por ejemplo, se pueden usar funciones de otros paquetes para agregar paletas de colores específicas, títulos, entre otras características, que ayudarán a mejorar la visualización.

Son tantas las opciones que con seguridad te podrás divertir experimentando poco a poco con las diferentes alternativas disponibles. Hacer buenas visualizaciones no es tarea fácil y siempre implicará probar muchas opciones de capas, incluyendo la geometría.

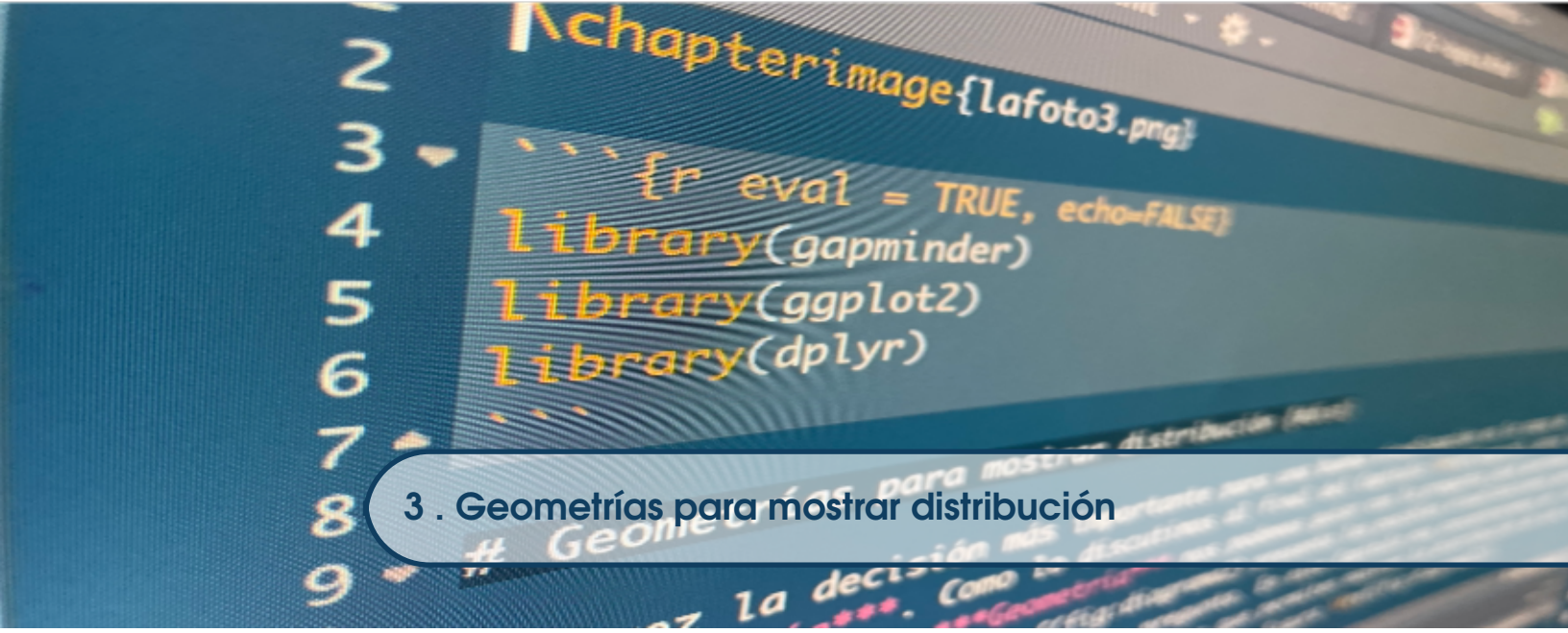
Como se definió en la Sección 2.1, la capa de ***Geometría*** indica el tipo de visualización que se construirá. La selección de la geometría adecuada es tal vez la decisión más importante para obtener una visualización poderosa. Para escoger la capa de ***Geometría*** debemos preguntarnos qué queremos mostrar con ella (ver por ejemplo Abela (2008) o Alonso y González (2012)). En la Figura 2.12 se presenta un diagrama que te puede ayudar para decidir que tipo de gráfico (capa de ***Geometría***) emplear según la intención y la clase de variable que se tenga.

Figura 2.12. Tipos de gráficos más comunes según lo que se desea comunicar y la clase de variable



Fuente: Adaptación de Alonso y González (2012).

En el Capítulo 3 discutiremos los gráficos más empleados para mostrar la distribución de los datos (ver Figura 2.12). En el Capítulo 4, las visualizaciones más empleadas para mostrar la evolución de una variable (ver Figura 2.12) y en el Capítulo 5 aquellas para mostrar relaciones entre variables.



3 . Geometrías para mostrar distribución

Tal vez la decisión más importante para una buena visualización es la capa de ***Geometría***. Como lo discutimos al final del Capítulo 2, para la selección de la ***Geometría*** nos podemos guiar con la pregunta: ¿qué queremos mostrar? La Figura 3.1 presenta las posibles visualizaciones para las posibles respuestas a esta pregunta. En este Capítulo nos concentraremos en cómo implementar los gráficos más comunes que permiten mostrar la distribución de una o varias variables (ver área sombreada de la Figura 3.1).

Los gráficos más comunes para mostrar la distribución de una o varias variables son[1]:

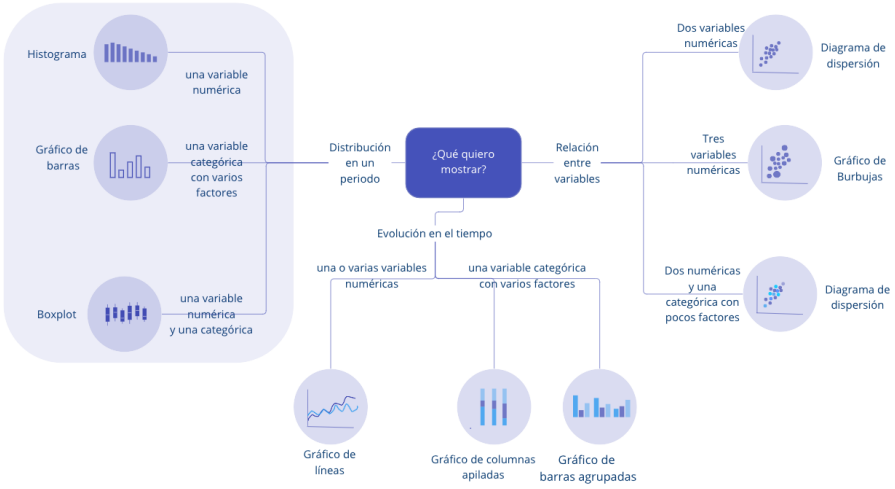
- **Histograma**
- **Gráfico de barras**
- **Boxplot** o Diagrama de cajas[2].

Estos gráficos nos permiten representar todas las observaciones de la variable, cuáles son más frecuentes, cuál es la tendencia central de los datos y su dispersión. En otras palabras, estas visualizaciones están diseñadas para mostrar rápidamente cómo se comportan todos los datos. En general estas geometrías hacen más sencillo mostrar grandes cantidades de datos.

[1]Las gráficas de tortas no son recomendables. Como se discutirá con más detalle en el Capítulo 6, el gráfico de torta **no es una buena elección** debido a que el ojo humano no es bueno calculando los ángulos. ¡No se recomienda emplear esta visualización! Existen mejores alternativas.

[2]También conocido como diagrama de cajas y bigotes.

Figura 3.1. Tipos de gráficos más comunes según lo que se desea comunicar y la clase de variable.



Fuente: Adaptación de Alonso y González (2012).

3.1 Histograma

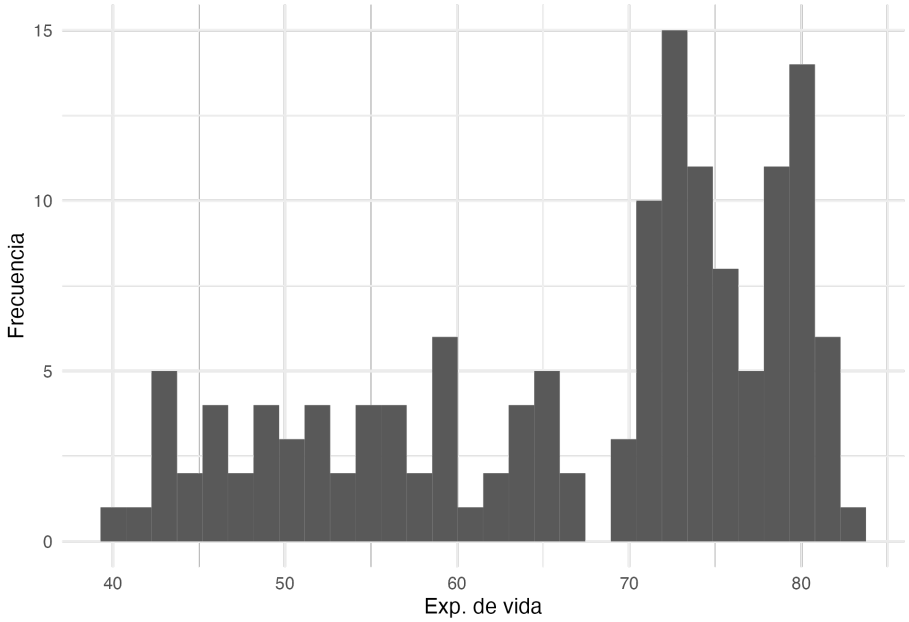
Los histogramas están diseñados para mostrar la distribución de una variable de clase **numeric** (variables cuantitativas contínuas) o de clase **integer** (variables cuantitativas discretas). Un histograma agrupa los datos en intervalos pequeños que se miden típicamente en el eje horizontal, y en el eje vertical se representa la frecuencia de aparición de las observaciones dentro de los límites del intervalo. La función para este tipo de gráficos es **geom_histogram()**.

Veamos un ejemplo con los datos empleados en el Capítulo 2, del objeto `gapminder` del paquete del mismo nombre. Representemos la distribución de la expectativa de vida al nacer de todos lo países para el año 2007. Carguemos los paquetes y filtremos los datos para el año 2007. Esta será la primera capa. Ahora, empleemos el paquete *ggplot2* para construir el histograma. Necesitamos enviar a la capa de datos el objeto `gapminder`, en la capa de ***Aesthetics*** especificar la variable que mapearemos en el eje **x**. Después, la capa de ***Geometría*** corresponderá a **geom_histogram()**. El siguiente código genera la Figura 3.2.

```
# cargar los paquetes
library(ggplot2)
library(dplyr)
```

```
library(gapminder)

# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x=lifeExp)) +
    geom_histogram() +
    labs( y="Frecuencia",
          x="Exp. de vida") +
    theme_minimal()
```

Figura 3.2. Histograma de la esperanza de vida al nacer de todos los países del mundo (2007)



Fuente: Datos del paquete gapminder.

Nota que en el código incluimos en la capa de **Escalas** la función **labs()**, que corresponde a las etiquetas en inglés. Esta función nos permite ponerle nombres a los ejes, título (argumento **title**) y subtítulo (argumento **subtitle**), entre otras etiquetas.

La Figura 3.2 muestra un histograma en color gris; color que es el valor por defecto de esta función. Si quisieras cambiar el color del histograma[3], lo podemos hacer con el argumento **fill**. Por ejemplo, corre estas líneas de código y obtendrás un histograma de color azul claro:

```
# Histograma de colo azul claro
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = lifeExp)) +
    geom_histogram(fill = "lightblue") +
    labs( y="Frecuencia",
          x="Exp. de vida") +
    theme_minimal()
```

Si quieres ajustar otros aspectos del histograma, como el número de clases (grupos), puedes emplear los diferentes argumentos de esta función. Recuerda que buscar ayuda sobre una función es muy fácil.

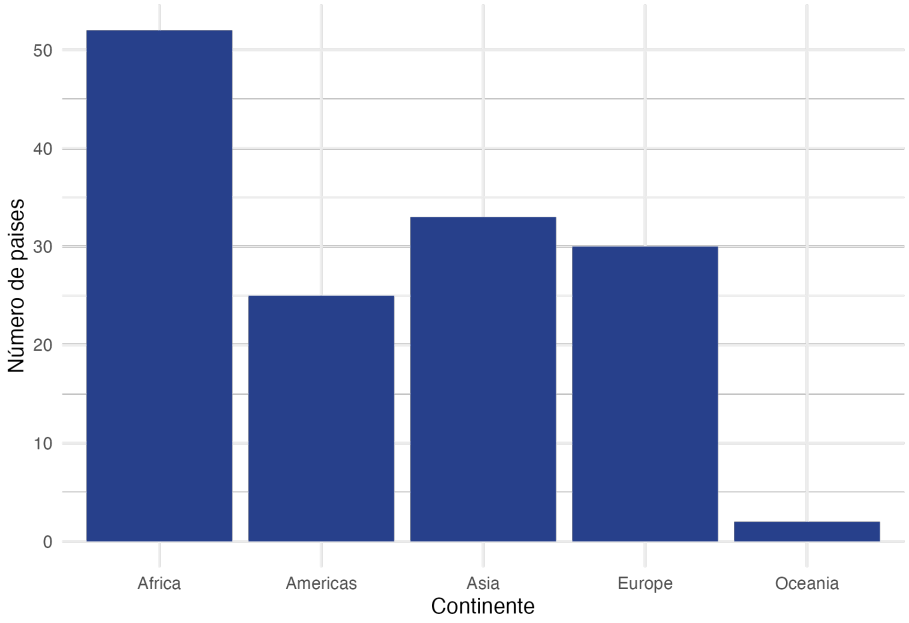
3.2 Gráfico de barras

Los gráficos de barras permiten visualizar la distribución de una variable cualitativa; es decir, variables de clase **character** o **factor**. Este gráfico muestra con barras cuál es la frecuencia con que se observa cada uno de los posibles valores de la variable cualitativa. Este tipo de gráficos se puede construir con la función **geom_bar()**.

Por ejemplo, consideremos la variable `continent`, que es un factor con cinco posibles valores. Visualicemos cómo es la distribución de países por continente en los datos de *gapminder* para el 2007. Es decir, contemos cuántas veces se repite cada continente en los datos de 2007. Esto lo podemos hacer con el siguiente código:

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent)) +
    geom_bar(fill = "royalblue4") +
    labs( y="Frecuencia",
          x="Continente") +
    theme_minimal()
```

Figura 3.3. Distribución de los países por continente disponibles en los datos de gapminder para 2007



Fuente: Datos del paquete gapminder.

Recuerda que existen muchos más parámetros en estas funciones de la geometría que pueden ayudar a mejorar tu visualización. Es importante que mires en la ayuda todas las opciones que tienes.

La misma información anterior puede mostrarse de manera relativa (como porcentaje de todos los países) y no de manera absoluta (el número de países por continente). Este gráfico se conoce como un **gráfico de barras de porcentajes**. Esto lo podemos hacer de varias formas. Una forma es crear una base de datos con una columna que tenga el nombre del continente y otra con el porcentaje de países que el respectivo continente representa. Otra forma es modificar el código anterior, para calcular dicho porcentaje directamente en la función.

Por ejemplo, intenta evaluando el siguiente código:

```
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent,
             y=100 * (..count..)/sum(..count..))) +
  geom_bar( fill = "royalblue4") +
  labs( y="%",
        x="Continente") +
  theme_minimal()
```

Mira con cuidado cómo se creó la variable **y** empleando **..count..**. Esta función cuenta la variable **x** para cada uno de los factores de esta.

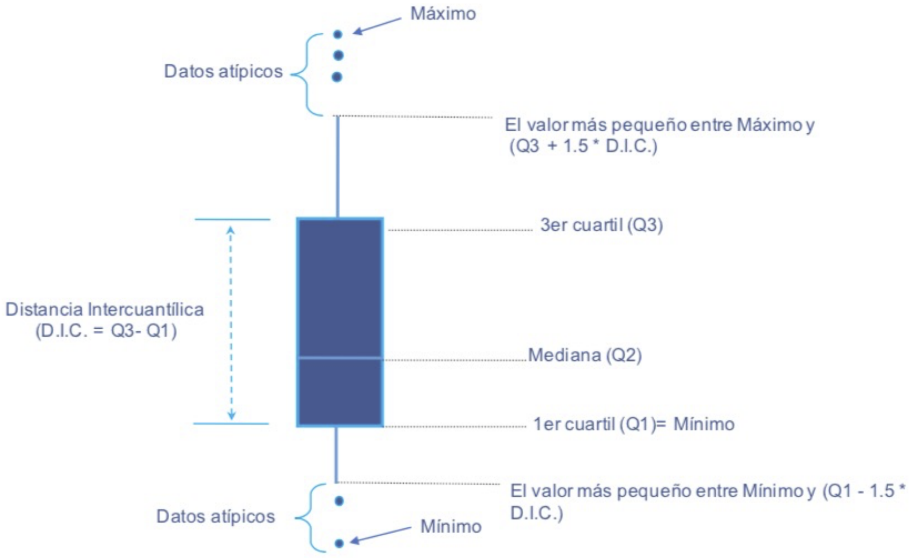
3.3 Boxplot

Los **Boxplot** son conocidos como diagrama de cajas y bigotes o diagrama de cajas. Estos gráficos requieren de una audiencia con una formación en estadística para poder transmitir bien los mensajes poderosos que revelan. En tus cursos de estadística con seguridad estudiaste o estudiarás esta visualización. Por ahora recordemos que esta visualización permite observar el primer, segundo y tercer cuartil, la distancia intercuartílica y la existencia o no de datos atípicos. La Figura 3.4 te permitirá recordar la interpretación de este gráfico.

Este gráfico es muy empleado para comparar la distribución de una variable cuantitativa para los diferentes valores posibles de una variable cualitativa. Típicamente, la variable cualitativa se representa en el eje horizontal y en el eje vertical se representa la variable cuantitativa. La función que permite construir este gráfico es **geom_boxplot()**.

[3]Nota que esto es muy diferente a mapear una variable al color. Aquí solo estamos cambiando el color del histograma.

Figura 3.4. Elementos de un Boxplot



Fuente: Elaboración propia.

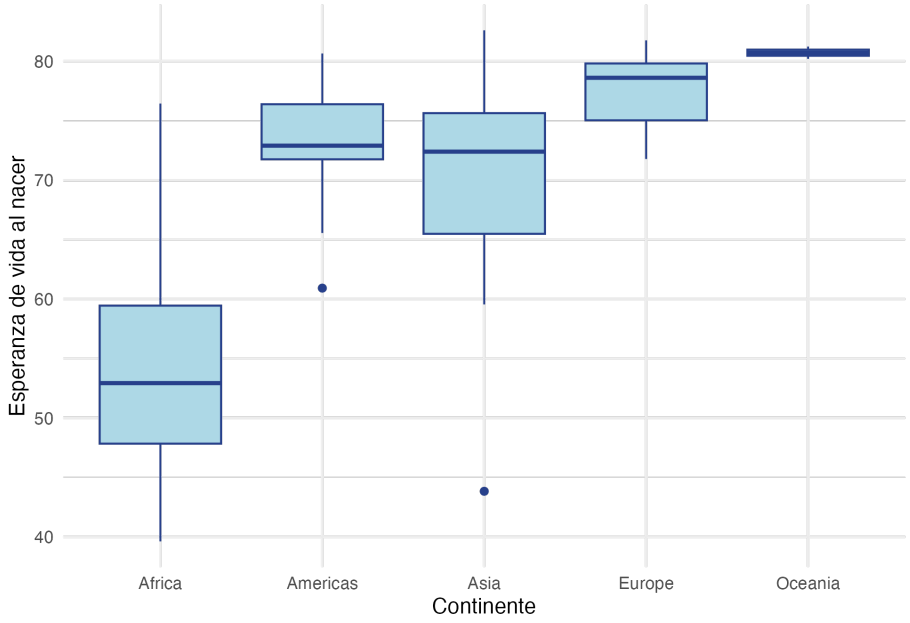
Visualicemos la distribución de la esperanza de vida al nacer por continente para el año 2007. Esto lo podemos hacer de la siguiente manera:

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent, y = lifeExp)) +
    geom_boxplot(fill = "lightblue",
                 col = "royalblue4" ) +
    labs( y="Esperanza de vida al nacer",
          x="Continente") +
    theme_minimal()
```

3.4 Comentarios finales

Antes de continuar con las capas de ***Geometría*** que permiten mostrar evolución de una o más variables, es importante anotar que existen

Figura 3.5. Boxplot de la esperanza de vida al nacer por continente para 2007.



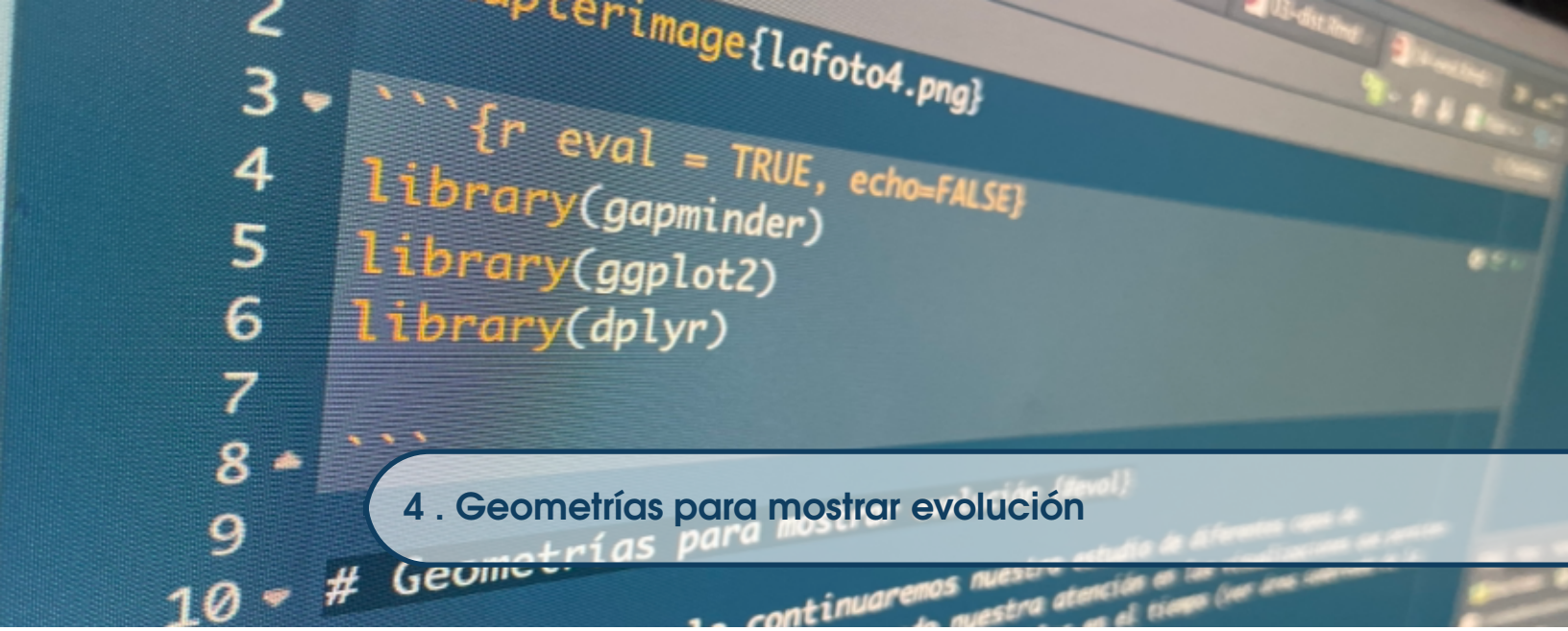
Fuente: Elaboración propia.

otras visualizaciones no tan comunes para mostrar distribución. Por ejemplo, está el gráfico de densidad, diagrama de violines[4], los gráficos de donas, los gráficos de waffles, los gráficos de bombones, los treemaps (como la Figura 1.4), los mapas de calor, los diagramas de Sankey y las nubes de palabras[5]. Si te interesa conocer más de estas visualizaciones puedes encontrar una amplia documentación en línea mantenida por la comunidad de usuarios de R y *ggplot2*. También, en el Capítulo 9 de este libro puedes aprender a construir algunas de estas visualizaciones más avanzadas.

Ya conoces la lógica y la gramática que emplea este paquete. Con seguridad, será muy sencillo entender y modificar los códigos que encuentres disponibles en línea. Y no olvides compartir los códigos que creas de utilidad para los miembros de la comunidad.

[4]Si quieres conocer sobre los diagramas de violines, puedes ver una breve introducción en el siguiente enlace: https://bit.ly/3Jc71YC. En el Capítulo 9 puedes aprender a construir este tipo de visualizaciones.

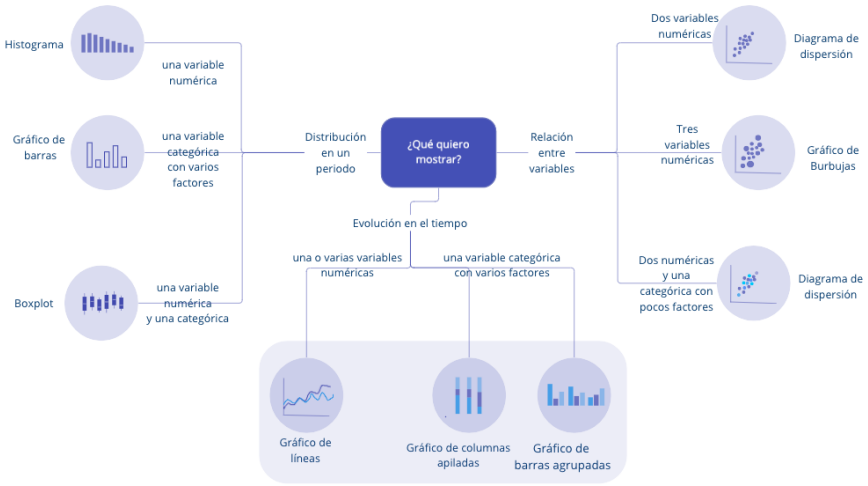
[5]Si quieres conocer sobre las nubes de palabras, puedes ver una breve introducción en el siguiente enlace: https://bit.ly/3HO46Fa.



4 . Geometrías para mostrar evolución

En este capítulo continuaremos nuestro estudio de diferentes capas de ***Geometría***, concentrando nuestra atención en las visualizaciones que permiten mostrar evolución de una o más variables en el tiempo (ver área sombreada de la Figura 4.1).

Figura 4.1. Tipos de gráficos más comunes según lo que se desea comunicar y la clase de variable



Fuente: Adaptación de Alonso y González (2012).

Los gráficos más comunes para mostrar evolución en el tiempo de una o más variables son:

- **Gráficos de líneas**
- **Gráficos de barras agrupadas**
- **Gráfico de columnas apiladas** o barras apiladas

Al igual que en los capítulos anteriores, emplearemos en nuestros ejemplos los datos del objeto `gapminder` del paquete del mismo nombre.

4.1 Gráfico de líneas

Los gráficos de líneas son apropiados para visualizar el comportamiento histórico de una o más variables de clase **numeric** (variable cuantitativa continua) e **integer** (variable cuantitativa discreta). También, nos permiten comparar el comportamiento en el tiempo de dos o más variables.

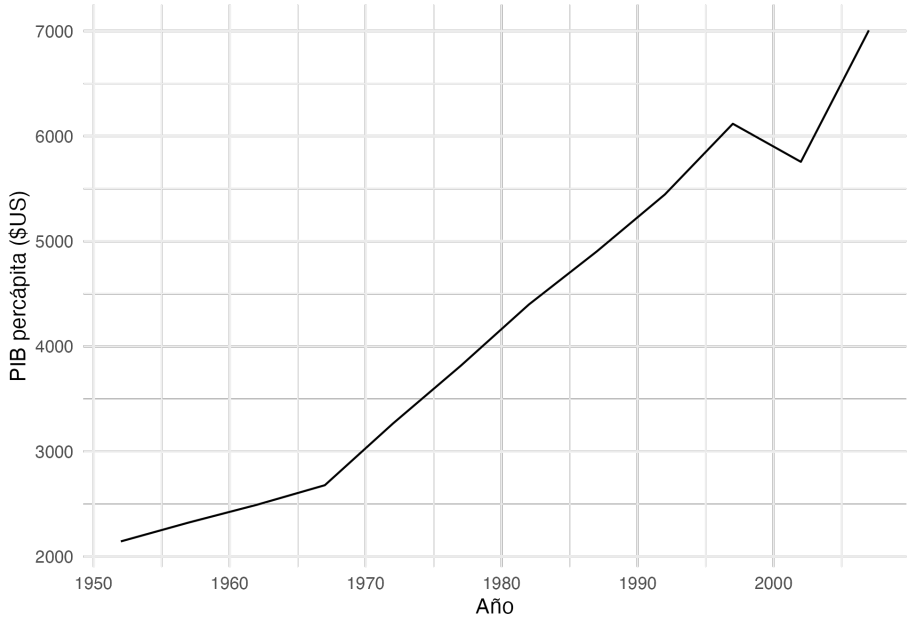
Estos gráficos reportan en el eje horizontal el período de tiempo (años, trimestres, meses o días según sea el caso) y en el eje vertical la variable de interés. Para cada periodo se identifica el correspondiente valor de la variable y todos los puntos se unen con líneas rectas. De ahí su nombre de gráfico de líneas. La función **geom_line()** permite construir este tipo de visulizaciones.

Por ejemplo, grafiquemos la evolución del PIB percápita de Colombia. En este caso recuerda que en el eje horizontal tendremos el año (**x = year**) y en el eje vertical tendremos el PIB percápita (**y =gdpPercap**). Entonces, primero filtramos los datos para Colombia y los pasamos a la capa de ***Datos***. Posteriormente, agregamos la capa de ***Aesthetics*** y finalmente la capa de ***Geometría*** con la función **geom_line()**. Agregamos una capa de **Escalas** para poner las etiquetas y una de **Tema** para mejorar la apariencia.

```
# cargamos los paquetes
library(ggplot2)
library(dplyr)
library(gapminder)

# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
   filter(country == "Colombia") %>%
    ggplot( aes(x = year, y =gdpPercap)) +
    geom_line() +
    labs( y="PIB percápita ($US)",
```

```
          x="Año") +
   theme_minimal()
```

Figura 4.2. Evolución del PIB percápita de Colombia (1952 - 2007) (datos cada 5 años)



Fuente: Datos del paquete gapminder.

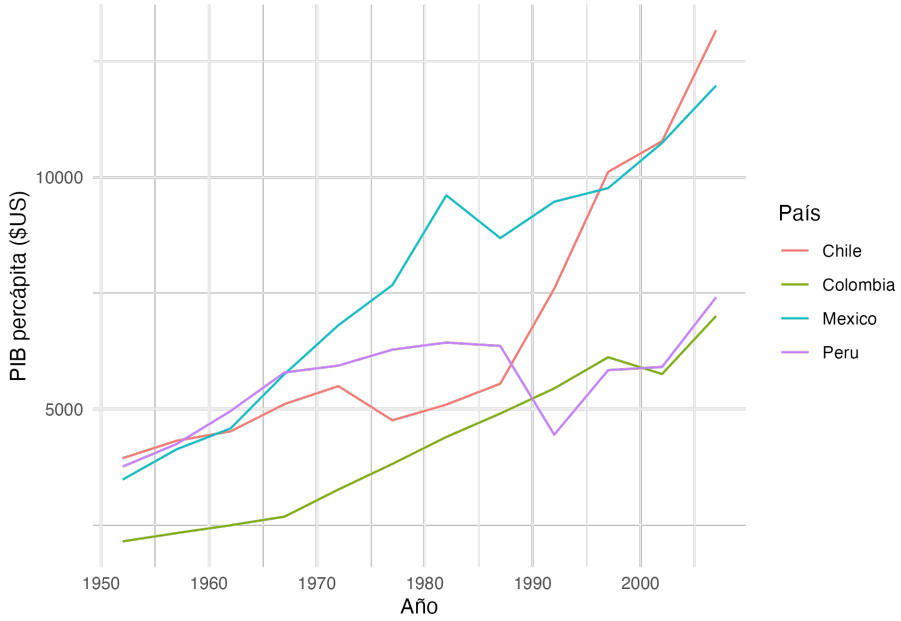
La Figura 4.2 presenta la evolución del PIB percápita de Colombia. En algunas ocasiones, también queremos incluir otros países para, además, comparar la evolución en el tiempo entre países. Esto lo podemos hacer rápidamente. En este caso, dada la estructura de los datos, podemos incluir la variable país de clase **factor** en el color en la capa ***Aesthetics***.

Por ejemplo, grafiquemos la evolución del PIB percápita de los países de la Alianza del Pacífico (Colombia, Chile, Perú y México). Esto lo podemos hacer con el siguiente código:

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
   filter(country %in%
```

```
          c("Colombia", "Peru", "Chile", "Mexico")) %>%
ggplot( aes(x = year, y =gdpPercap, color = country)) +
 geom_line() +
 labs( y="PIB percápita ($US)",
       x="Año", color = "País") +
 theme_minimal()
```

Figura 4.3. Evolución del PIB percápita de los países de la Alianza del Pacífico (1952 - 2007) (datos cada 5 años).



Fuente: Datos del paquete gapminder.

La Figura 4.3 nos permite ver la evolución en el tiempo del PIB percápita de los cuatro países. Esta tal vez es una de las visualizaciones más populares.

4.2 Barras agrupadas

Las barras agrupadas permiten comparar la evolución de la composición en el tiempo de una variable cualitativa de clase **factor**. Intuitivamente, esta visualización es un gráfico de barras (ver Sección 3.2) reproducido en diferentes periodos. Es decir, permite representar cómo cambia en el tiempo la participación porcentual en el total de cada uno de los posibles valores de la variable cualitativa. Típicamente, en el eje horizontal tendremos los períodos y en el eje vertical mediremos la participación porcentual de cada uno de los factores. El gráfico de barras agrupadas también lo podemos construir con la función **geom_bar()**. La diferencia en este caso es que tendremos que especificar qué variable se mapea al eje horizontal y cuál al vertical.

En la Sección 3.2 construimos un gráfico de barras (ver Figura 3.3) para la distribución de los países por continente disponibles en los datos del paquete *gapminder*. Eso lo realizamos para el año 2007. En esta oportunidad nos gustaría tener el mismo gráfico (en porcentaje) pero para todos los años que hacen parte del objeto *gapminder*. Para construir el gráfico necesitamos contar con un objeto de datos que tenga una columna con el número de países por continente (o la participación en el total de países) por año. En esta ocasión, por razones pedagógicas, separemos la preparación de la base de datos de la construcción de la visualización.

Los datos para nuestra visualización se pueden construir con el siguiente código[1]:

```
d1 <- gapminder %>%
  # se agrupan los casos por año y continente
  group_by(year, continent) %>%
  # se cuentan los países por cada continente, para cada año
  # frecuencia
  summarise( paises = n()) %>%
  # se crea la nueva variable con la frecuencia relativa
  mutate(Prop_paises = 100 * paises / sum(paises))

glimpse(d1)
```

```
## Rows: 60
## Columns: 4
## Groups: year [12]
## $ year        <int> 1952, 1952, 1952, 1952, 1952, 1957, 1957,~
## $ continent   <fct> Africa, Americas, Asia, Europe, Oceania, ~
```

[1] Asegúrate que entiendes cómo se resumen las observaciones y se crean las variables. Si necesitas ayuda, puedes ver el Capítulo 2 de Alonso (2022).

```
## $ paises      <int> 52, 25, 33, 30, 2, 52, 25, 33, 30, 2, 52,~
## $ Prop_paises <dbl> 36.619718, 17.605634, 23.239437, 21.12676~
```

Ahora, podemos crear la visualización empleando en la capa de ***Datos*** el objeto `d1`. En la capa de ***Aesthetics*** podemos mapear los años al eje horizontal (**x = as.character(year)**)[2]; en el eje vertical la participación porcentual de cada continente (**y = Prop_paises**); y el relleno de las barras corresponderá al respectivo continente (**fill = continent**). En la capa de ***Geometría*** emplearemos la función **geom_bar()**. Para esta visualización necesitamos dos argumentos. El primero es la posición de las barras que estarán una al lado de la otra. Para esto tendremos el siguiente argumento: **position = "dodge"**. El segundo argumento corresponde a qué estadística queremos que se represente en el eje vertical. Por defecto, **geom_bar()** cuenta el número de casos en cada grupo y lo pone en el eje vertical (**stat="bin"**)[3]. En este caso queremos que se mapee al eje vertical los valores de una columna (**y = Prop_paises**). Esto lo logramos poniendo este argumento igual a **"identity"** (**stat = "identity"**).

```
ggplot(d1, aes(x = as.character(year), y = Prop_paises,
               fill = continent)) +
  geom_bar(position = "dodge", stat = "identity") +
  labs(y = "% del número de países total",
       x = "Año", fill = "Continente") +
  theme_minimal()
```

La Figura 4.4 muestra que no ha tenido cambio la participación de los continentes durante el periodo 1952-2007. Este gráfico en esta ocasión no es interesante, pero esto dependerá de la base de datos que tengas.

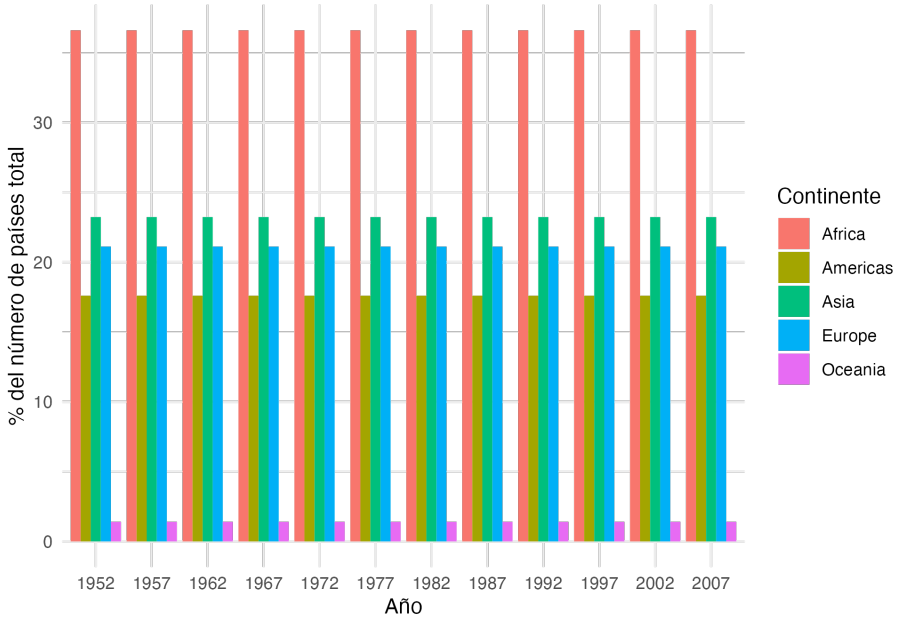
El gráfico de barras agrupadas también puede ser empleado para mostrar la evolución de una variable cuantitativa y una cualitativa. Por ejemplo, podemos ver cómo ha cambiado la participación en la población mundial de cada continente. Esto lo podemos hacer con el siguiente código que genera la Figura 4.5. Puedes observar que Asia es el continente más poblado y su participación ha venido creciendo levemente.

```
d2 <- gapminder %>%
      group_by(year, continent) %>%
      summarise( Poblacion = sum(pop)) %>%
      group_by(year) %>%
```

[2] Aquí estamos empleando un truco. Pasando los años como texto hace que aparezcan todos los años en el eje. Intenta mapear la variable año sin la transformación a texto (**x = year**) y verás que no aparecerán todos los años.

[3] Nota que esto fue lo que hicimos cuando creamos el gráfico de barras que se reporta en la Figura 3.3.

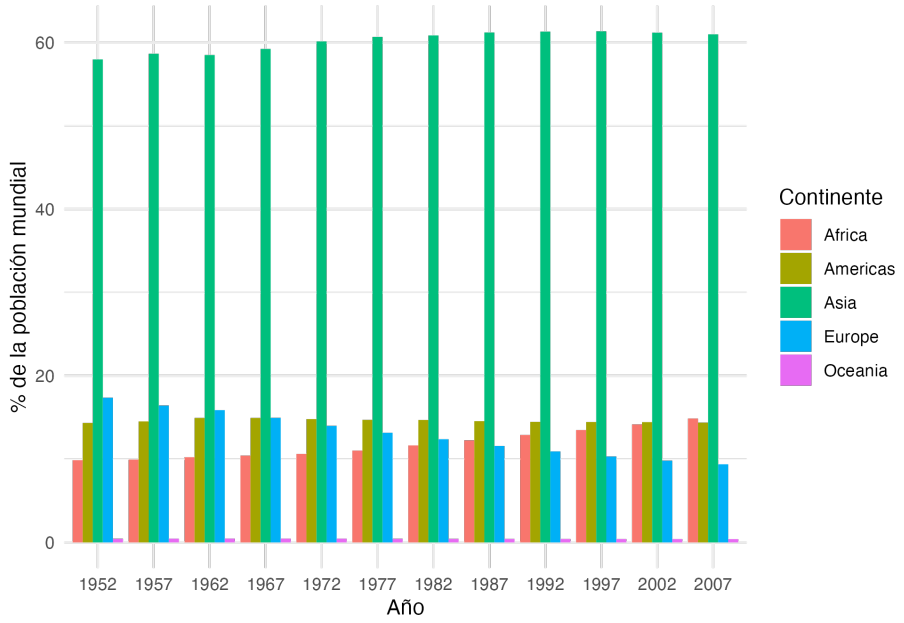
Figura 4.4. Evolución de la distribución de los países por continente disponibles en los datos de gapminder (1952 - 2007)



Fuente: Datos del paquete gapminder.

```
      mutate(Prop_Poblacion = 100 * Poblacion / sum(Poblacion))

ggplot(d2, aes(x = as.character(year),
               y = Prop_Poblacion, fill = continent))+
  geom_bar(position = "dodge", stat="identity")+
  labs(y = "% de la población mundial",
       x = "Año", fill = "Continente")+
  theme_minimal()
```

Figura 4.5. Evolución de la distribución de la población mundial por continente (1952 - 2007)

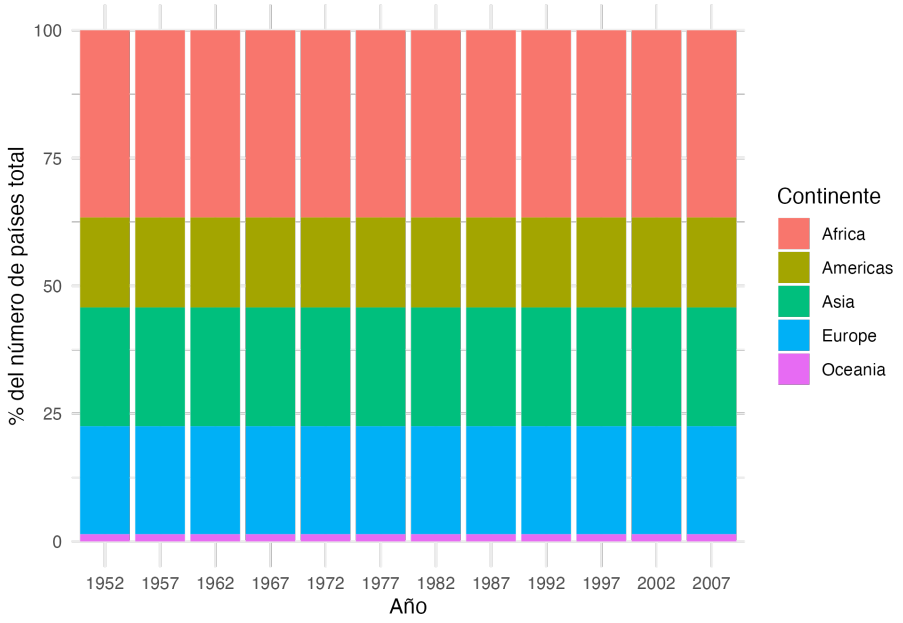


Fuente: Cálculos propios empleando datos del paquete gapminder.

4.3 Columnas apiladas

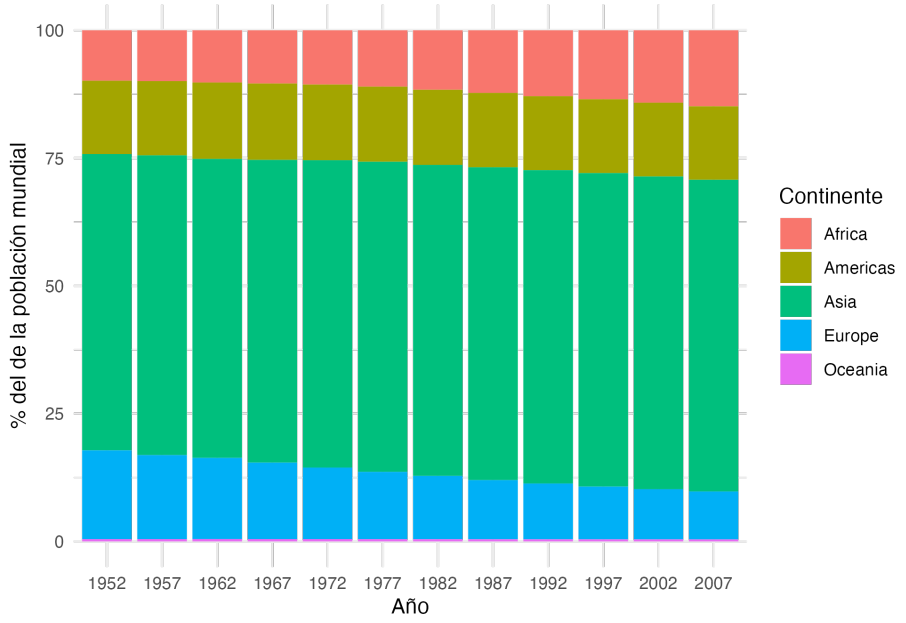
Otra alternativa a las barras agrupadas son las barras apiladas. En este caso, como su nombre lo implica, las barras están una encima de la otra. Esta visualización implica una leve modificación al código que genera las barras agrupadas. En el argumento de posición de la función **geom_bar()** empleamos **position = "stack"**. Intenta modificar el código para obtener las Figuras 4.6 y 4.7.

Figura 4.6. Evolución de la distribución de los países por continente disponibles en los datos de gapminder (1952 - 2007)



Fuente: Cálculos propios empleando datos del paquete gapminder.

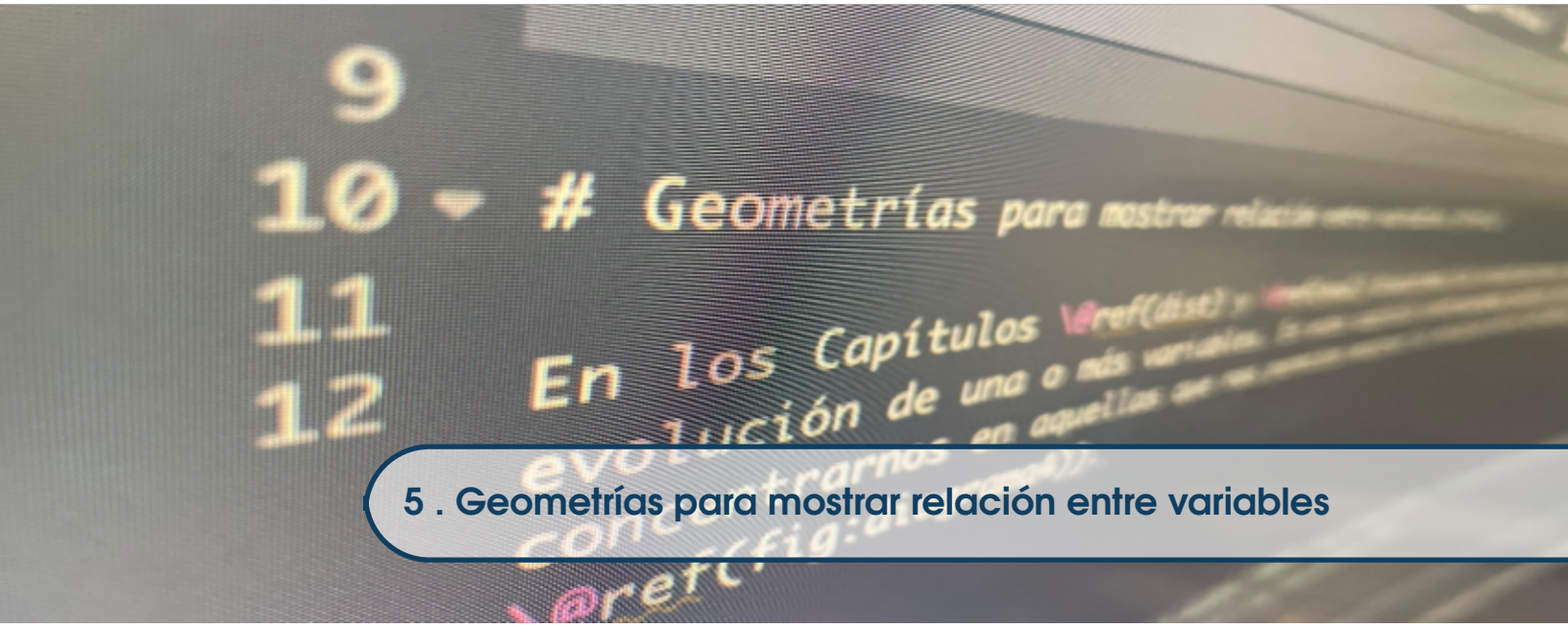
Figura 4.7. Evolución de la distribución de la población mundial por continente (1952 - 2007)



Fuente: Cálculos propios empleando datos del paquete gapminder.

4.4 Comentarios finales

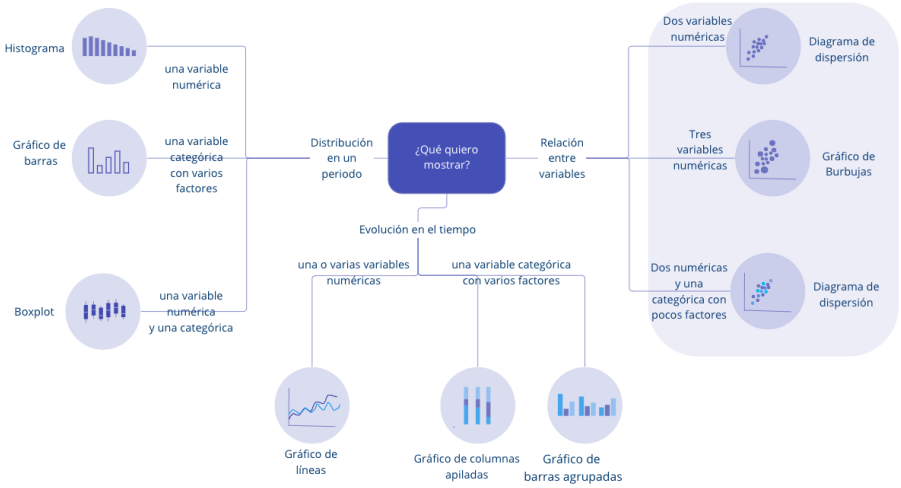
En este capítulo evidenciamos que a veces crear visualizaciones implica transformar un objeto con datos para tener el conjunto de estos mismos de la manera más adecuada, y así, poder generar la visualización. Es así como el paquete *dplyr* (Wickham et al., 2023a) y el paquete *ggplot2* presentan una integración ideal para facilitar el flujo de trabajo. Si aún no dominas el paquete *dplyr* puedes ver Alonso (2022). Con seguridad, si dominas estos dos paquetes, podrás generar visualizaciones de gran impacto en tu audiencia. Los dos paquetes te darán una gran versatilidad para mejorar la apariencia de tus visualizaciones. De hecho, en el Capítulo 6 discutiremos algunas estrategias que te serán muy útiles al momento de generar gráficos; pero antes, estudiaremos las visualizaciones que permiten mostrar relación entre variables (Capítulo 5).



5 . Geometrías para mostrar relación entre variables

En los Capítulos 3 y 4 discutimos las visualizaciones más comunes para mostrar la distribución y la evolución de una o más variables. En este capítulo continuaremos nuestro estudio de diferentes capas de ***Geometría*** para concentrarnos en aquellas que nos permiten mostrar la relación entre variables (ver parte sombreada de la Figura 5.1).

Figura 5.1. Tipos de gráficos más comunes según lo que se desea comunicar y la clase de variable



Fuente: Adaptación de Alonso y González (2012).

La gráfica más empleada para mostrar la relación entre dos variables es el diagrama de dispersión. Esta visualización es muy versátil y nos

permite incluir aún más variables si jugamos con el tamaño de los puntos (conocido como gráfico de burbujas), su color o su tamaño. En este capítulo nos concentraremos en este gráfico.

5.1 Diagrama de dispersión

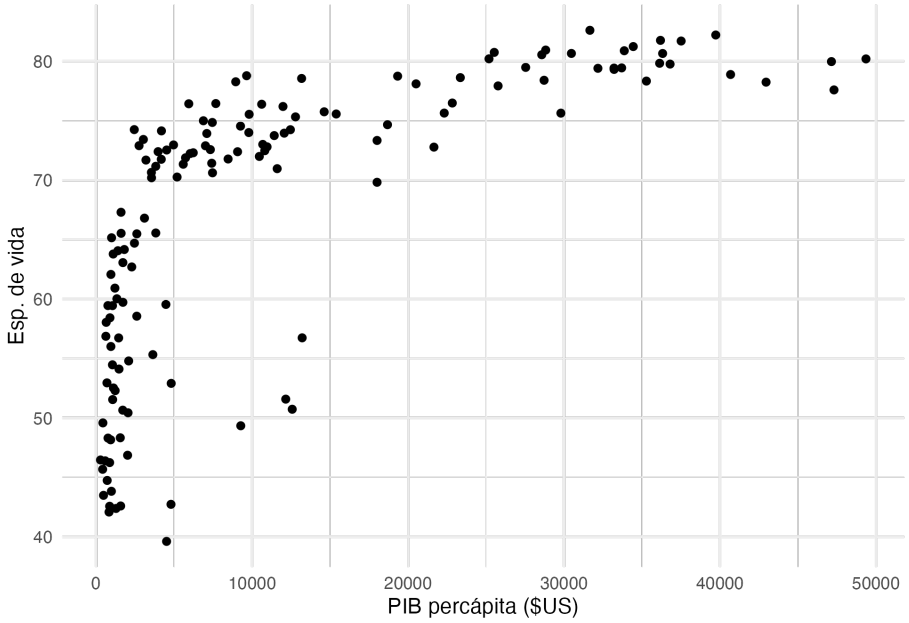
Cuando se busca mostrar una relación entre dos variables numéricas, lo mejor es hacer uso de un diagrama de dispersión. En la Figura 5.2 se muestra un ejemplo de la relación entre la expectativa de vida y el PIB percápita para todos los países en 2007. Este fue el primer gráfico que aprendimos ha construir en el Capítulo 2.

Recuerda que este gráfico lo podemos generar con la función **geom_point()** empleando el siguiente código:

```
# cargamos los paquetes
library(ggplot2)
library(dplyr)
library(gapminder)

# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  labs(  x="PIB percápita", y="Esp. de vida") +
  theme_minimal()
```

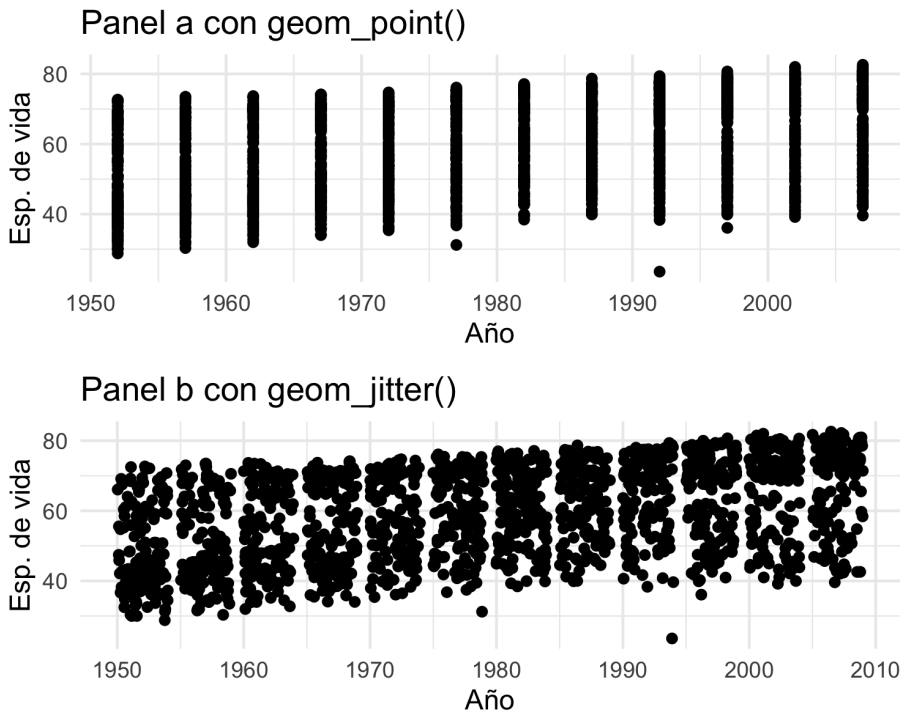
Es importante anotar que, en algunas ocasiones, se pueden tener muchos puntos que se superponen (en inglés *over-plotting*) , y se tapan entre sí cuando se mapean. En esos casos, nos podemos dar unas libertades "artísticas" añadiendo una pequeña cantidad de variación aleatoria a la ubicación de cada punto. De esta manera, los puntos no se sobreponen y la visualización nos permitirá entender mejor los datos. Esto se logra con la función **geom_jitter()** . Por ejemplo, en el panel a de la Figura 5.3 se muestra un diagrama de dispersión con todos los datos de *gapminder* con el año en el eje horizontal y la esperanza de vida al nacer en el vertical empleando la función **geom_point()**. Observa que los puntos se ven sobrepuestos. Esta visualización es apenas natural (asegúrate que te queda claro el porqué obtenemos esta visualización).

Figura 5.2. PIB percápita y expectativa de vida al nacer por país (2007).



Fuente: Datos del paquete gapminder.

Figura 5.3. Relación entre la expectativa de vida al nacer y el año.



Fuente: Datos del paquete gapminder.

En el panel b de la Figura 5.3 se muestran los mismo datos, pero tomándonos una libertad "artística" de mover un poco horizontal y verticalmente los puntos para que no se sobrepongan. Esta vizualización fue creada con la función **geom_jitter()** . Si bien, esta visualización no es exacta, brinda una mejor idea de cómo se relacionan el año y la esperanza de vida al nacer. Del contexto de tu visualización, podrás tomar la decisión de si tiene o no sentido emplear la función **geom_jitter()** en vez de **geom_point()**. La Figura 5.3 fue creada con el siguiente código:

```
# panel a
ggplot(gapminder, aes(x = year, y = lifeExp)) +
                geom_point() +
                labs(title = "Panel a con geom_point()",
                     x="Año", y="Esp. de vida") +
               theme_minimal()
# panel b
ggplot(gapminder, aes(x = year, y = lifeExp)) +
                geom_jitter() +
                labs(title = "Panel b con geom_jitter()",
                     x="Año", y="Esp. de vida") +
               theme_minimal()
```

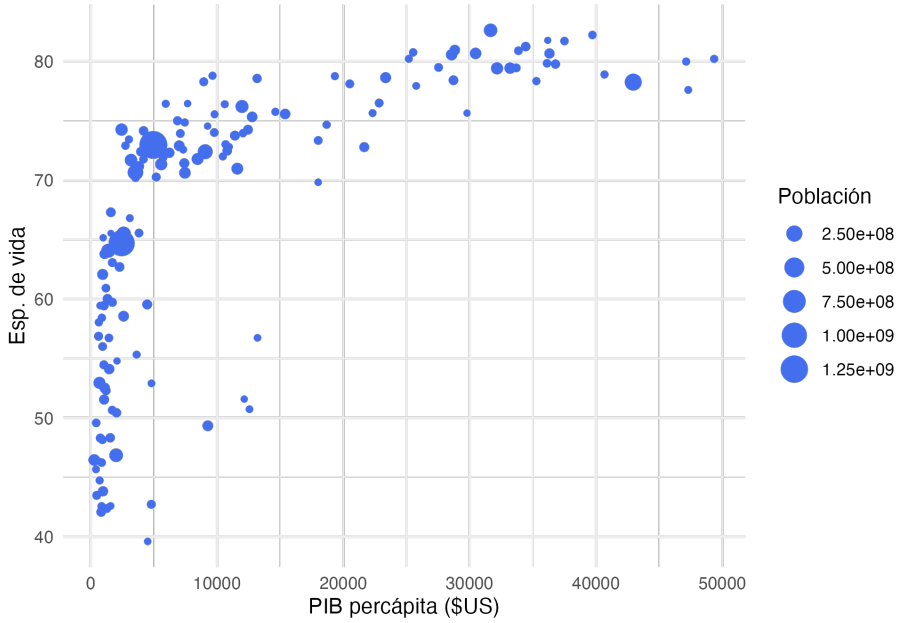
5.2 Gráfico de burbujas

En algunas situaciones, será interesante incluir una tercera variable numérica a nuestra visualización. El diagrama de dispersión permite una tercera variable numérica si la mapeamos al tamaño del punto. Esto tendrá una apariencia de burbujas, por eso el nombre de esta visualización. En general será mucho más clara y transmitirá mejor el mensaje que emplear gráficas en tres dimensiones.

Generarla es muy sencillo. Tendremos que mapear nuestra tercera variable numérica al argumento **size** de la capa **Aesthetics**. La Figura 5.4 presenta un gráfico de burbujas construido con el siguiente código:

```
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         size="Población") +
  theme_minimal()
```

Figura 5.4. PIB percápita, expectativa de vida al nacer y población por país (2007)



Fuente: Datos del paquete gapminder.

5.3 Diagrama de dispersión y variables cualitativas

También podemos incluir variables cualitativas en un diagrama de dispersión. Esto lo podemos hacer mapeando la variable cualitativa al color de los puntos. Esto nos da bastante flexibilidad en nuestros gráficos. Es más, podríamos incluir una quinta variable en nuestro diagrama de dispersión si mapeamos una variable cualitativa a la forma del punto. Pero tenemos que ser cuidadosos de no saturar nuestra visualización. Típicamente en las visualizaciones "menos es más".

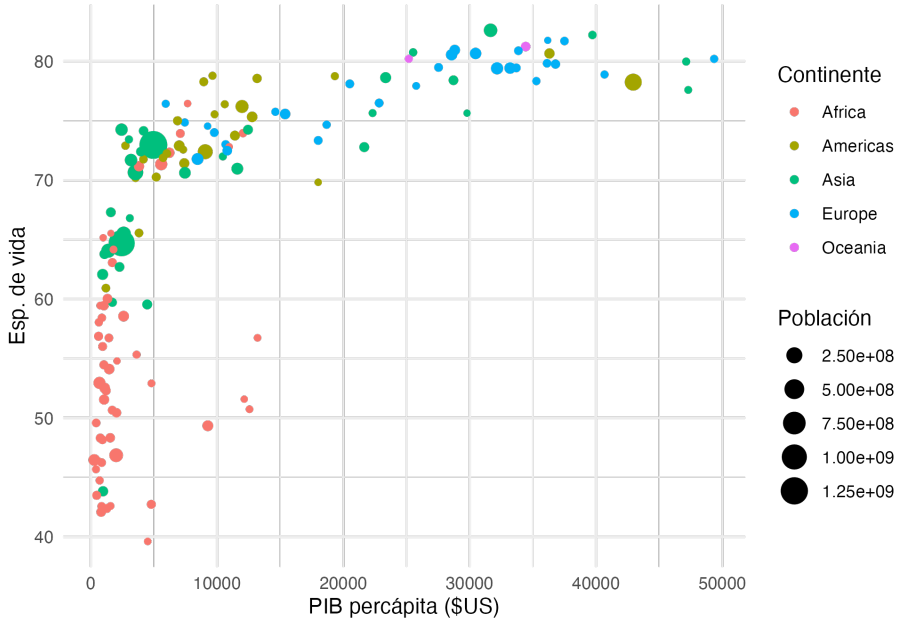
Por ejemplo, incluyamos la variable continente a nuestro gráfico de burbujas. El siguiente código permite generar la Figura 5.5.

```
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop, col = continent )) +
  geom_point() +
  labs(  x = "PIB percápita ($US)",
         y = "Esp. de vida",
         size = "Población",
         col = "Continente") +
  theme_minimal()
```

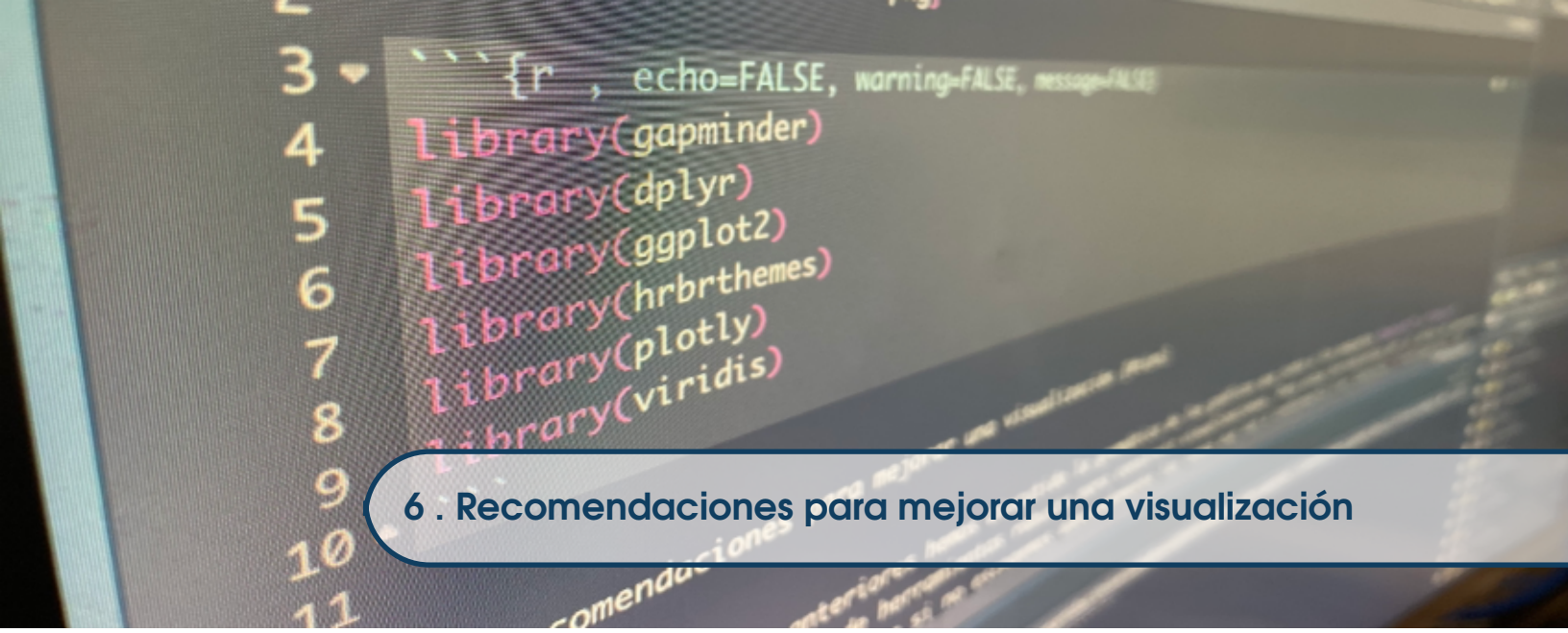
5.4 Comentarios finales

En este capítulo hemos trabajado tres visualizaciones más. Con esto completamos 9 tipos de gráficos que nos permiten mostrar distribución, evolución y relación entre variables. Ya tienes en tu caja de herramientas estas poderosas funciones que te permitirán mostrar conjuntos de datos grandes o pequeños. En el Capítulo 6 discutiremos unas estrategias para mejorar la apariencia de tus visualizaciones.

Figura 5.5. PIB percápita, expectativa de vida al nacer y población por país (2007)



Fuente: Datos del paquete gapminder.



6 . Recomendaciones para mejorar una visualización

En los capítulos anteriores hemos discutido la gramática de los gráficos que junto a los paquetes *ggplot2* y *dplyr* constituyen una caja de herramientas robusta para construir visualizaciones. Pero estas herramientas por sí solas no generan una visualización efectiva si no escogemos adecuadamente las capas que van a conformarla (ver Capítulo 2). En especial, en los Capítulos 3, 4 y 5 se discutió la importancia de seleccionar el tipo de gráfico (capa de ***Geometría***) de acuerdo a lo que queremos mostrar y a la clase de variables que tenemos.

Hasta aquí nos hemos concentrado más en la parte técnica de cómo generar un gráfico, pero es innegable que una buena visualización implica también unas nociones de estética y, por supuesto, un mensaje que comunicar. De hecho, una visualización busca contar una historia y, para que se cumpla ese objetivo, debemos visualizar nuestros datos de la manera más estética y fiel a ellos.

Como se mencionaba con anterioridad, crear una buena visualización no es tarea fácil. Implica ensayo y error. Es un proceso iterativo; difícilmente lograremos la mejor al primer intento.

Para empezar la construcción de una visualización siempre deberíamos responder por lo menos las siguientes tres preguntas:

- ¿Cuál es el mensaje que queremos comunicar?
- ¿Qué variables de las disponibles permiten comunicar el mensaje?
- ¿Quién es mi audiencia?

Estas preguntas nos ayudan a determinar las capas de la visualización. Y, al momento de construir nuestras visualizaciones, deberíamos tener en cuenta que nuestros gráficos deberán siempre:

- **Mostrar fielmente los datos**. Esto ayuda a que las decisiones que

puedan ser tomadas con las visualizaciones no estén sesgadas a datos distorsionados.
- **Reducir las distracciones**. Es decir, no incluir elementos innecesarios en nuestras visualizaciones. En la construcción de una buena visualización, ¡menos es más!
- **Agregar texto cuando sea posible**. El texto facilita la comunicación y nos permite enfatizar nuestro mensaje.

En este capítulo encontrarás cuatro recomendaciones prácticas que te permitirán implementar estos tres elementos, mejorando tus visualizaciones de manera rápida y fácil al emplear *ggplot2*.

6.1 Ordena los datos

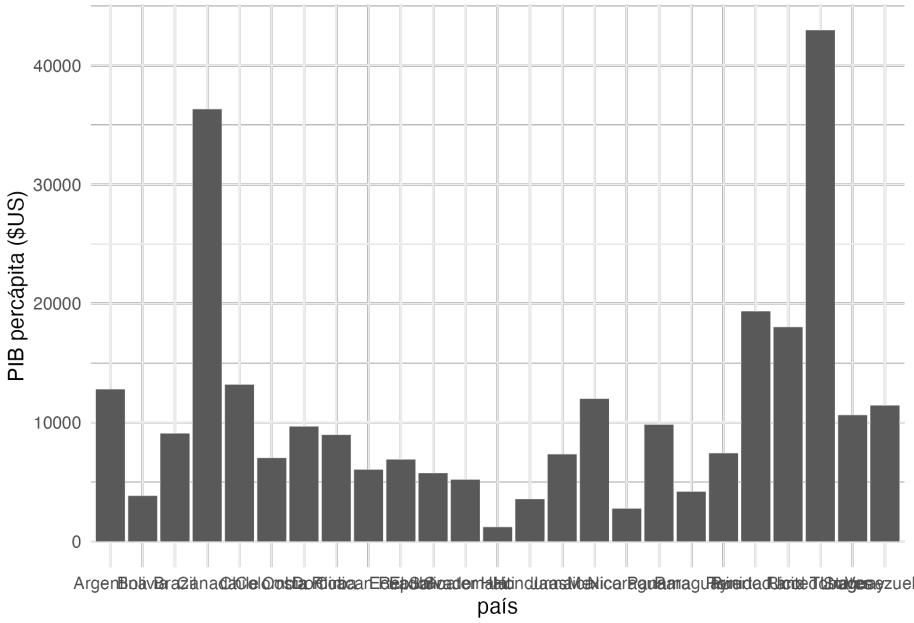
La visualización puede tener un mayor impacto ordenando los datos. La lectura de una visualización con datos ordenados es mucho más sencilla para la audiencia y esto termina por hacerla más simple y efectiva.

Por defecto, las visualizaciones que contienen una variable de clase **character** o **factor**, suelen ordenarse alfabéticamente. Por ejemplo, construyamos una visualización para el PIB percápita del 2007 de todos los países del continente americano. El siguiente código produce el correspondiente gráfico de barras (ver Figura 6.1):

```
# cargar los paquetes
library(ggplot2)
library(dplyr)
library(gapminder)

# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos
gapminder %>%
  filter(year == 2007, continent == c("Americas")) %>%
  ggplot( aes(x=country, y=gdpPercap) ) +
    geom_bar(stat="identity") +
    labs(x = "país", y = "PIB percápita ($US)") +
    theme_minimal()
```

La Figura 6.1 presenta los datos, pero el mensaje no es muy claro. Podemos mejorar sustancialmente la visualización si hacemos dos cosas. Primero, volteemos los ejes en la capa de **Coordenadas** con la función **coord_flip()**. Esto le dará más espacio a los nombres de los países. Y lo

Figura 6.1. Gráfico no ordenado del PIB percápita para los países de América (2007)

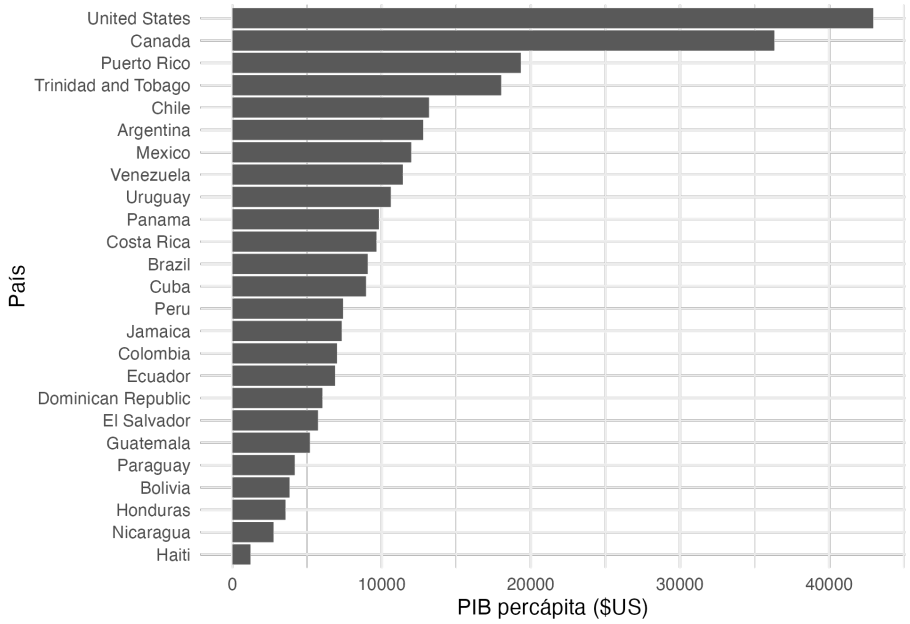


Fuente: Datos del paquete gapminder.

segundo que podemos hacer es ordenar los países por su PIB percápita[1]. El correspondiente código será:

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos
gapminder %>%
  filter(year == 2007, continent == c("Americas")) %>%
  arrange(gdpPercap) %>%
  mutate(country=factor(country, country)) %>%
  ggplot( aes(x=country, y=gdpPercap) ) +
    geom_bar(stat="identity") +
    labs(x = "País", y = "PIB percápita ($US)") +
    coord_flip() +
    theme_minimal()
```

Figura 6.2. Gráfico ordenado del PIB percápita para los países de América (2007)



Fuente: Datos del paquete gapminder.

En la Figura 6.2 se ven mucho más claros los datos, permitiendo que

[1] Además, empleamos un "truco" con la variable país al convertirla en una variable de clase **factor**. ¡Intenta correr el código sin el "truco" y verás la diferencia!

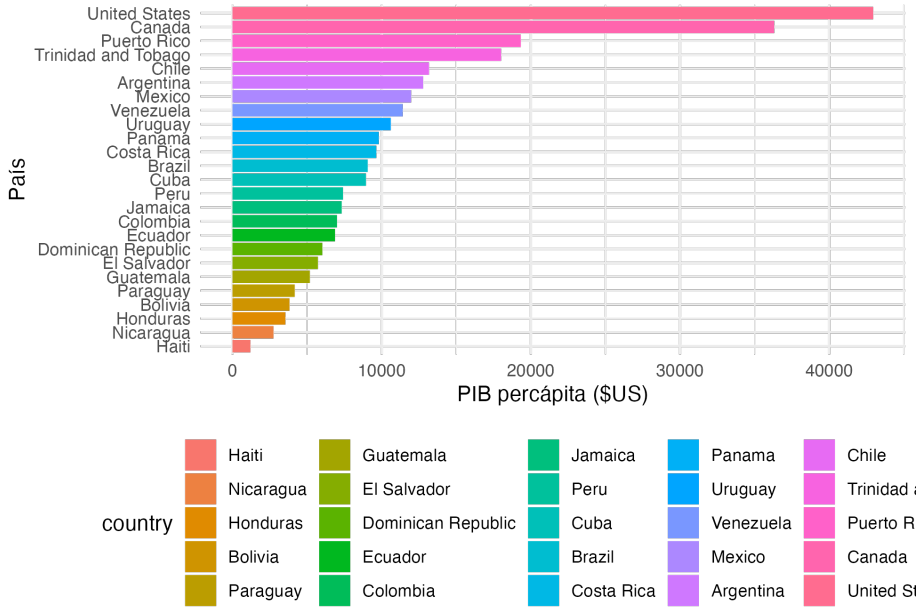
se entiendan fácilmente.

6.2 Ten cuidado con los colores que no comunican

El uso de colores en las visualizaciones debe hacerse con mucho cuidado. Los colores deben transmitir algo, al llamar la atención de la audiencia. Muchos colores pueden resultar engañosos si su uso no es el adecuado. Como ejemplo, se muestra en la Figura 6.3, donde la cantidad de colores hace muy difícil la comparación entre categorías. Es más, los colores son redundantes, pues en el eje vertical ya tenemos el nombre de cada barra. El color no está comunicando nada. Estos colores están distorsionando el mensaje verdadero de la visualización.

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos
gapminder %>%
  filter(year == 2007, continent == c("Americas")) %>%
  arrange(gdpPercap) %>%
  mutate(country=factor(country, country)) %>%
  ggplot( aes(x=country, y=gdpPercap, fill=country) ) +
    geom_bar(stat="identity") +
    labs(x = "País", y = "PIB percápita ($US)",
         color = "País") +
    coord_flip() +
    theme_minimal() +
    theme(legend.position = "bottom")
```

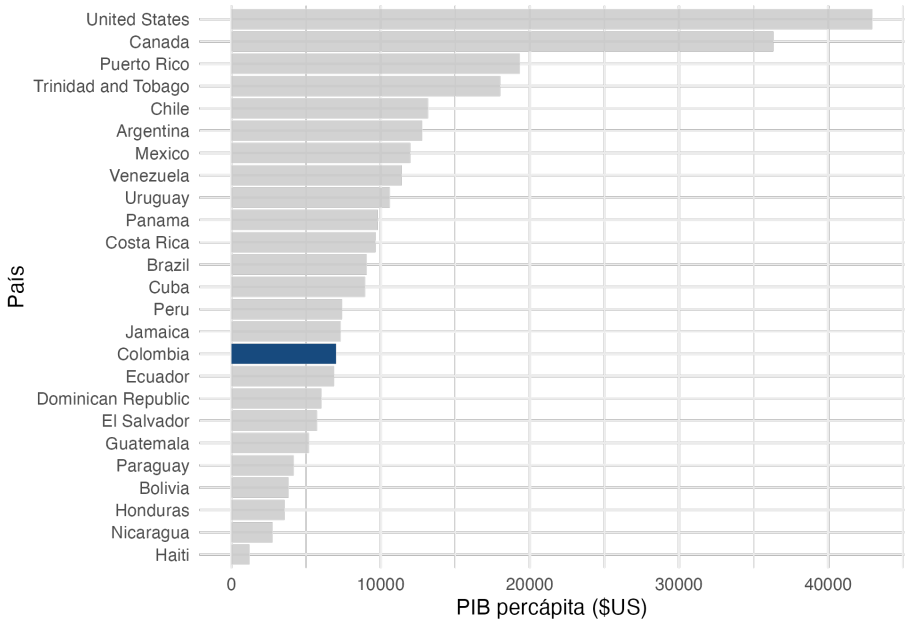
Figura 6.3. Gráfico con colores que no comunican



Fuente: Datos del paquete gapminder.

La Figura 6.3 se puede mejorar usando un solo color como en la Figura 6.2. Sin embargo, los colores sí se pueden emplear para comunicar un mensaje. Si por ejemplo queremos hacer énfasis en el dato de Colombia, podemos ayudarnos con el color. Los colores nos podrían ayudar para resaltar los datos de una sola categoría, como se muestra en la Figura 6.4.

Figura 6.4. Gráfico con color para resaltar



Fuente: Cálculos propios empleando datos del paquete gapminder.

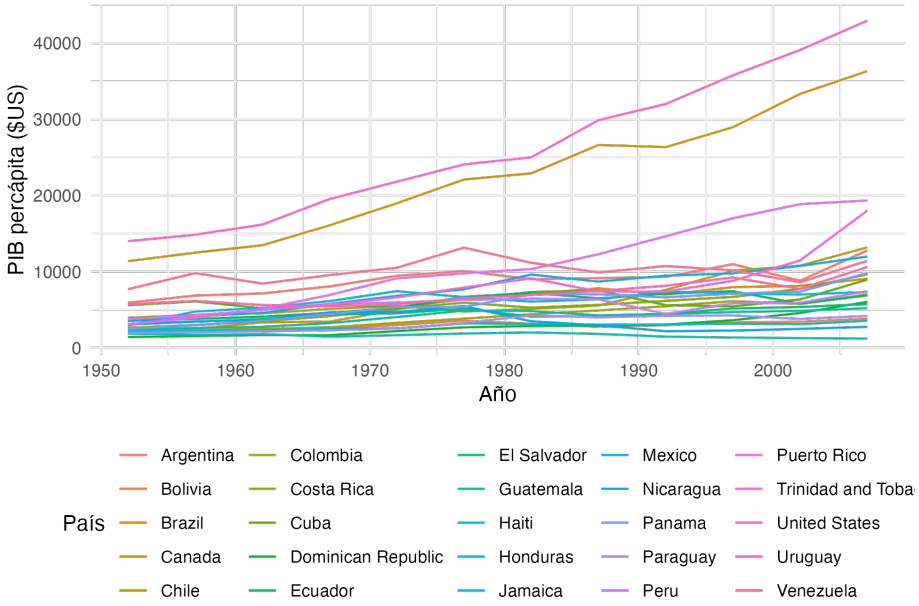
El código que generó la Figura 6.4 es el siguiente. Mira con detalle la última línea. En este caso, empleamos la función **gghighlight()** del paquete *gghighlight* (Yutani, 2020) que permite agregar la posibilidad de resaltar una sola barra.

```
# Cargar el paquete
library(gghighlight)
# Definir el colo azul que se empleará
BLUE1 <- '#174A7E'

gapminder %>%
  filter(year == 2007, continent == c("Americas")) %>%
  arrange(gdpPercap) %>%
  mutate(country=factor(country, country)) %>%
  ggplot( aes(x=country, y=gdpPercap, fill=country) ) +
  geom_bar(stat="identity", fill = BLUE1) +
    labs(x = "País", y = "PIB percápita ($US)",
         color = "País") +
    coord_flip() +
    theme_minimal() +
    theme(legend.position = "bottom") +
  gghighlight(country == 'Colombia', use_direct_label = F)
# la última línea agrega el acento para Colombia
```

6.3 Evita los gráficos *Spaghetti*

Los gráficos de líneas pueden ser confusos si contamos con muchas variables. Por ejemplo, en la Figura 6.5, es complicado seguir la evolución de una sola línea; incluso los colores de la leyenda son difíciles de identificar. Estos gráficos se conocen como gráficos *Spaghetti*, pues se parecen a un plato de Spaghetti donde es difícil identificar un solo *Spaghetti*.

Figura 6.5. Gráfico Spaghetti de la evolución del PIB percápita en los países de América (1955-2007)



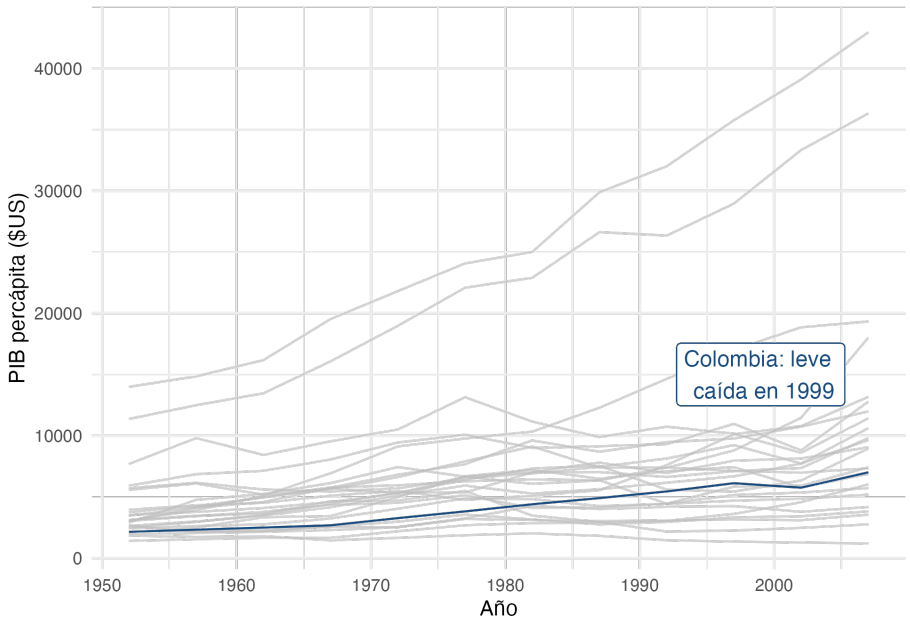
Fuente: Cálculos propios empleando datos del paquete gapminder.

El código que generó la Figura 6.5 es el siguiente (míralo detalladamente):

```
gapminder %>%
  filter(continent == c("Americas")) %>%
  ggplot( aes(x=year, y=gdpPercap,  color=country)) +
    geom_line() +
    labs(x = "Año", y = "PIB percápita ($US)",
         color = "País") +
    theme_minimal() +
    theme(legend.position = "bottom")
```

Ahora, supón que la historia que quieres contar tiene que ver con Colombia. En ese caso, la anterior visualización se puede mejorar empleando un color para resaltar los datos de Colombia (ver la Figura 6.6). Al resaltar la variable que se quiere enfatizar, se reducen las distracciones. Además, podemos eliminar los colores de las demás leyendas. Y finalmente, podemos incluir un texto para enfatizar el mensaje que queramos transmitir.

Figura 6.6. Evolución del PIB percápita en Colombia y otros países de Américа (1955-2007)



Fuente: Cálculos propios empleando datos del paquete gapminder.

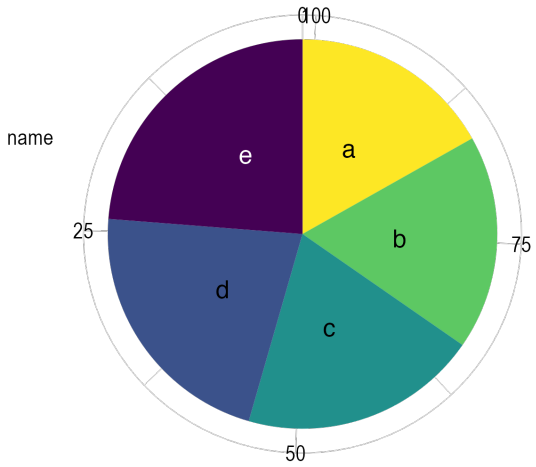
Para generar la Figura 6.6 empleamos por primera vez la función **geom_label()** que incluye una capa con texto (capa de **Texto**). Esta función requiere tres argumentos: los primeros dos corresponden a las coordenadas en las que se pondrá el inicio de la leyenda (argumentos **x** y **y**), el tercer argumento es el texto de la etiqueta (**label**). El código que generó la Figura 6.6 es el siguiente:

```
gapminder %>%
   filter(continent == c("Americas")) %>%
  ggplot( aes(x = year, y = gdpPercap, group = country )) +
    geom_line(col= BLUE1) +
   labs(x = "Año", y = "PIB percápita ($US)",
         color = "País") +
  # agregar la capa de texto
  geom_label( x=1999, y=15000,
              label="Colombia: leve \n caída en 1999",
              size=4, color = BLUE1) +
    theme_minimal() +
    theme(legend.position="none") +
 #  agregar el acento para Colombia
  gghighlight(country == 'Colombia', use_direct_label = F)
```

6.4 No uses gráficos de torta

Los gráficos de torta o pastel (*pie charts*) suelen ser utilizados para mostrar proporciones, donde la suma de todos los grupos da el 100 %. Estos gráficos no son recomendados, porque el ojo humano no es bueno midiendo ángulos. El uso de los gráficos de pastel termina distorsionando la información al no lograr comunicar de forma efectiva el mensaje. Resulta complicado ordenar o determinar qué grupo es el más grande, como se ve en la Figura 6.7.

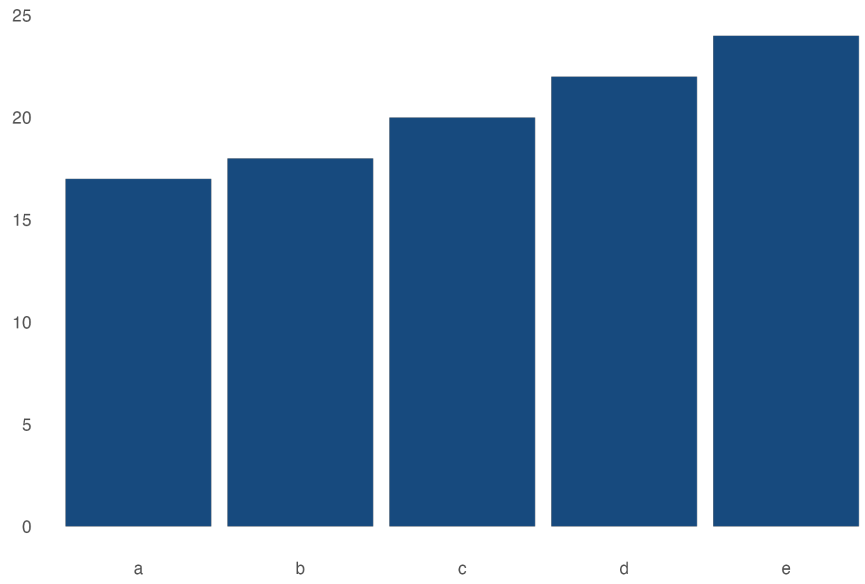
Figura 6.7. Gráfico de torta



Fuente: Cálculos propios empleando datos simulados.

Una alternativa para presentar los datos con mayor claridad es el gráfico de barras (ver Sección 3.2), con el que se presenta en la Figura 6.8. Esta visualización resulta mucho más sencilla de leer, y corresponde a los mismos datos de la Figura 6.7.

Figura 6.8. Gráfico de barras como alternativa al gráfico de pastel (se emplean los mismos datos del grafico de pastel)



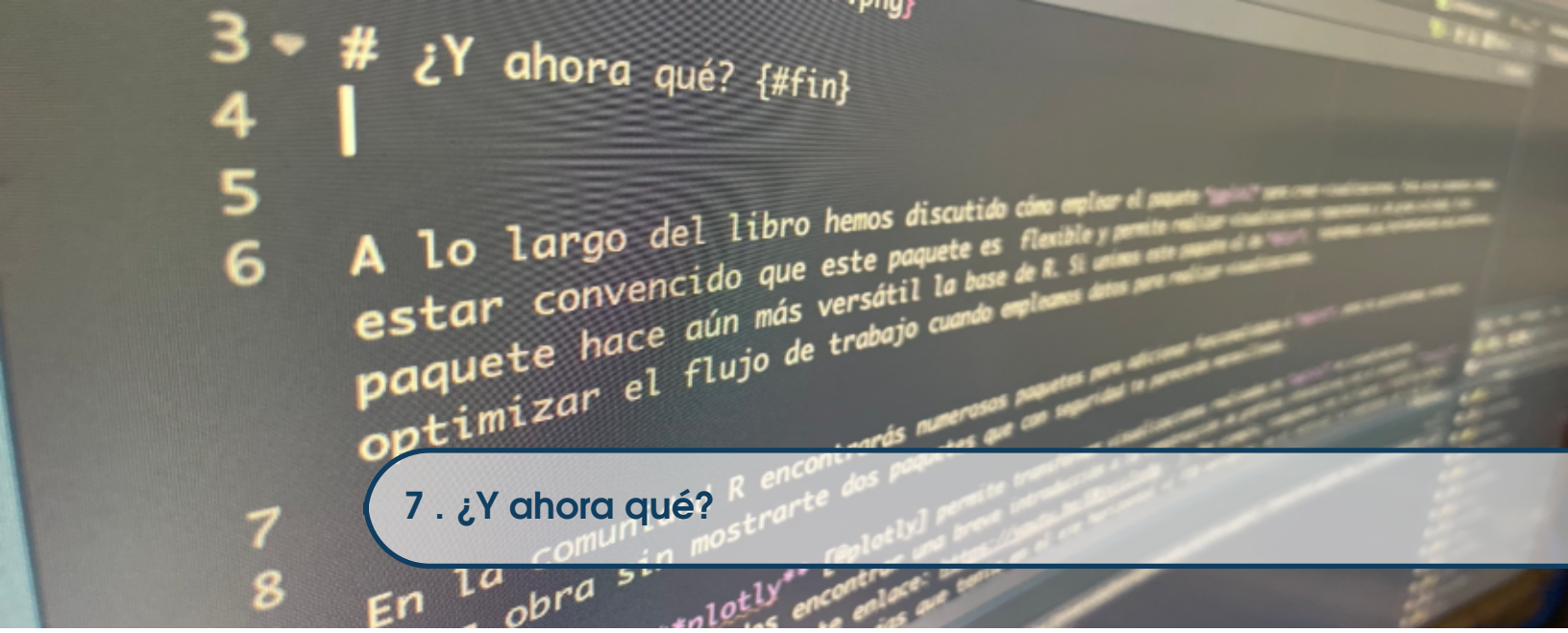
Fuente: Cálculos propios empleando datos simulados.

En resumen, no uses los gráficos de pastel y menos aquellos en 3D. Los gráficos de pastel en 3D resultan aún peor para la lectura de la información.

6.5 Comentarios finales

En este capítulo hemos visto cómo agregando unos pocos elementos podemos mejorar una visualización. Organizar los datos, emplear los colores para comunicar una idea y agregar texto, pueden ser herramientas muy potentes para mejorar tus visualizaciones. Y todo esto lo podemos hacer rápidamente con *ggplot2*. En el Capítulo 8 podrás aprender cómo mejorar tu visualización y persolizarla modificando la capa de ***Tema***.

Ya contamos con una caja de herramientas llena de potentes funciones que te permitirán generar visualizaciones. Pero tal vez la herramienta más importante al momento de construir una visualización, es la paciencia. Debemos intentar diferentes configuraciones de las capas hasta llegar a una visualización que transmita el mensaje que deseamos. Ensayar y errar hacen parte del proceso de construcción de una visualización.



A lo largo del libro hemos discutido cómo emplear el paquete *ggplot2* para crear visualizaciones. Para este momento, debes estar convencido que este paquete es flexible y permite realizar visualizaciones impactantes y de gran calidad. Este paquete hace aún más versátil la base de R. Si unimos este paquete al de *dplyr*, tendremos unas herramientas que permiten optimizar el flujo de trabajo cuando empleamos datos para realizar visualizaciones.

En la comunidad R encontrarás numerosos paquetes para adicionar funcionalidades a *ggplot*, pero no quisiéramos terminar esta obra sin mostrarte dos paquetes que, con seguridad, te parecerán maravillosos.

El paquete **plotly** (Sievert, 2020) permite transformar visualizaciones realizadas en *ggplot2* en visualizaciones interactivas[1]. Por ejemplo, trabajemos con la Figura 8.2. Esta era un gráfico de burbujas que tenía en el eje horizontal el PIB percápita, en el vertical la esperanza de vida al nacer, el tamaño de cada punto es la población y el color corresponde al continente. Recreemos ese gráfico y guardémoslo en un objeto.

```
# cargamos los paquetes
library(ggplot2)
library(dplyr)
library(gapminder)

# se guarda el gráfico en un objeto
g1 <- gapminder %>%
  filter(year == 2007) %>%
```

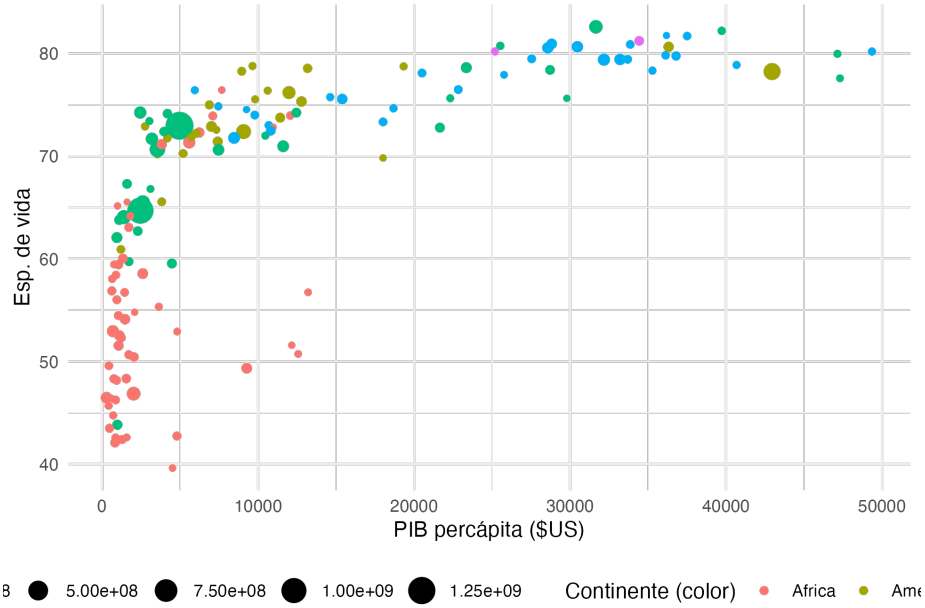
[1]Puedes encontrar una breve introducción a la construcción de gráficos interactivos con el paquete **plotly** (Sievert, 2020) en el siguiente enlace: https://youtu.be/EWjxic2ce9g .

```
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop, col = continent )) +
  geom_point() +
  labs(  x = "PIB percápita ($US)",
         y = "Esp. de vida",
         size = "Población (tamaño)",
         col = "Continente (color)") +
  theme_minimal() +
   theme(legend.position = "bottom")
```

Ahora podemos animar este gráfico empleando solo una línea de código. La función **ggplotly()** del paquete *plotly* (Sievert, 2020) solo requiere como argumento un objeto de clase **ggplot**. Es decir, la Figura 7.1 la podemos crear con la siguiente línea de código.

```
# instala el paquete si no o tienes
# install.packages("plotly")
# cargamos el paquete
library(plotly)
# convertimos a interactivo el gráfico
ggplotly(g1)
```

Figura 7.1. Gráfico interactivo de la relación del PIB percápita, la esperanza de vida y población alrededor del mundo (2007)

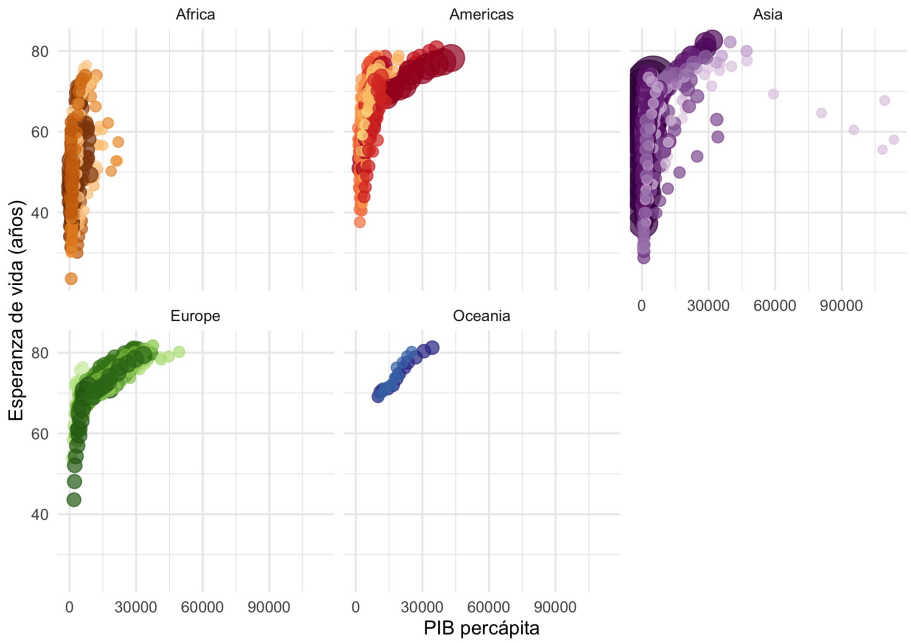


Fuente: Cálculos propios empleando datos simulados.

Este tipo de visualizaciones (ver Figura 7.1) permiten al usuario interactuar con los datos. Juega un rato con este gráfico. Se pueden apagar los continentes y hacer *zoom*. Pasa el cursor por encima de un punto para ver la información. La interacción solo funciona en la versión web del libro (http://www.icesi.edu.co/editorial/empezando-visualizar-2ed).

Otro paquete que puede llevar tus visualizaciones a otro nivel es *gganimate* (Pedersen y Robinson, 2020). Este paquete permite generar gráficos animados como el de la Figura 7.2. Esta gráfica no es interactiva como la Figura 7.1, pero sí permite ver la evolución en el tiempo de la misma relación entre la esperanza de vida como el PIB percápita. Aquí podemos ver cómo, por continente, han mejorado tanto la esperanza de vida como el PIB percápita. La animación solo funciona en la versión web del libro (http://www.icesi.edu.co/editorial/empezando-visualizar-2ed).

Figura 7.2. Visualización animada de la evolución del PIB percápita y la esperanza de vida alrededor del mundo (1952-2007).



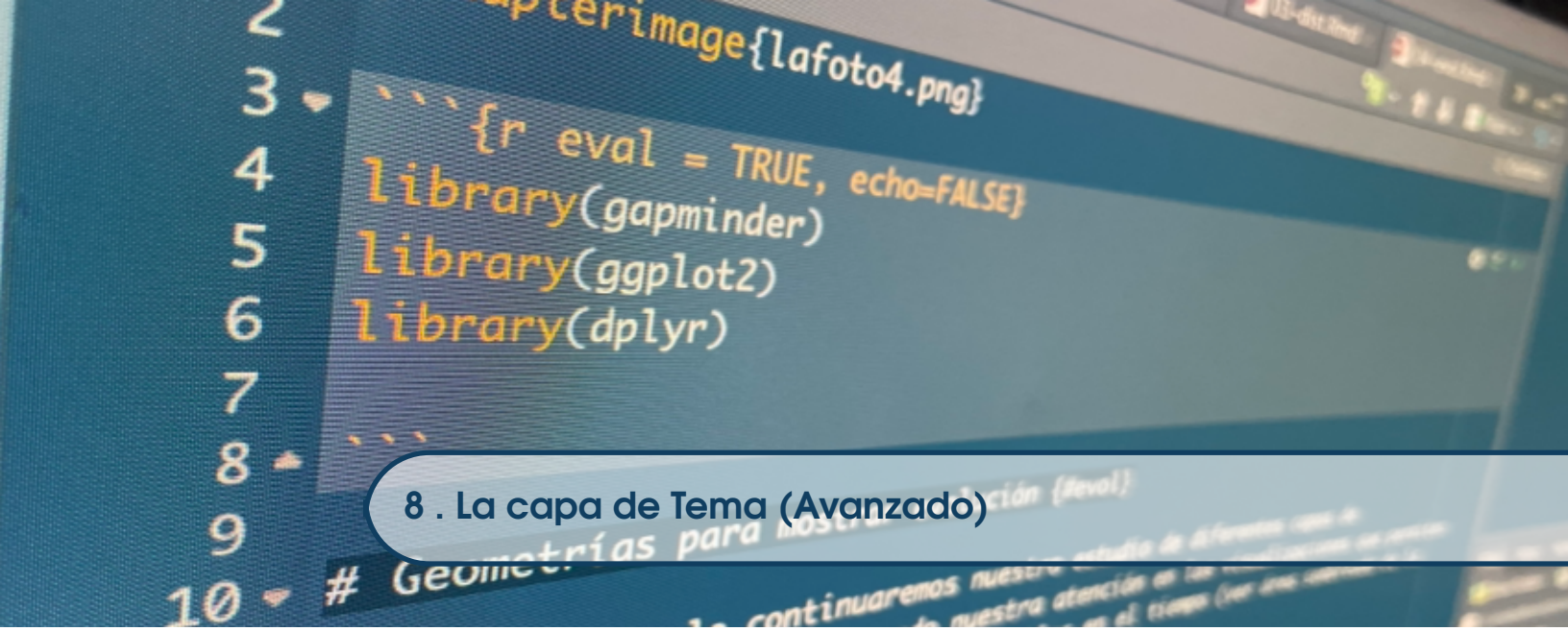
Fuente: Cálculos propios empleando datos simulados.

El código que produce la Figura 7.2 se presenta a continuación. No es tan complicado, pero debes seguirlo con cuidado. Nota que estamos empleando nuevas funciones de *ggplot2* como **facet_wrap()** y **scale_colour_manual()**. No obstante, estas funciones son nuevas para ti, con lo aprendido hasta ahora ya puedes saber cuáles son las capas que modifican. Y recuerda que siempre está disponible la ayuda y la documentación para aprender más sobre las nuevas funciones.

```
# instalar paquete si no se tiene
# install.packages('gganimate')

# cargar paquetes
library(gganimate)


g.anidado <- ggplot(gapminder, aes(x = gdpPercap, y= lifeExp,
                                   size = pop, colour = country)) +
  # capa de geometría
  geom_point(alpha = 0.7, show.legend = FALSE) +
  # modificar capa de Escalas
  scale_colour_manual(values = country_colors) +
  scale_size(range = c(2, 12)) +
  #modificar capa de Facets
  facet_wrap(~continent) +
    theme_minimal() +
  labs(title = 'Año: {frame_time}',
       x = 'PIB percápita',
       y = 'Esperanza de vida (años)') +
  # incluir elelmentos para la animación
  # corresponden al paquete gganimate
  transition_time(year) +
  ease_aes('linear')

# ver la animación
g.anidado
```

Estas visualizaciones interactivas y animadas son ejemplo de lo versátil que es R para generar visualizaciones de tus datos. Esperamos que esta obra te motive a continuar tu camino de aprendizaje y unirte a la gran comunidad de R. En este universo de R, ¡la imaginación es el límite!

En los siguientes capítulos presentamos temas mas avanzados relacionados con las visualizaciones creadas con *ggplot2*. En el Capítulo 8 podrás aprender cómo personalizar las visualizaciones empleando la capa de ***Tema***. Y si quieres seguir aprendiendo sobre geometrías más avanzadas para crear tus visualizaciones puedes ver el Capítulo 9.



8 . La capa de Tema (Avanzado)

En el Capítulo 2 discutimos cómo las visualizaciones que construimos en *ggplot2* siguen la gramática de los gráficos propuesta por Wilkinson (2012). Esta gramática implica construir las visualizaciones empleando capas (*layers* en inglés). Si bien existen 8 capas básicas en *ggplot2*, también vimos en el Capítulo 2 que no todas las capas son necesarias para construir una visualización con este paquete. Las tres capas esenciales son:

- ***Datos*** o data
- ***Aesthetics***
- ***Geometría***

Hasta aquí, hemos realizado un recorrido grande por diferentes clases de visualizaciones de datos empleando diferentes opciones de capas de ***Geometría*** y discutimos cómo la elección de dicha depende de la cantidad de variables, su clase y de lo que queremos comunicar. En el Capítulo 6 estudiamos cómo cambiarle la apariencia a nuestras visualizaciones para comunicar mejor el mensaje deseado.

En este capítulo, centraremos nuestra atención a la capa de ***Tema***. Esta capa nos permitirá personalizar la apariencia de nuestra visualización a la imagen institucional o al tipo de diseño de todo el documento. En esta capa se modifica el color del fondo, ejes, tamaños, grilla, posición de los nombres de los ejes, entre otros.

8.1 Temas integrados en *ggplot2*

Como se discutió en la Sección 2.1, el paquete *ggplot2* viene con una serie de temas integrados, que podemos emplear rápidamente para cambiar el aspecto de una visualización. Para entender mejor el uso

de esta capa, empleemos el gráfico de burbujas que trabajamos en la Sección 5.2. En esa visualización queríamos mostrar, empleando los datos del paquete *gapminder* (Bryan, 2017), la relación entre la esperanza de vida al nacer, el PIB percápita y la población para el año 2007. Recordemos la visualización:

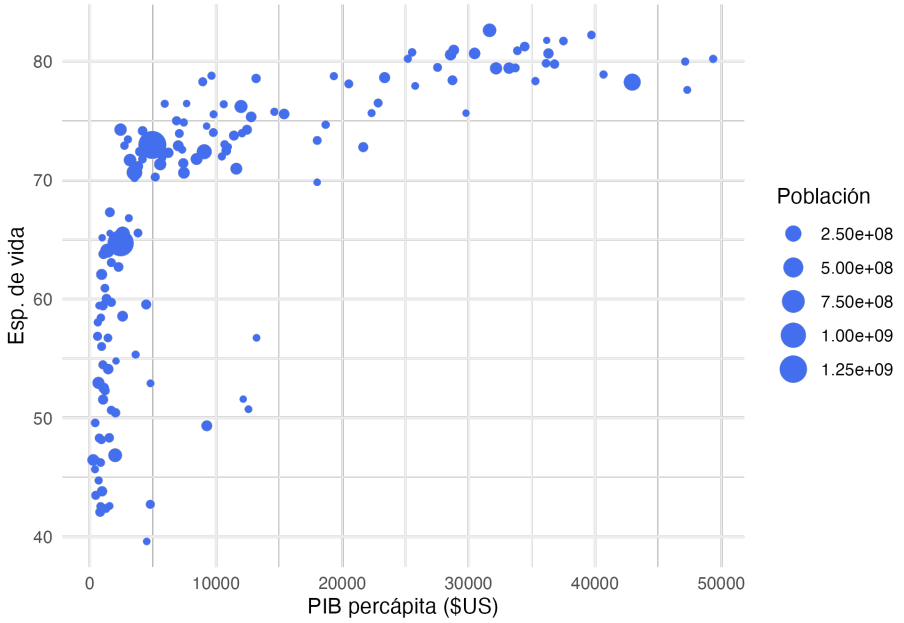
```
# cargar los paquetes
library(ggplot2)
library(dplyr)
library(gapminder)

# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         size="Población") +
  theme_minimal()
```

Noten que la Figura 8.1 está usando en la capa de ***Tema*** el tema minimalista (**theme_minimal()**), que se adicionó en la última capa. Este tema lo hemos estado usando durante todo el libro.

De manera similar a la capa de ***Geometría***, las diferentes opciones para la capa de ***Tema*** inician con el prefijo **theme_**. En el paquete encontraremos los siguientes temas:

- **theme_gray()**: genera un tema de fondo gris con líneas de cuadrícula blancas (grilla). Este es el tema por defecto; es decir, si no se incluye la capa de ***Tema*** en un gráfico construido con *ggplot2*, entonces se empleará este tema. En el panel A de la Figura 8.2 se presenta un ejemplo. El mismo tema se puede obtener empleando **theme_grey()** , dado que en inglés británico el color gris se escribe *grey* y en inglés de Estados Unidos *gray*.
- **theme_bw()**: fondo claro con líneas de grilla negras, con líneas en los ejes y un cuadrado que enmarca el área dónde se presentan los datos. Puede funcionar mejor para presentaciones mostradas con un proyector. En el panel B de la Figura 8.2 se presenta un ejemplo.
- **theme_linedraw()**: tema muy parecido al **theme_bw()** pero con líneas más delgadas, dando una apariencia un poco más minima-

Figura 8.1. PIB percápita, expectativa de vida al nacer y población por país (2007)



Fuente: Datos del paquete gapminder.

lista (ver panel C de la Figura 8.2).
- **theme_light()**: tema muy parecido al **theme_linedraw()** pero con líneas grices y no negras en los ejes y en la caja que enmaraca los datos. En teoría esto permite mayor atención del lector en los datos y no en los ejes (ver panel A de la Figura 8.3).
- **theme_dark()**: fondo gris oscuro con grilla smilar al del tema **theme_light()** (ver Figura 8.3, panel B).
- **theme_classic()**: tema considerado más "clásico", al no incluir fondo, ni grilla, ni caja que enmarque a los datos. Solo una línea negra en los ejes (ver Figura 8.3, panel C).
- **theme_minimal()**: como su nombre indica, tema con un diseño minimalista sin fondos grises, grilla en gris y sin línea en los ejes, ni caja alrededor de los datos. Es perfecto para gráficos que requieren una apariencia limpia y sin distracciones (ver Figura 8.4, panel A).
- **theme_void()**: tema que no tiene fondo, ni grilla, ni líneas de los ejes (ver Figura 8.4, panel B).
- **theme_test()** tema que solo incluye líneas en los ejes y en los lados para construir una caja para enmarcar los datos. Según los desarrolladores del paquete, este es un tema experimental que permite a los usuarios probar diferentes estilos y configuraciones (ver Figura 8.4, panel C).

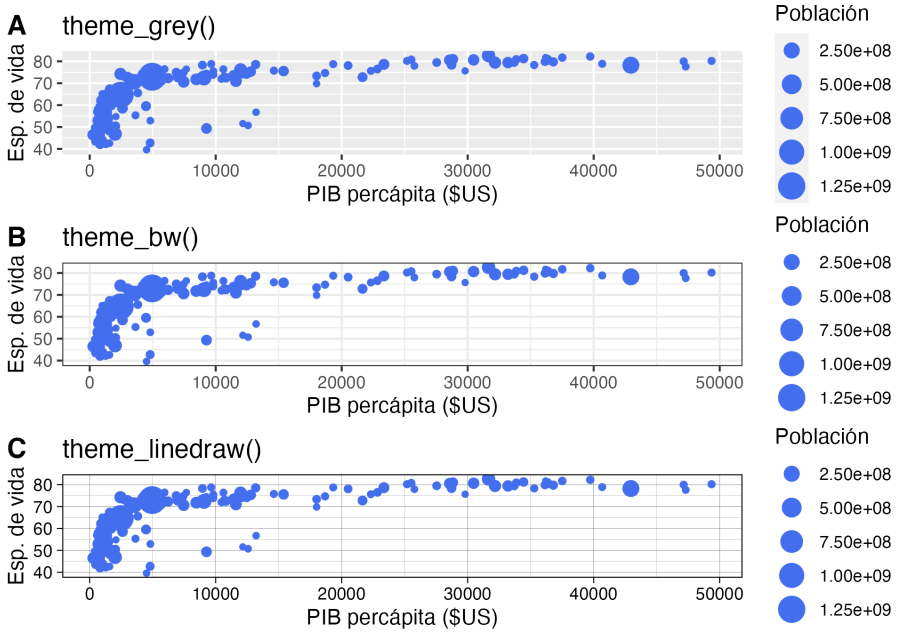
Los temas incluidos en el paquete *ggplot2* dan gran versatilidad para personalizar nuestras visualizaciones. La elección del tema para tu visualización depende de la naturaleza de tus datos, el contexto en el que se presentarán y tu gusto personal. Con estos temas, puedes crear visualizaciones que sean atractivas, claras y efectivas en la comunicación de tu mensaje. Recuerda que, al final del día, el objetivo principal es que tus visualizaciones transmitan los datos de manera fiel y sin distracciones.

8.2 Modificando los temas

Como se puede observar en las Figuras 8.2, 8.3 y 8.4 podemos observar las diferentes opciones que tenemos para la capa de ***Tema*** en el paquete *ggplot2*. Pero la versatilidad de este paquete para crear visualizaciones no para ahí. Estos temas preestablecidos pueden ser modificados a nuestro gusto para lograr la visualización deseada.

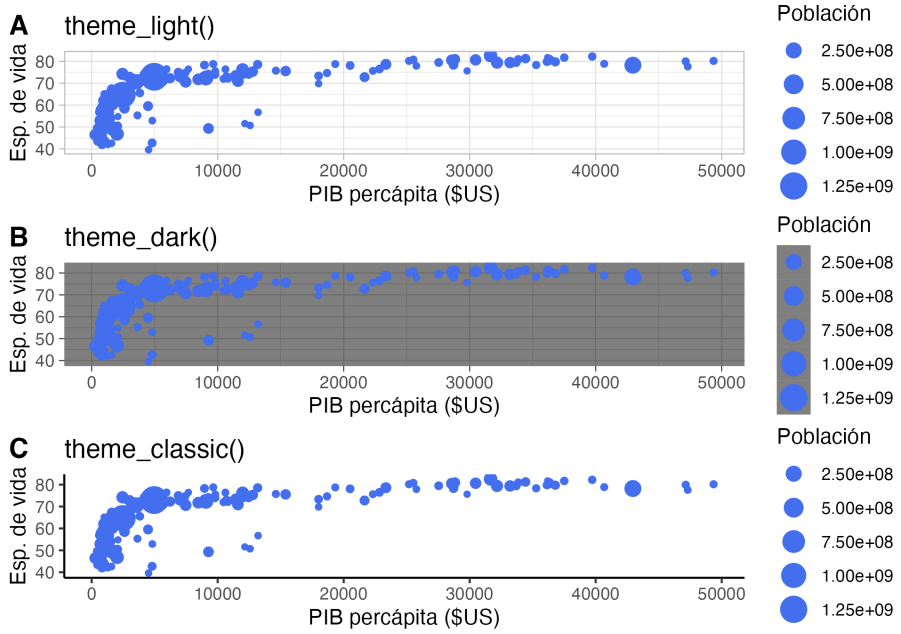
Por ejemplo, la posición de la leyenda para la variable `Población` en las Figuras 8.2, 8.3 y 8.4 es al lado del gráfico. En algunas ocasiones quisiéramos mover esa leyenda a la parte inferior para dar mas espacio a los datos. O por ejemplo, quisiéramos cambiar el fondo gris, el color de las líneas de la grilla o su grosor, o cambiar el tamaño o tipo de letra de los nombres de los ejes o de los números de las marcas de los ejes. Todo esto lo podemos hacer modificando la capa de ***Tema***.

Figura 8.2. Visualización con los gray, bw() y linedraw del paquete ggplot2



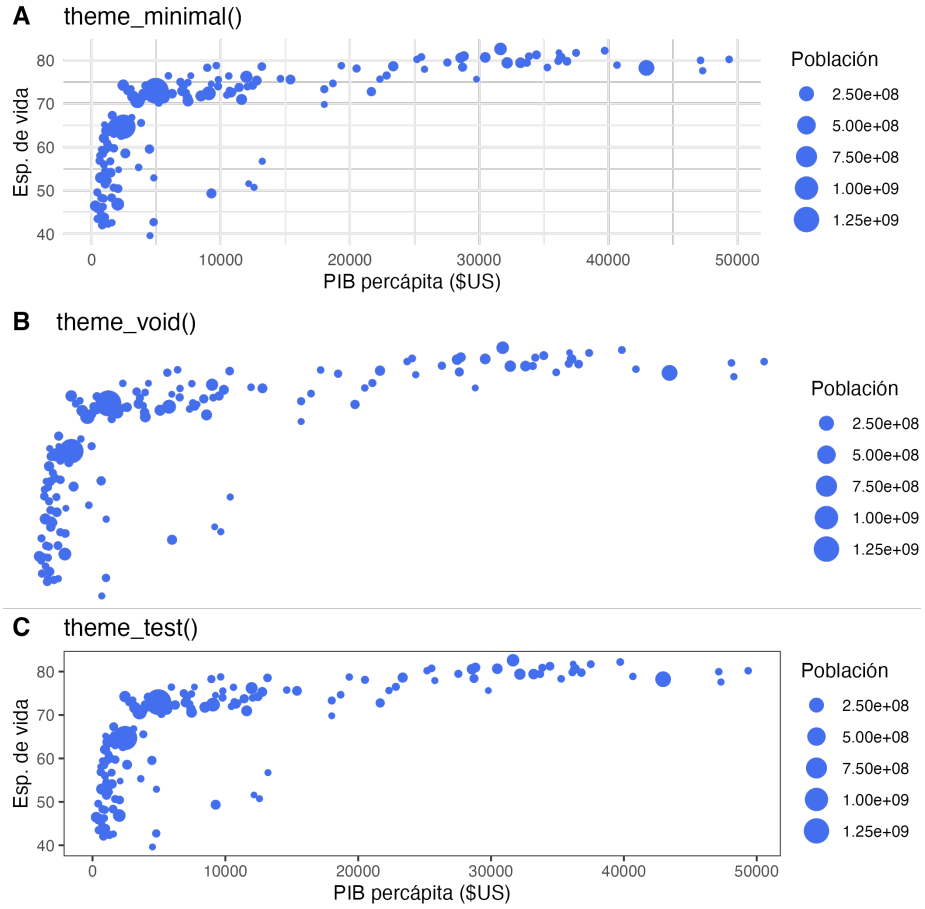
Fuente: Datos del paquete gapminder.

Figura 8.3. Visualización con los temas light(), dark() y classic del paquete ggplot2



Fuente: Datos del paquete gapminder.

Figura 8.4. Visualización con los temas minimal(), void() y test del paquete ggplot2



Fuente: Datos del paquete gapminder.

La función **theme()** del paquete *ggplot2* (Wickham et al., 2023a) permite modificar los diferentes elementos de la capa de ***Tema*** que se esté empleando. Por ejemplo, con el argumento **legend.position** podemos modificar la posición de la leyenda. Regresemos a la Figura 8.1. Ésta emplea el tema minimalista que pone por defecto la leyenda en la parte izquierda del gráfico. Con el siguiente código podemos modificar el **theme_minimal** para ubicar la leyenda en la parte inferior.

```
# se emplea el operador pipe para
# pasar los datos y filtrar los
# datos

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         size="Población") +
  theme_minimal() +
  # modificar el tema cambiando la posición de la leyenda
  theme(legend.position = "bottom")
```

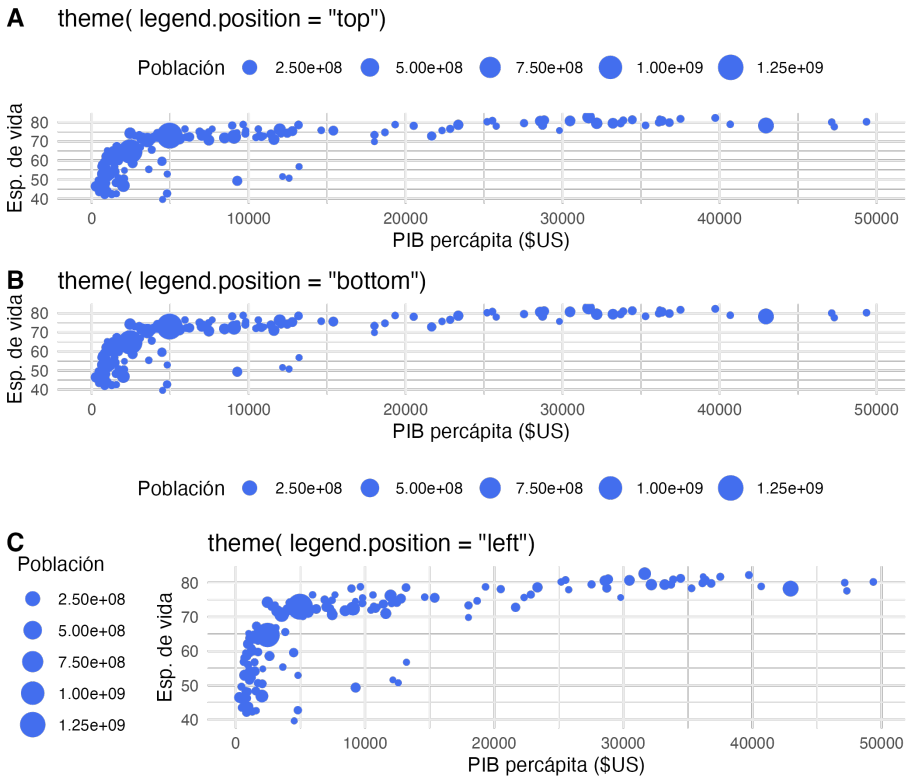
El código anterior generará una visualización como la del panel A de la Figura 8.5. En la Figura 8.5 se presentan todas las opciones de posición de la leyenda diferentes a la por defecto del tema minimal.

Es importante tener en cuenta que la función **theme()** debe usarse después de cargar la capa de ***Tema***, dado que esta función modifica el tema que esté cargado. Si posteriormente a usar la función tema se carga el tema minimal, entonces no se modificará el tema minimal. ¡Inténtalo cambiando el orden y verás los resultados!

Son muchos los elementos del tema de una visualización y cada uno de ellos se puede modificar con la función **theme()**, empleando el respectivo argumento. La mayoría de los elementos los podemos clasificar en:

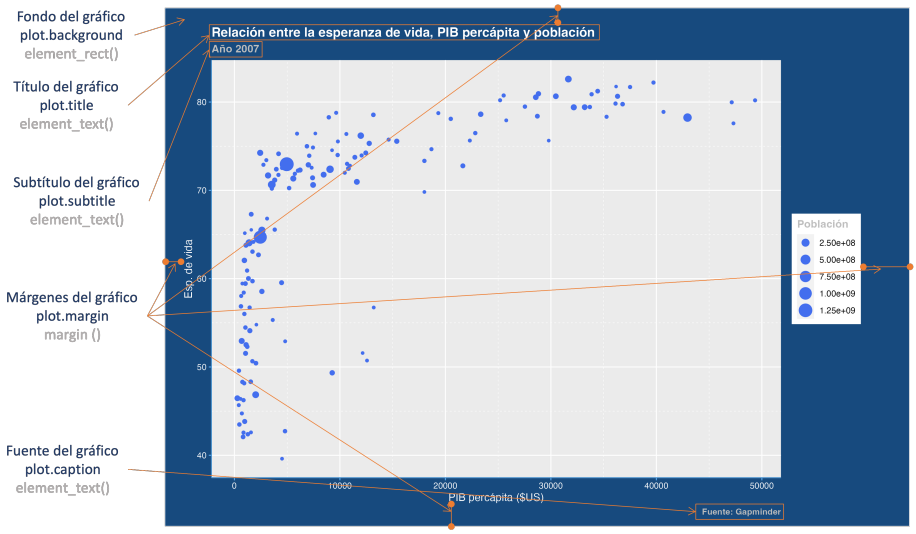
- Elementos que modifican el gráfico (**plot elements**): Entre estos elementos se encuentran el fondo del gráfico (**plot.background**), el título (**plot.title**), el subtitulo (**plot.subtitle**), la fuente (**plot.caption**) y las márgenes (**plot.margin**). (Ver Figura 8.6)
- Elementos que modifican los ejes (*axis elements*): Estos elementos incluyen el título de los dos ejes (**axis.title**[1]), marcas del eje (**axis.ticks**) y líneas de los ejes (**axis.line**). (ver Figura 8.7)

Figura 8.5. Visualización con el tema minimal y diferentes posiciones de la leyenda



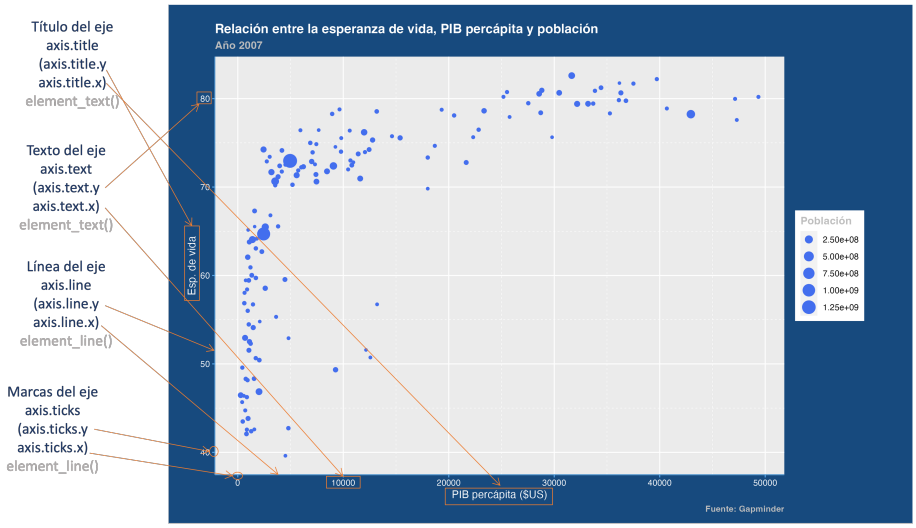
Fuente: Datos del paquete gapminder.

Figura 8.6. Elementos de un tema en ggplot2: Elementos del gráfico (plot elements)



Fuente: Creado a partir de Wickham (2016).

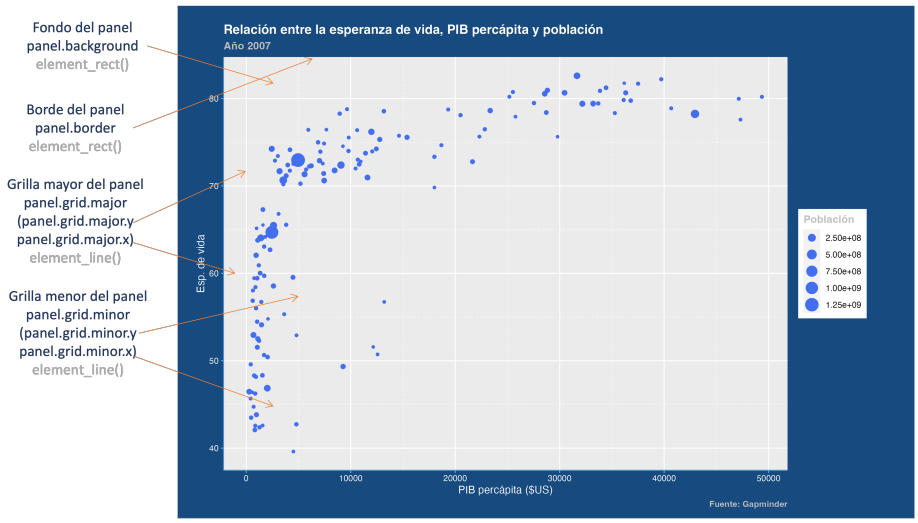
Figura 8.7. Elementos de un tema en ggplot2: Elementos de los ejes (axis elements)



Fuente: Creado a partir de Wickham (2016).

- Elementos que modifican el panel (*panel elements*): estos incluyen el fondo (**panel.background**), el borde (**panel.border**), grilla mayor (**panel.grid.major**) y grilla menor (**panel.grid.minor**). (ver Figura 8.8)

Figura 8.8. Elementos de un tema en ggplot2: Elementos del panel (panel elements)



Fuente: Creado a partir de Wickham (2016).

- Elementos que modifican la leyenda (*legend elements*): incluyen las márgenes (**legend.margin**), el título (**legend.title**), la clave (**legend.key**), el texto (**legend.text**) y el fonfo (**legend.backgroud**). (ver Figura 8.9)

Los elementos del tema (ver Figuras 8.6, 8.7, 8.8 y 8.9) que se desean modificar son argumentos de la función **theme()**. Estos elementos tienen la estructura (en inglés):

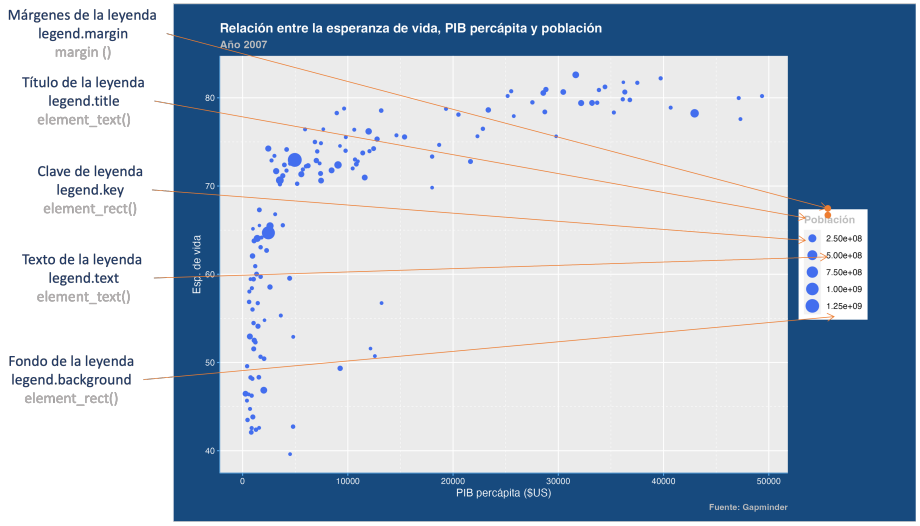
elemento.nombre

Por ejemplo, para modificar los títulos (el nombre será *title*) de los ejes (el elemento sera *axis*) el argumento será **axis.title**.

Para modificar cada uno de los elementos se pueden emplear diferentes funciones de acuerdo con el elemento a cambiar. Estas funciones están relacionadas con modificar: textos (**element_text()**), líneas (**element_line()**), márgenes (**margin()**) y áreas rectangulares (**element_rect()**). También es posible emplear la función **element_blank()** para

[1] El texto del eje horizontal corresponde en la terminología del paquete **axis.title.x** y el del eje vertical es **axis.title.y**. Esa misma lógica aplicará para cuando se desea modificar un elemento de un solo eje.

Figura 8.9. Elementos de un tema en ggplot2: Elementos de la leyenda (legend elements)



Fuente: Creado a partir de Wickham (2016).

eliminar del tema un elemento; esta función no necesita argumentos. Cada una de estas funciones que permiten modificar los elementos del tema tiene argumentos diferentes, por su naturaleza.

Veamos en detalle cada uno de estas funciones. La función **element_text()** que modifica elementos que implican letras o números tiene como argumentos principales[2]:

element_text(face, colour, size, angle)

donde:

- **face**: tipo de letra. Es decir, itálica ("italic"), negrita ("bold") y negrita e itálica ("bold.italic").
- **colour**: color de la letra.
- **size**: tamaño de la letra en puntos.
- **angle**: ángulo de la letra que debe ser un número entre 0 y 360.

Por ejemplo, para modificar el tema para que el color de los números que están en los ejes (**axis.text**) cambie a blanco se podrá hacer agregando la siguiente línea de código a las que construyen la visualización:

[2]Mira en la ayuda de esta función los otros argumentos de esta función.

```
+ theme(axis.text = element_text(colour = "white"))
```

Por otro lado, la función **element_line()** que modifica los elementos asociados a líneas tiene como argumentos principales:

element_line(colour, linewidth, linetype)

donde:

- **colour**: color de la línea.
- **linewidth**: tamaño de la línea en milímetros.
- **linetype**: el tipo de línea(por ejemplo solida punteada con interrupciones. El tipo de línea será un número entero entre 0 y 8. ¡Prueba cada una de las opciones!

Como ejemplo, modifiquemos la líneas menores de la grilla del panel (elemento **panel.grid.minor**) dibujándolas en blanco y con una línea punteada. Eso lo podemos hacer sumando la siguiente línea de código a nuestro gráfico:

```
+ theme(panel.grid.minor = element_line(color = "white",
                                         linetype = 2))
```

De manera similar, la función **element_rect()**, que modifica los elementos relacionados con áreas, tiene como argumentos principales:

element_rect(fill, colour, linewidth, linetype)

donde:

- **fill**: color para rellenar el área.
- **colour**: color de la línea del borde.
- **linewidth**: tamaño de la línea en milímetros.
- **linetype**: el tipo de línea (por ejemplo, sólida punteada con interrupciones. El tipo de línea será un número entero entre 0 y 8.

Como ejemplo, cambiemos el fondo de la gráfica (elemento **plot.background**) al color azul (`BLUE1` que definimos en el Capítulo 6). Eso lo podemos realizar sumando la siguiente línea de código par modificar el tema:

```
+ theme(plot.background = element_rect(fill = BLUE1))
```

Finalmente, la función **margin()**, que modifica los márgenes tiene los siguientes argumentos:

margin(t, r, b, l, unit)

donde:

- **t**: el margen superior (*top*). Este argumento deberá ser un número. Por defecto, el margen superior es cero.
- **r**: el margen derecho (*right*). Este argumento deberá ser un número. Por defecto, el margen derecho es cero.
- **b**: el margen inferior (*bottom*). Por defecto, el margen inferior también es cero.
- **l**: el margen izquierdo (*left*). Por defecto, el margen izquierdo también es cero.
- **unit**: las unidades en las que se expresan los márgenes. Por defecto las unidades son puntos ("pt").

Para mostrar un ejemplo, cambiemos las márgenes del gráfico (elemento **plot.margin**) sumando la siguiente línea de código al código que teníamos hasta ahora para modificar los márgenes del gráfico:

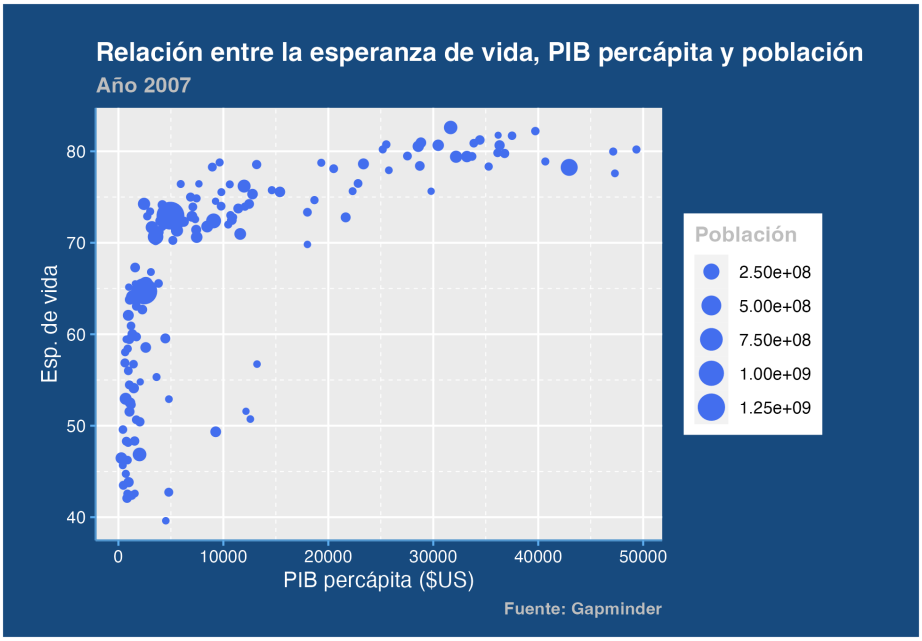
```
+ theme( plot.margin =  margin(t = 20,  # Margen superior
                               r = 50,  # Margen derecho
                               b = 10,  # Margen inferior
                               l = 20), # Margen izquierdo)
```

Es importante anotar que se pueden cambiar diferentes elementos al mismo tiempo empleando comas (,) para separar los elementos que van al interior de la función **theme()**. Un ejemplo de modificar el tema **theme_gray()** (que es el tema por defecto) se puede observar en la la Figura 8.10.

El siguiente código genera la Figura 8.10.

```
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         size="Población",
         title = "Relación entre la esperanza de vida,
                      PIB percápita y población",
         subtitle = "Año 2007",
         caption = "Fuente: Gapminder") +
  theme(plot.background = element_rect(fill = BLUE1),
        plot.title = element_text(face = "bold",
                                  colour = "white"),
        plot.subtitle = element_text(face = "bold",
                                     colour = "grey"),
        plot.caption = element_text(face = "bold",
```

Figura 8.10. PIB percápita, expectativa de vida al nacer y población por país (2007) (con tema modificado)



Fuente: Datos del paquete gapminder.

```
                                colour = "grey"),
      plot.margin =  margin(t = 20,  # Margen superior
                            r = 50,  # Margen derecho
                            b = 10,  # Margen inferior
                            l = 20), # Margen izquierdo

      legend.title = element_text(face = "bold",
                                  colour = "grey"),

      axis.line = element_line(colour = "steelblue3",
                               linewidth = 0.5),
      axis.ticks = element_line(colour = "steelblue2"),
      axis.text = element_text(colour = "white"),
      axis.title = element_text(colour = "white"),

      panel.grid.minor = element_line(color = "white",
                                      linetype = 2)
    )
```

Este código modificó diferentes elementos. En la visualización se modificaron los siguientes elementos relacionados con el gráfico:

- el fondo cambiándolo a `BLUE1`,
- el título, subtítulo y título cambiando su color y el tipo de letra a negrita
- el margen

También se modificó un elemento relacionado con la leyenda: el color del título. Con respecto los ejes, se modificaron:

- el color de las líneas a `steelblue3`
- el color de las marcas a `steelblue2`
- el color de los números y títulos a blanco

Finalmente, se modificó la grilla menor del panel cambiando su color a blanco y el tipo de línea a punteada.

Como puedes observar, modificar un tema no es complicado, pero puede ser un poco tedioso. Una alternativa para realizar las modificaciones al tema empleando una interfaz gráfica (con el ratón apuntando y haciendo click), es el paquete *ggThemeAssist*[3] (Gross y Ottolinger, 2016).

[3]En el siguiente enlace puedes encontrar una guía de cómo usar esta interfaz gráfica: https://github.com/calligross/ggthemeassist.

8.3 Fijando un tema por defecto y creando un tema

En algunas ocasiones será deseable cambiar el tema por defecto de *ggplot2*, de tal manera que no necesitemos establecer la capa de ***Tema*** cada vez que hacemos una visualización en nuestra sesión de R. Por ejemplo, en la mayoría el libro hemos adicionado la capa de ***Tema*** minimal (**theme_minimal()**) y esto puede ser tedioso. Puede ser mejor definir la capa de ***Tema*** desde el principio a minimal y no tener que adicionarla a cada visualización que realicemos.

Para lograr esto, podemos emplear la función **theme_set()** y como argumento el tema que queremos establecer para el resto de nuestra sesión de R[4]. Por ejemplo, si queremos establecer el tema `bw` como el preestablecido, podemos emplear la siguiente línea de código.

```
# cambiar el tema por defecto
theme_set(theme_bw())
```

Por ejemplo, observa lo que ocurre si corres el siguiente código:

```
gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         size="Población")
```

Además, podemos realizar cambios en los elementos del tema preestablecido empleando la función **theme_update()** y como argumento el elemento que se desea modificar y la función correspondiente. Por ejemplo, siguiendo nuestro ejemplo anterior, modifiquemos el tema preestablecido para que la leyenda esté en la parte inferior.

```
 theme_update(legend.position = "bottom")

gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop)) +
  geom_point(col = "royalblue2") +
  labs(  x="PIB percápita ($US)",
```

[4]una vez cierres R o cargues nuevamente el paquete *ggplot2* se restablecerá el tema por defecto de dicho paquete.

```
        y="Esp. de vida",
        size="Población")
```

Para conocer todos los detalles del tema por defecto que tenemos establecido en nuestra sección de R podemos emplear la función **theme_get()**.

También podemos crear nuestro propio tema personalizado y establecerlo como el tema global. Para crear un tema personalizado, podemos crear una función que contenga la capa de ***Tema***. Esta nueva función deberá emplear la función **theme()** para modificar un tema ya existente o para crear un tema desde cero.

Por ejemplo, la siguiente función crea el tema **theme_azul()** que tiene fondo azul, lineas de grilla principal azules, no tiene grilla menor, y las letras de los ejes y leyenda se cambian al color azul y los títulos de los ejes y de la leyenda se cambian a azul con negrilla y letra *Times New Roman*.

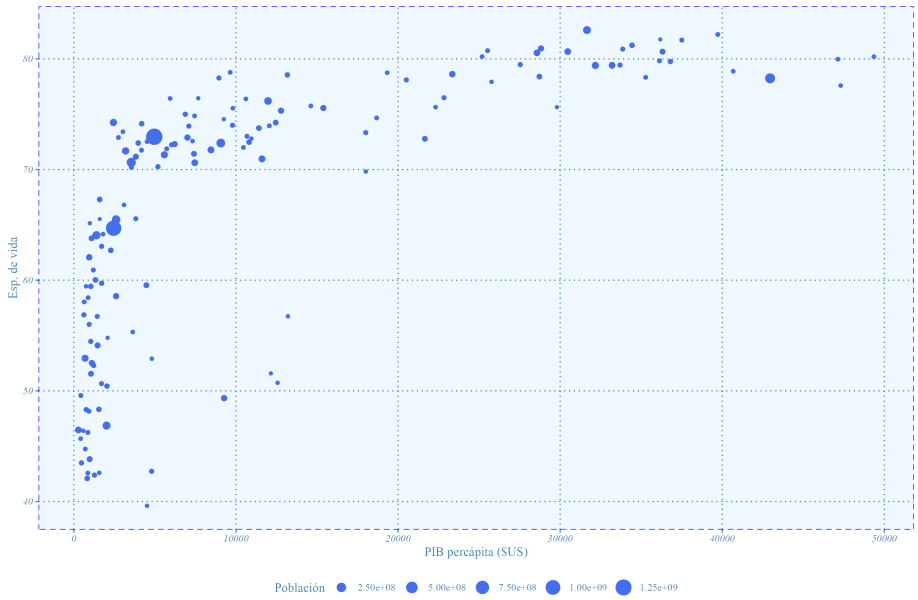
```
theme_azul <- function() {
   theme(
     # agregar bordes
     panel.border = element_rect(colour = "blue", linetype = 2),
     # agregar colo al fondo
     panel.background = element_rect(fill = "aliceblue"),
     # modificar la grilla
     panel.grid.major.x = element_line(colour = "steelblue",
                                       linetype = 3,
                                       linewidth = 0.5),
     panel.grid.minor.x = element_blank(),
     panel.grid.major.y =  element_line(colour = "steelblue",
                                        linetype = 3,
                                       linewidth = 0.5),
     panel.grid.minor.y = element_blank(),
     # modificar los elementos de loe ejes
     axis.text = element_text(colour = "steelblue",
                              face = "italic",
                              family = "Times New Roman"),
     axis.title = element_text(colour = "steelblue",
                               family = "Times New Roman"),
     axis.ticks = element_line(colour = "steelblue"),
     # ubicar la leyenda en la parte inferior
     # y cambiar el color del texto
     legend.position = "bottom",
     legend.title = element_text(colour = "steelblue",
                                 family = "Times New Roman"),
```

```
      legend.text = element_text(colour = "steelblue",
                                 family = "Times New Roman")
    )
}
```

Apliquemos esta función, como lo hemos hecho hasta aquí, para obtener una visualización como la presentada en la Figura 8.11.

```
gapminder %>%
      filter(year == 2007) %>%
      ggplot(aes(x = gdpPercap, y = lifeExp,
                 size = pop)) +
      geom_point(col = "royalblue2") +
      labs(  x="PIB percápita ($US)",
             y="Esp. de vida",
             size="Población") + theme_azul()
```

Figura 8.11. Visualización con tema propio



Fuente: Datos del paquete gapminder.

Ahora, si lo deseas, puedes establecer tu propio tema como el tema por defecto (tema global).

8.4 Temas de otros paquetes

En la sección anterior vimos cómo se puede personalizar un tema, esto podría tomar mucho tiempo o necesitar habilidades de diseñador. Afortunadamente, existen varios paquetes que incluyen otros temas. En esta sección estudiaremos 2 paquetes que nos pueden ayudar a personalizar nuestras visualizaciones.

8.4.1 Paquete ggthemes

El paquete *ggthemes* (Arnold, 2021) incluye temas inspirados en varias fuentes, como *Google Docs*, el periódico *Wall Street Journal* (WSJ) y la revista *The Economist*, entre otros. Adicionalmente, el paquete provee escalas de colores que hacen juego con el estilo para que tus visualizaciones pueden tener una apariencia familiar para algunas audiencias, y pueden ser especialmente útiles si estás buscando replicar estilos específicos.

Para mostrar este paquete empleemos una visualización de los datos disponibles en el paquete *gapminder* (Bryan, 2017), de la relación entre la esperanza de vida al nacer y el PIB percápita para el 2007 empleando un color diferente para el continente al que pertenece cada país.

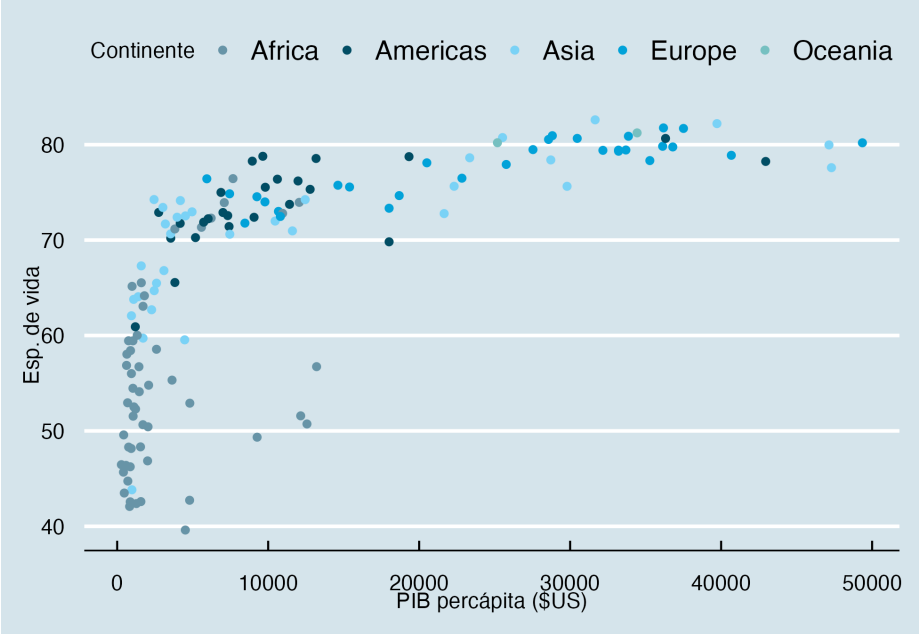
```
p <-  gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             color = continent)) +
  geom_point() +
  labs(  x="PIB percápita ($US)",
         y="Esp. de vida",
         color="Continente")
p
```

Ahora, para emplear los temas del paquete *ggthemes* solo debemos agregar el tema deseado y para mejorar aún mas la visualización podemos agregar una paleta de color que haga juego con el diseño del tema. Las paletas de colores modifican la capa de ***Escalas***. Por ejemplo, para construir una visualización empleando el tema de la revista *The Economist* podemos emplear la función **theme_economist()** y una escala de color[5] (también es una función) **scale_colour_economist()** . Es muy sencillo aplicar estas dos capas, el siguiente código crea la Figura 8.12.

[5]La escala de color le pasa los colores a la geometría de puntos y evita que se usen los colores por defecto de *ggplot2*.

```
# si no tienes instalado el paquete instálalo
# install.packages("ggthemes")
# cargamos el paquete
library(ggthemes)

p + theme_economist() + scale_colour_economist()
```

Figura 8.12. Visualización con el tema The Economist del paquete ggthemes



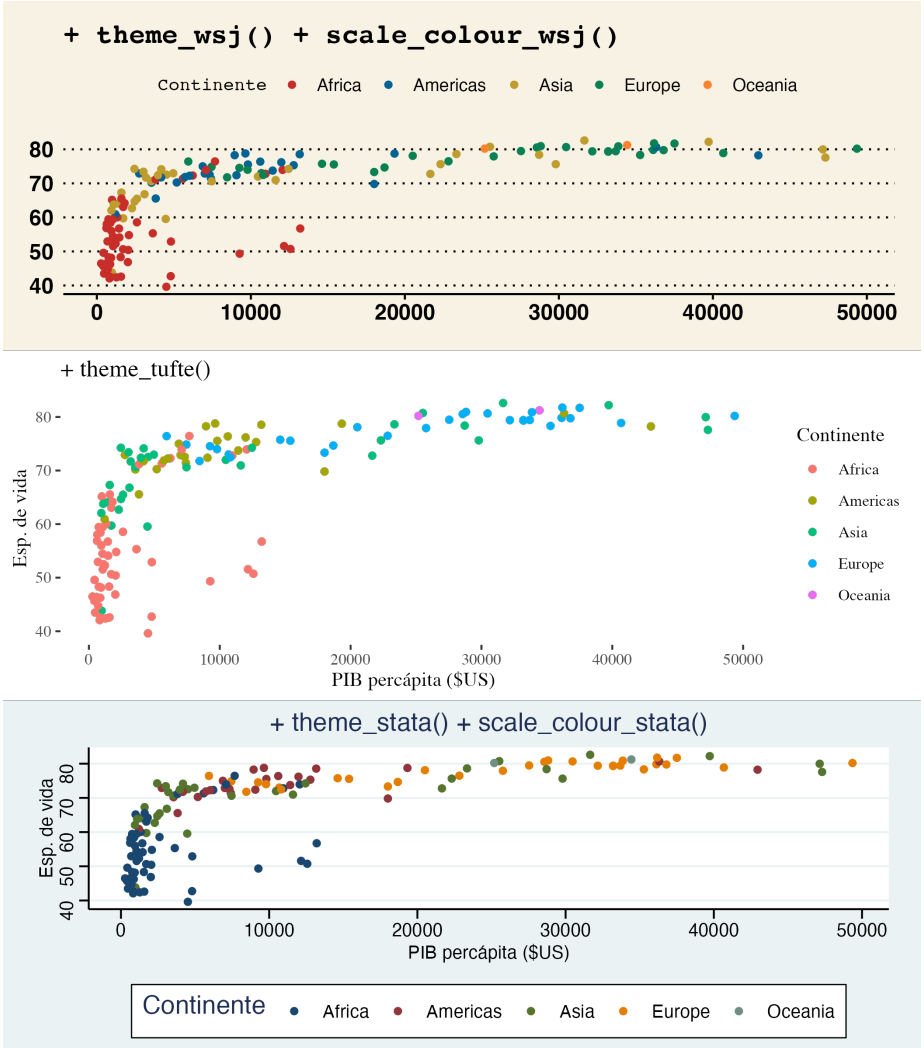
Fuente: Datos del paquete gapminder.

En las Figuras 8.13 y 8.14 se presentan otras 6 opciones de temas presentes en el paquete *ggtheme*. Este paquete ofrece más temas de paletas de colores, lo que da una gran cantidad de opciones de personalización de tus visualizaciones. Adicionalmente, siempre podremos modificar los elementos del tema como lo vimos en la sección anterior. Experimenta con estas opciones para encontrar la combinación que mejor se adapte a tus necesidades.

8.4.2 Paquete hrbrthemes

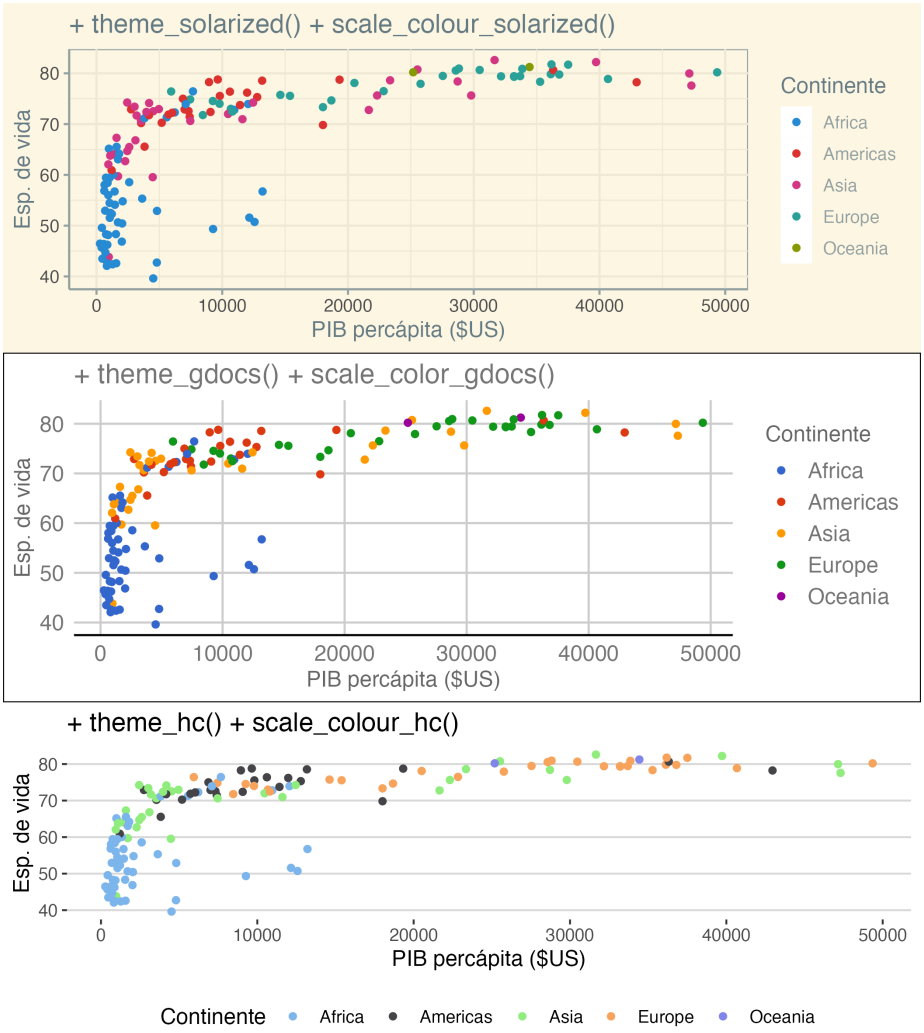
El paquete *hrbrthemes* (Rudis, 2020) también ofrece tres familias de temas interesantes para *ggplot2*:

Figura 8.13. Visualización con los temas WSJ, Tufte y Stata del paquete ggthemes



Fuente: Datos del paquete gapminder.

Figura 8.14. Visualización con los temas Solarized, Gdocs y Hc del paquete ggthemes



Fuente: Datos del paquete gapminder.

- Familia de temas *ipsum*: tema elegante y minimalista. Los temas de esta familia se diferencias por el tipo de letra que emplean. Puedes ver la ayuda para ver las diferentes opciones disponibles.
- Familia modo oscuro (*modern*): tema con fondo oscuro
- *tinyhand*: tema poco convencional ideal para visualizaciones poco formales.

El tema **theme_ipsum()** no requiere instalar ninguna letra especial en tu computador, mientras que los otros dos temas sí requieren la instalación de los tipos de letra `Roboto Condensed`[6] y `Timy Hand`[7]. respectivamente. Tras cargar el paquete (y si se tiene el tipo de letra correcto instalado), estos tema funcionan como hemos estudiado hasta ahora. En la Figura 8.15 se pueden ver ejemplos del uso de estos tres temas.

8.5 Comentarios finales

La capa de ***Tema*** es una forma poderosa de personalizar los componentes de una visualización creada en *ggplot2*. Ya sea que empleemos los temas integrados en *ggplot2* o los que vienen en otros paquetes, como *ggthemes* o *hrbrthemes*, siempre tendremos la opción de modificar los elementos de la capa de ***Tema*** para personalizar aún más nuestra visualización.

Constantemente son publicados paquetes oficiales y no oficiales de R que incluyen nuevos temas. Por ejemplo, podemos encontrar el paquete *ggpomological* (Aden-Buie, 2023) que tiene un tema que genera visualizaciones no tan comunes como se observa en la Figura 8.16.

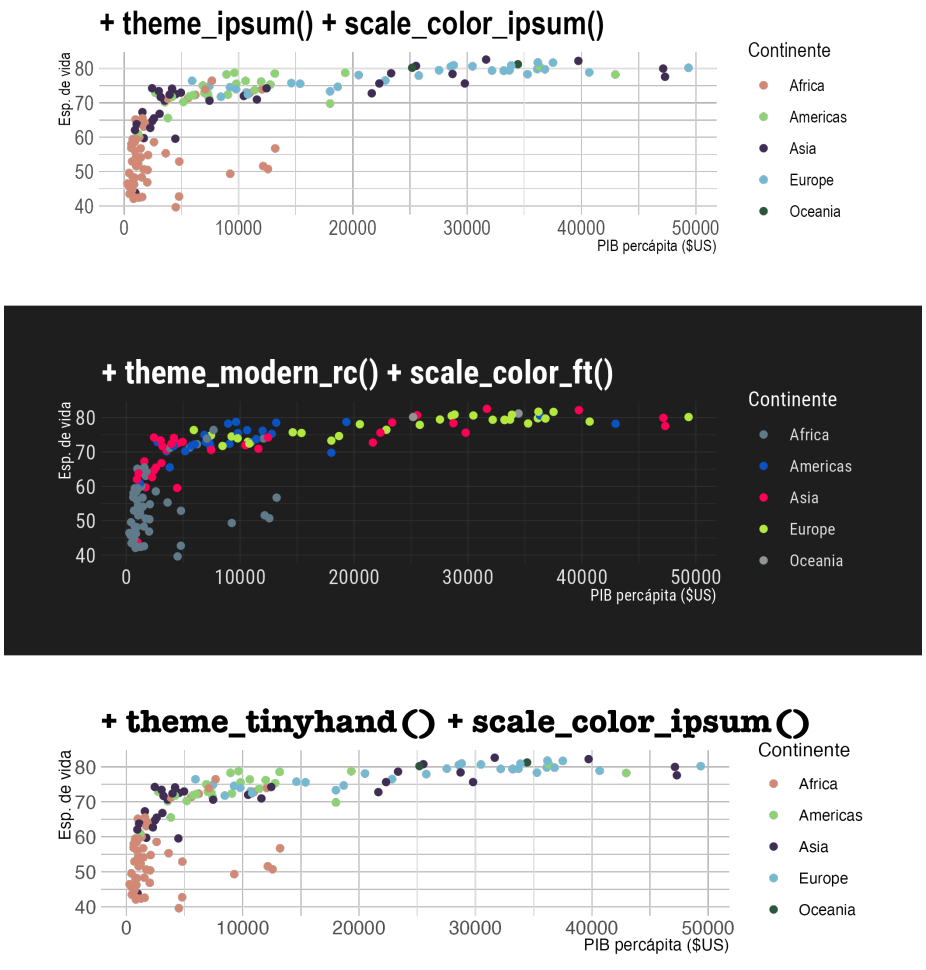
Independientemente del paquete que empleamos, la capa de ***Tema*** nos permitirá personalizar nuestra visualización. Te invitamos a que juegues con la capa de ***Tema*** hasta que encuentres un diseño de esta capa que sirva para tu objetivo. Por ejemplo, puedes llegar a personalizar tu visualización con tu correo y el logo de tu empresa, como en la Figura 8.17. ¡La imaginación es el límite!

Esperamos este capítulo te haya brindado una introducción a los temas en *ggplot2*. Recuerda que la comunidad de usuarios de R es muy grande. Si encuentras un problema o te atascas en alguna parte de tu código, con seguridad encontrarás ayuda en los foros de usuarios en la web.

[6] Esta letra se puede descargar del siguiente enlace: https://fonts.google.com/specimen/Roboto+Condensed.

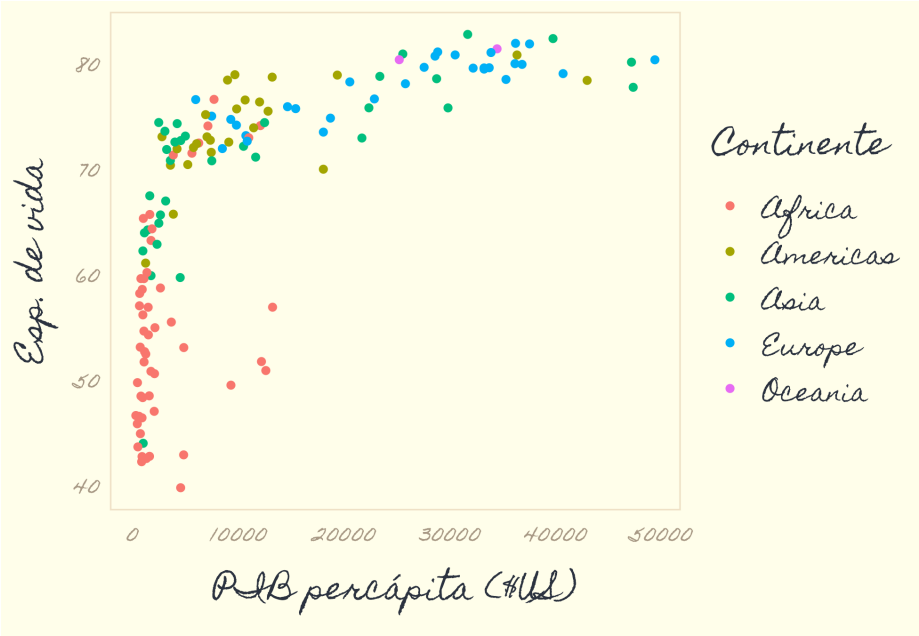
[7] Esta letra se puede descargar del siguiente enlace: https://fontsarena.com/bf-tiny-hand-by-buzzfeed-news/#google_vignette.

Figura 8.15. Visualización con los temas ipsum, modern y TinyHand del paquete hrbrthemes



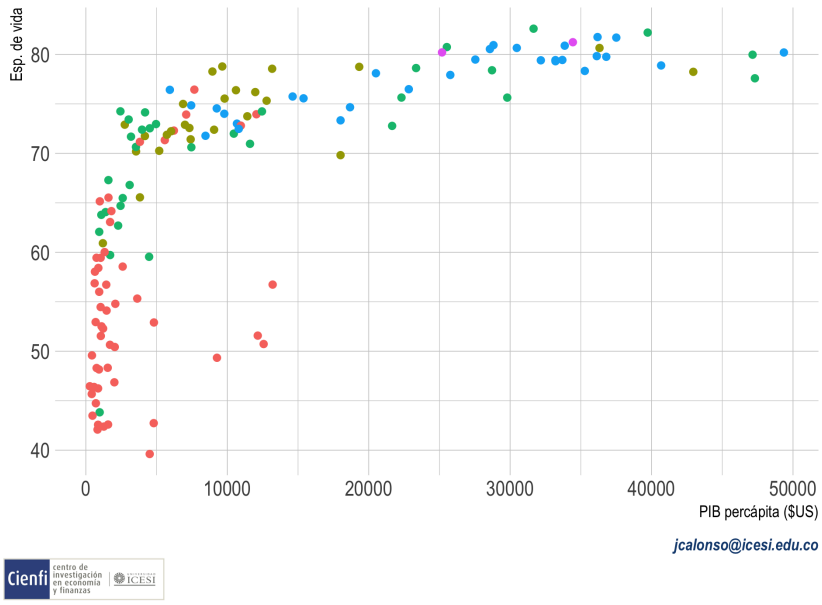
Fuente: Datos del paquete gapminder.

Figura 8.16. Visualización creada con la capa de Tema del paquete ggpomological

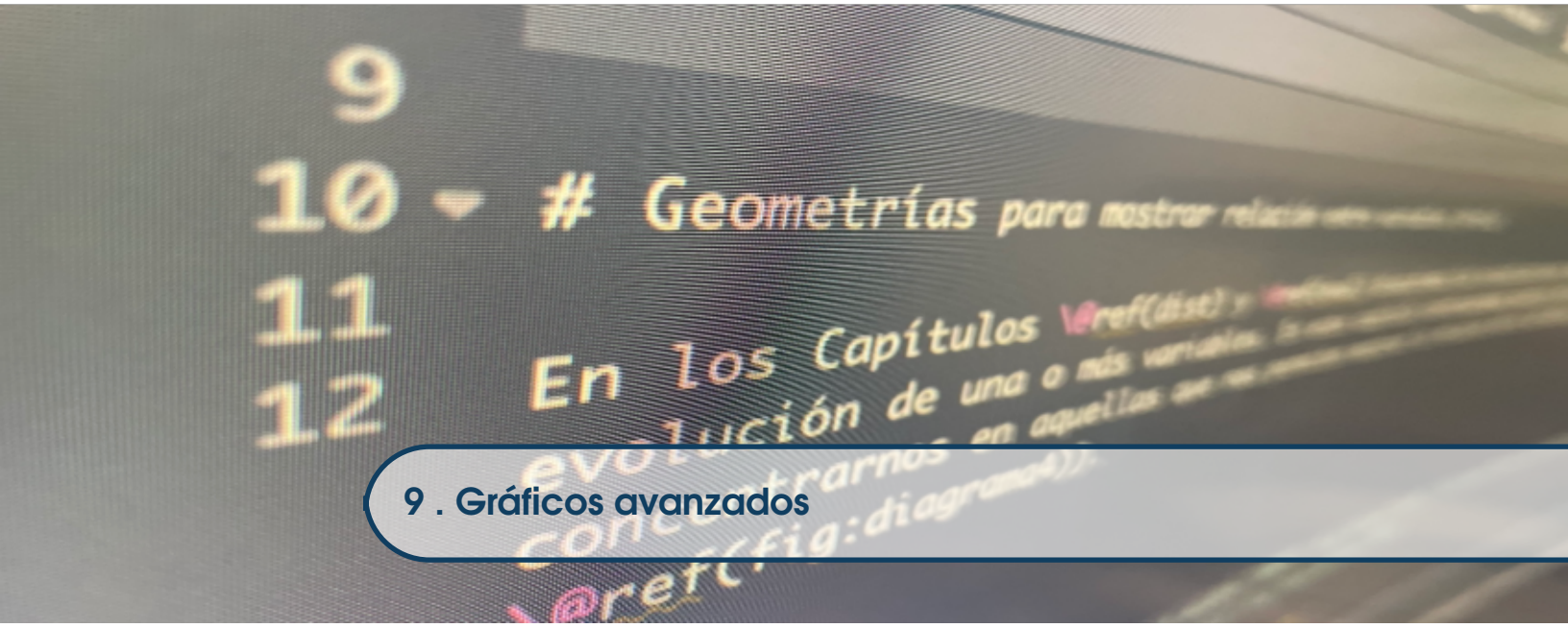


Fuente: Datos del paquete gapminder.

Figura 8.17. Visualización personalizada con logo y correo electrónico



Fuente: Datos del paquete gapminder.



9 . Gráficos avanzados

En los Capítulos 3, 4 y 5 se discutieron tipos de gráficos (capa de ***Geometría***) que generan visualizaciones de uso relativamente frecuente y sencillas de entender. Sin embargo, hay otras geometrías no tan comunes que pueden cumplir una mejor tarea en comunicar el mensaje, dependiendo del auditorio que se cuente y de la posibilidad de explicar cómo se debe interpretar la visualización.

Las visualizaciones que presentaremos en este capítulo representan un mayor reto a la hora de escribir las líneas de código. Por esta razón, creamos un capítulo especialmente dedicado a gráficos avanzados, para que tengas en cuenta otras posibilidades de mostrar los datos. Así mismo, estas visualizaciones probablemente no serán entendidas por todo tipo de auditorio si no se explica cómo funcionan. Estas visualizaciones son un gran reto, pero en algunas ocasiones podrán ser más potentes que las visualizaciones tradicionales.

Recuerda que la naturaleza de los datos y el tipo de público al que irá dirigida la visualización indicará qué tipo de visualización (capa de ***Geometría***) es apropiada.

9.1 Gráficos avanzados de distribución

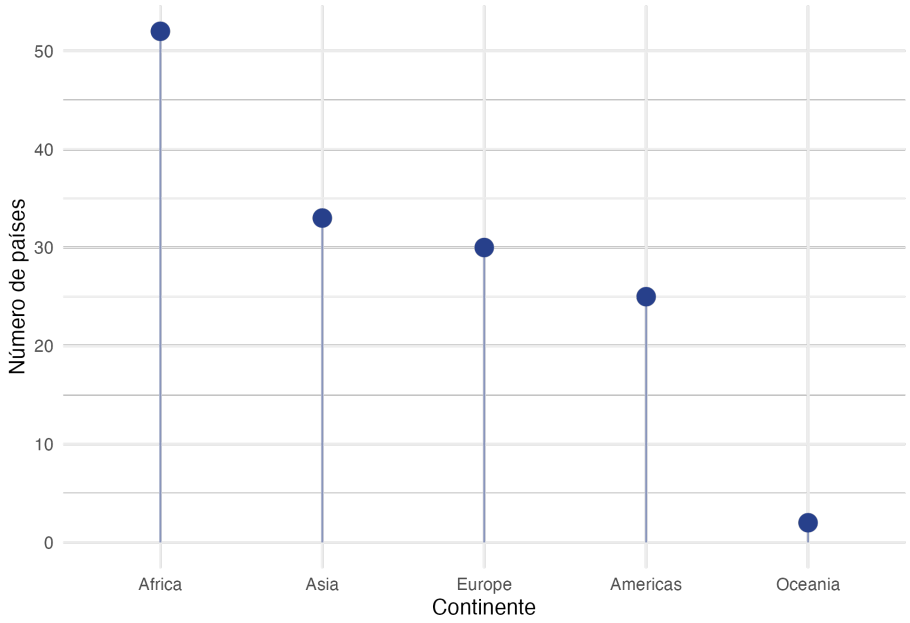
9.1.1 Lollipop

En la Figura 9.1 observamos una visualización conocida como *lollipops*[1]. Esta representa la distribución de países por continente para el

[1] La traducción de *lollipop* sería bombón, chupeta, piruleta o pirulí. Dependiendo del lugar donde te encuentres, la traducción adecuada será diferente. En últimas, nos estamos refiriendo a un dulce que está sujetado por un palo y se consume al chuparlo.

año 2007, empleando la información del paquete *gapminder* (Bryan, 2017).

Figura 9.1. Distribución de los países por continente disponibles en los datos de gapminder para 2007



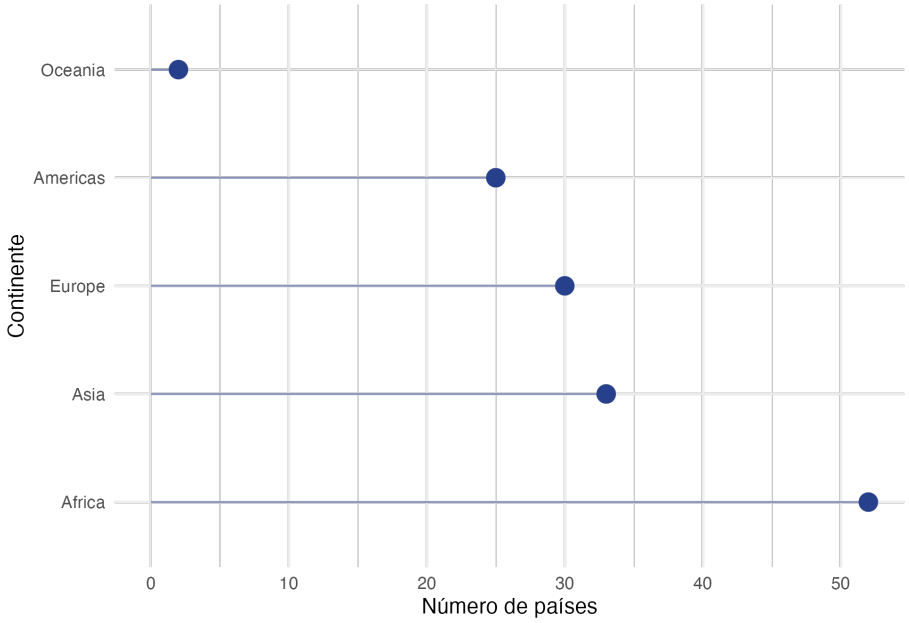
Fuente: Datos del paquete gapminder.

Esta visualización implica una línea "anclada" desde el eje horizontal y un punto al final (círculo) para marcar el valor. También se puede modificar la visualización para que la línea empiece en el eje vertical, como el caso de la Figura 9.2. Siguiendo nuestras recomendaciones para mejorar una visualización, también organizamos los datos.

Esta visualización cumple la misma función del gráfico de barras que vimos en la sección 3.2 (ver Figura 3.3). La ventaja de este tipo de gráfica es que puede ser menos cargada que emplear barras, en especial si se trata de un gran número de observaciones que contienen valores altos; por ejemplo, en el rango de 80 a 90% (sobre 100%). Un gran conjunto de columnas altas puede resultar visualmente agresivo y los *lollipops* funcionarían mejor. La Figura 9.3 nos presenta un ejemplo de esto.

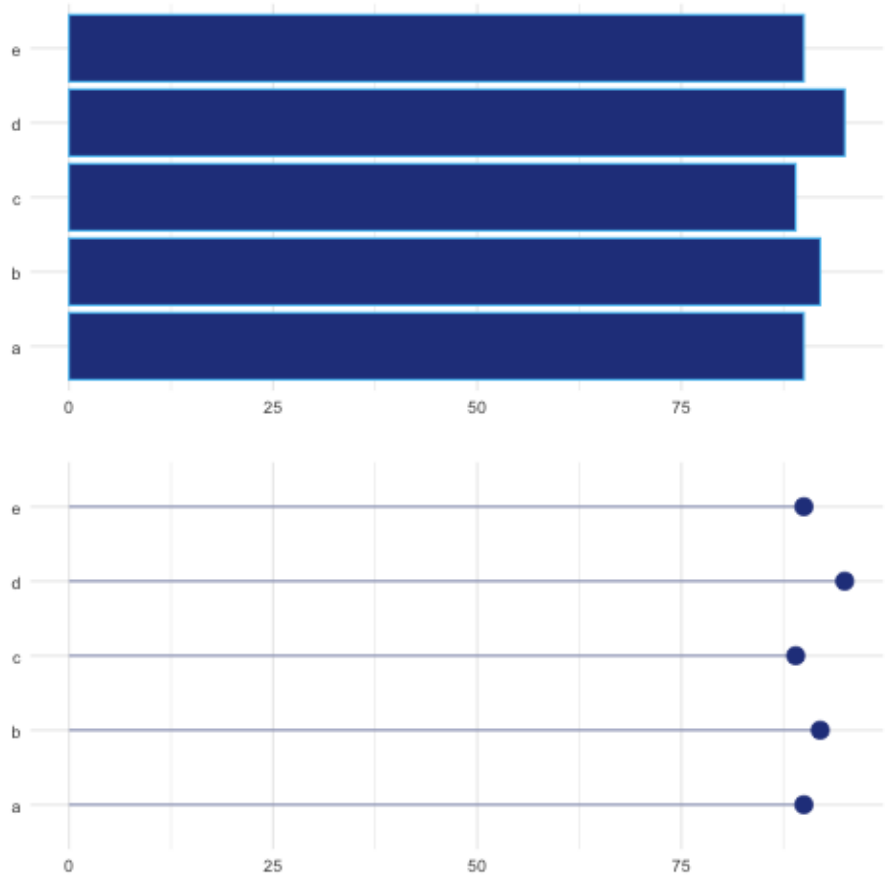
Las visualizaciones de *lollipops* se pueden construir fácilmente con el paquete *ggplot2*, agregando dos capas específicas: los puntos y los segmentos. Para incluir los puntos utilizaremos la función **geom_point()**, mientras que los segmentos serán dibujados utilizando la función

Figura 9.2. Distribución de los países por continente disponibles en los datos de gapminder para 2007



Fuente: Datos del paquete gapminder.

Figura 9.3. Ejemplo cuando funciona mejor un grafico de lolipops que uno de barras



Fuente: Datos del paquete gapminder.

geom_segment(). Esta última función dibuja un segmento que va desde **x** hasta **xend** de forma horizontal y de **y** hasta **yend** de forma vertical. Estos argumentos son necesarios para que nuestra capa de geometría realice un gráfico de *lollipops*.

La Figura 9.1 se puede crear empleando el objeto `gapminder` del paquete con su mismo nombre, haciendo los siguientes pasos:

- Filtrar los datos para el año 2007[2].
- Agrupar las observaciones y contar cuántas observaciones hay por continente.[3]
- Crear la visualización con **ggplot2** empleando una capa de ***Geometría*** de puntos y una capa de ***Geometría*** de segmentos.

Es decir, el código será el siguiente:

```
#Cargar los paquetes
library(ggplot2)
library(gapminder)
library(dplyr)

# se emplea el operador pipe para
# pasar y filtrar los datos

gapminder %>%
  filter(year==2007) %>%
  group_by(continent) %>%
  count() %>%
ggplot(aes(x=continent, y=n)) +
  # crear segmentos
  geom_segment( aes(x=reorder(continent, -n),
                    xend=continent, y=0, yend=n),
                color="royalblue4", alpha = 0.5) +
  #crear puntos
  geom_point(color="royalblue4", size=4) +
  labs(y="Número de países",
       x="Continente")+
  theme_minimal()
```

Sin embargo, utilizar este tipo de visualización tiene algunas desventajas. El centro del círculo del *lollipop* marca el valor, de la misma forma del borde recto en un gráfico de barras, pero no es fácil de determinar de forma precisa. Además, la mitad del círculo se extiende más allá del

[2]Nota que para esto empleamos la función **filter()** del paquete *dplyr*. Para más información al respecto, puedes consultar Alonso (2022).

[3]Nota que para esto empleamos las funciones **group_by()** y **count()** del paquete *dplyr*. Para más información al respecto, puedes consultar Alonso (2022).

valor que representa, por lo que puede llevar a interpretaciones erradas de la variable que se está mostrando.

9.1.2 Violines

Otra forma de visualizar la distribución de una característica de una muestra (variable) es usando un *gráfico de violines*[4] . Estos son una forma más sofisticada de *Boxplot* (diagrama de cajas) estudiado en la Sección 3.3. Esta visualización también es conocida como *gráfico de densidad de espejo*. La Figura 9.4 muestra un ejemplo de este tipo de gráficos.

En la Figura 9.4 podemos apreciar la distribución de la variable PIB per cápita para el año 2007 por continente. Las partes más anchas de los *gráficos de violines* representan los valores para los cuales hay más observaciones. Por otro lado, las partes más delgadas muestran los valores para los cuales hay menos observaciones (ver Figura 9.4). Para más información, puedes ver el siguiente *video de 3 minutos*.

En general, a pesar de que las estadísticas descriptivas, como la media, la mediana y la desviación estándar son fáciles de calcular, puede ser difícil hacernos una idea de cómo son todos los datos disponibles. En estos casos, es muy útil tener una gráfica de la distribución de los datos como un gráfico de violín.

El gráfico de violín se puede construir en *ggplot2* empleando la capa de ***Geometría*** con ese nombre (en inglés): **geom_violin()**. Empleando el siguiente código, podemos obtener la Figura 9.4:

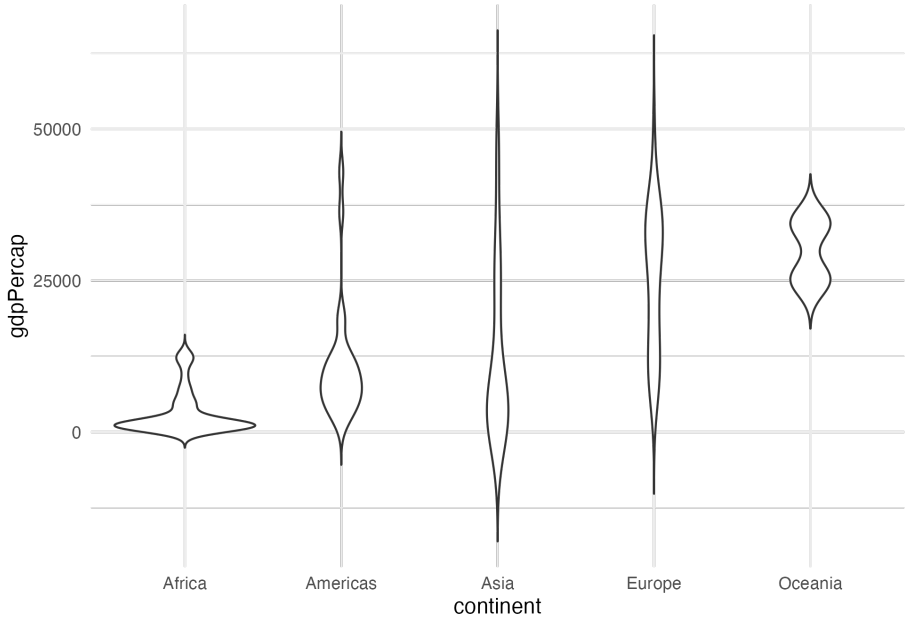
```
# cargar paquetes
library(ggplot2)
library(gapminder)
library(dplyr)

ggplot(gapminder %>%
  filter(year==2007), aes(x=continent, y=gdpPercap))+
  # incluir geometría de violín
  geom_violin(trim=FALSE)+
  theme_minimal()
```

El argumento *trim* de la la función **geom_violin()** hace que la cola de los violines se "aplaste", de forma que no quede cuadrada. Puedes jugar un poco haciendo caso omiso a este argumento.

A su vez, podemos ponerle más información a la gráfica. Agregando a la capa de ***Geometría*** la función **geom_boxplot()** podemos tener un

[4]Como podrás notar, su forma se asemeja a la de un violín. De ahí su nombre.

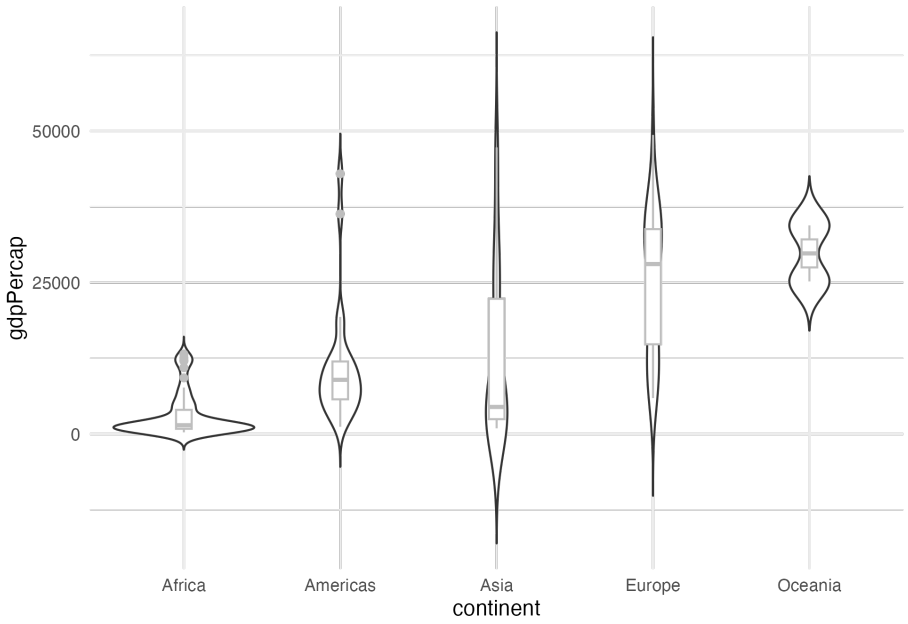
Figura 9.4. Distribución del PIB per cápita por continente para 2007



Fuente: Datos del paquete gapminder.

Boxplot en la misma visualización. Así, obtendríamos la visualización de la Figura 9.5. Intenta crear esta visualización[5].

Figura 9.5. Distribución del PIB per cápita por continente para 2007 según los datos de gapminder



Fuente: Datos del paquete gapminder.

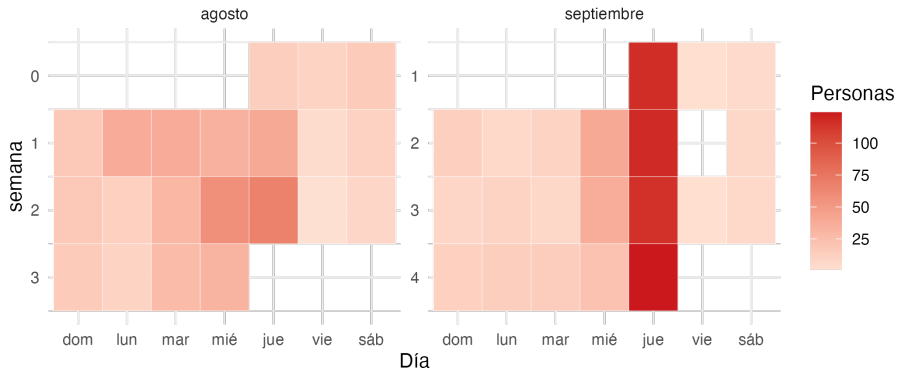
9.1.3 Calendario (distribución en el tiempo)

En ocasiones, contamos con datos de una variable cuantitativa observada diariamente y tenemos la correspondiente fecha para cada observación (una serie de tiempo diaria). La aproximación tradicional es realizar un gráfico de líneas (ver Sección 4.1) para mostrar la evolución en el tiempo de la variable. Pero en algunos casos sería interesante ver cómo se distribuye la variable en los diferentes días. En esos casos un gráfico de *Calendario* podría ser apropiado para mostrar los datos. El objetivo de este tipo de visualizaciones es mostrar los días en los que más observaciones hay, empleando una escala de colores para mostrar las observaciones por día. Es común que el día con más observaciones (o el valor a mapear es el mayor) se le asigne el color mas fuerte.

[5]Pista: emplea el argumento **width** igual a `0.1` en la geometría de *Boxplot* y el color gris para lograr una visualización igual a la presentada.

La Figura 9.6 muestra un ejemplo de este tipo de visualizaciones. Aquí, se muestra el número de visitas que hicieron los estudiantes de un curso virtual a la plataforma donde eran evaluados. El curso tenía una duración de seis semanas. Cada una de estas correspondía a un módulo, el cual concluía con un cuestionario. Los estudiantes tenían de jueves a jueves para acceder y realizar la evaluación correspondiente de la semana.

Figura 9.6. Número de visitas a la plataforma por día



Fuente: Datos propios.

Es fácil apreciar, por la intensidad del color, que los días jueves fueron en que más visitas tuvo la plataforma; relacionado con el último día en el que estaba la disponible la actividad de la semana.

Este gráfico es fácil de replicar. Sin embargo, hay que tener en cuenta que los datos deben tener el formato adecuado. Hemos dispuesto el archivo `calendario.RData` para que puedas ver cómo funciona el código y la estructura que deben tener los datos. En el archivo, hay guardado un objeto con el nombre de `calendario`. Este tiene cuatro variables correspondientes al día, semana, número de estudiantes y mes. Puedes comprobar fácilmente con la función **str()** (R Core Team, 2018) que la clase de las variables `semanadia` y `mes_texto` es **factor**, con sus respectivos niveles asignados[6]. Este paso es importante, pues determinará el orden en que se visualizarán los meses y los días en nuestra visualización.

A continuación, usamos la función **ggplot()** para iniciar nuestro gráfico. La capa de ***Geometría*** que llamaremos es la función **geom_tile()**. El eje **x** corresponderá al día y el eje **y** al número de la semana. El argumento **fill**, que determina la variable que coloreará cada día, será el número de estudiantes. La capa de ***Facets*** la modificaremos usando la

[6]Para más información al respecto, puedes consultar Alonso (2022).

función **facet_wrap()**, donde especificamos la variable para dividir los gráficos después del operador virgulilla (~). Por defecto, el orden de la variable `semana` será determinado de menor a mayor. Para cambiar esto, modificamos la capa de ***Escalas*** con la función **scale_y_reverse()**, la cual invierte el orden del eje.

Una función opcional que empleamos para personalizar nuestra visualización es **scale_fill_gradient()**, que cambia la capa de ***Escalas***. Esta nos permite elegir los colores de los valores altos y bajos del **fill** con los argumentos **high** y **low** respectivamente. Para cambiar el nombre de los ejes y leyendas, llamamos a la función **labs()**. Puedes jugar un poco con las funciones y argumentos para cambiar los resultados a tu gusto. Los datos necesarios los puedes descargar en la página Web del libro.

```
# cargar paquetes
library(ggplot2)
library(dplyr)

# cargar los datos
load("./09-avanz/calendario.RData")


ggplot(calendario, aes(x = semanadia, y = semana)) +
  geom_tile(aes(fill = n), colour = "white") +
  facet_wrap(~mes_texto, scales = "free") +
  scale_y_reverse() +
  theme_minimal() +
  scale_fill_gradient(low="#fee0d2",high="#cb181d")+
  labs(fill="Personas", x="Día")
```

9.2 Gráficos avanzados de evolución

9.2.1 Dumbbell

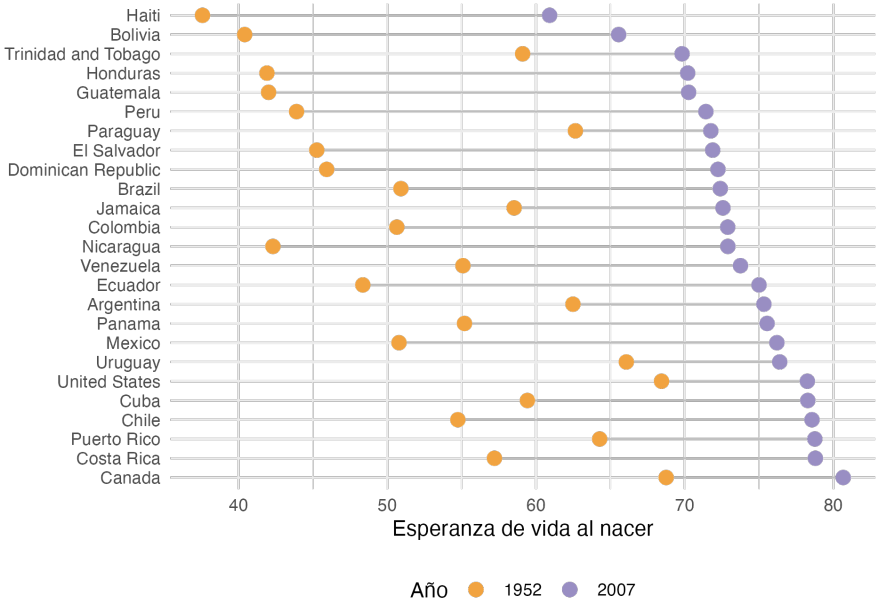
La Figura 9.7 nos presenta una visualización llamada *dumbbell*[7]. Este tipo de gráficas muestran la evolución de variables de clase **numeric** o **integer**.

A diferencia de los gráficos de líneas presentados en la sección 4.1, los *dumbbell* pueden mostrar de manera más puntual la comparación entre dos periodos de tiempo específicos (recordemos que los gráficos de líneas nos muestran varios períodos de tiempo, como es el caso mostrado por la Figura 4.3).

[7] El nombre de este gráfico viene de su parecido a las mancuernas, un tipo de pesa usada para hacer ejercicio, la cual es llamada *dumbbell* en inglés.

La Figura 9.7 es un ejemplo de gráfico de *dumbbell*, usando los datos del paquete *gapminder* (Bryan, 2017). Esta visualización muestra la evolución de la esperanza de vida al nacer entre 1952 y 2007 para cada uno de los países del continente americano.

Figura 9.7. Evolución de la esperanza de vida al nacer de América entre 1952 y 2007



Fuente: Datos del paquete gapminder.

Antes de realizar esta visualización es necesario acomodar la base de datos. Para eso, seguiremos los siguientes pasos:

- Filtrar los datos, teniendo en cuenta que las observaciones que necesitamos deben corresponder a América y ser del año 1952 o 2007[8].
- Seleccionar las variables que vamos a usar[9]. En este caso, necesitamos el país, el año y la expectativa de vida al nacer.
- Pasar los datos a formato ancho o *wide*[10].

[8]Nota que para esto empleamos la función **filter()** del paquete *dplyr* y los operadores lógicos **&** y **|**. Para más información al respecto, puedes consultar Alonso (2022).

[9]Para esto, usamos la función **select()** del paquete *dplyr*. Para más información al respecto, puedes consultar Alonso (2022).

[10]Para lograr esto, usamos la función **pivot_wider()** del paquete *tidyr* (Wickham et al., 2023b).

- Usamos **ggplot2** para crear la visualización. Para los *dumbbell* se deben tener en cuenta tres capas de la geometría: El punto inicial, el punto final y el segmento que los une. Esto lo haremos a través de las funciones **geom_point()** y **geom_segment()**.

```
# Cargar los paquetes

library(ggplot2)
library(gapminder)
library(dplyr)
library(tidyr)


gapminder %>%
  filter(continent=="Americas" & year==2007 |
           continent=="Americas" & year==1952) %>%
  select(country, year, lifeExp) %>%
  pivot_wider(names_from = year, values_from = lifeExp) %>%
ggplot() +
  geom_segment( aes(x=reorder(country, -`2007`),
                    xend=country, y=`1952`, yend=`2007`),
                color="grey") +
  geom_point( aes(x=country, y=`1952`, color="1952"),  size=3) +
  geom_point( aes(x=country, y=`2007`,color="2007"),  size=3 ) +
  coord_flip()+
  scale_color_manual(values = c(`1952` = "#f1a340",
                                `2007`= "#998ec3"))+
  labs(color="Año",x="",y="Esperanza de vida al nacer")+
  theme_minimal() + theme(legend.position = 'bottom')
```

Siguiendo las recomendaciones para mejorar las visualizaciones presentadas en el Capítulo 6, reorganizamos los datos empleando la función **reorder()** del core de R (R Core Team, 2018). Esta función organiza las observaciones de menor a mayor de acuerdo a la expectativa de vida del año 2007 (puedes jugar un poco con las anteriores líneas de código para cambiar la salida a tu gusto). También, debemos incluir dos veces la función **geom_point()**; especificando cada variable que irá en el punto. En nuestro ejemplo, corresponde a la esperanza de vida al nacer en los años 1952 y 2007. Para mayor facilidad a la hora de interpretar este gráfico, hemos agregado colores en los puntos, para identificar a qué año se refiere la información.

Nota que la construcción de este gráfico es muy parecido al del *lollipop*. Sin embargo, el segmento no está anclado a ningún eje y hay un **geom_point()** extra, indicando la posición del otro punto.

Utilizar este tipo de visualización tiene la ventaja de que el análisis

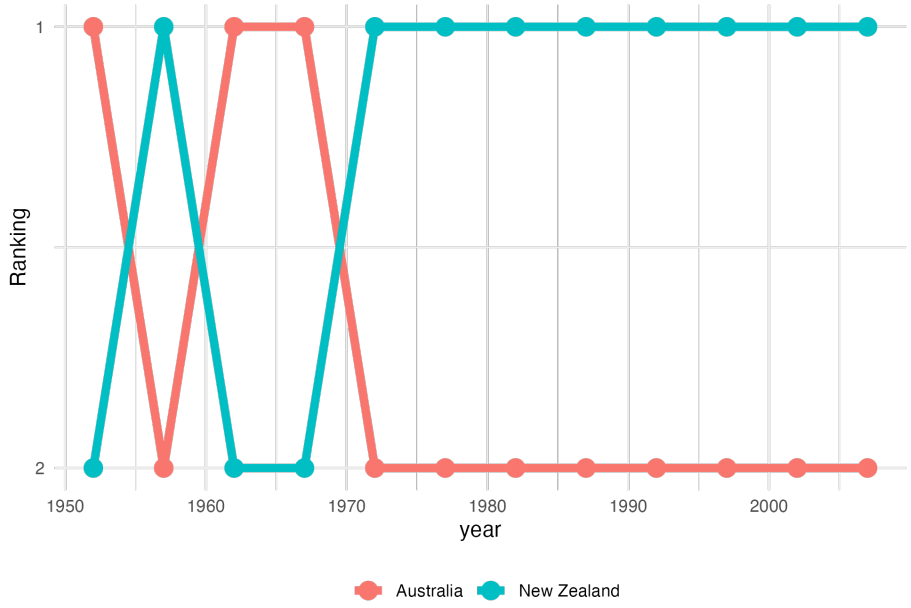
se concentra en solo dos periodos de tiempo; pero dependiendo de lo que se quiere mostrar, puede que se necesite una evolución que incluya más información, como lo realizamos en la sección 4.1.

9.2.2 Ranking (evolución de una variable ordinal)

La Figura 9.8 muestra como cambió la posición año tras año en el *Ranking* de los países de Oceanía con la mayor esperanza de vida al nacer, empleando los datos disponibles en el objeto `gapminder`.

Estas visualizaciones muestran el lugar que ocupan las observaciones al ordenar de mayor a menor (o viceversa), la variable a la que le queremos hacer el ranking. De esta forma, podemos ver la evolución de la variable desde otra perspectiva.

Figura 9.8. Ranking de la esperanza de vida al nacer para los países de Oceanía



Fuente: Datos del paquete gapminder.

Para realizar la Figura 9.8, puedes emplear las siguientes líneas de código:

```
#Se cargan los paquetes

library(ggplot2)
library(gapminder)
library(dplyr)


ggplot(gapminder %>%
  filter(continent=="Oceania") %>%
  group_by(year) %>%
  arrange(lifeExp)%>%
  mutate(Ranking = row_number()), aes(x = year,
                                      y = Ranking,
                                      group = country))+
  geom_line(aes(color = country), size = 2) +
  geom_point(aes(color = country), size = 4) +
  scale_y_reverse(breaks = 1:nrow(gapminder %>%
  filter(continent=="Oceania") %>%
  arrange(lifeExp)%>% mutate(Ranking = row_number())))+
  theme_minimal() +
  theme(legend.position="bottom") +
  labs(color="")
```

Primero, debemos manipular la base de datos, de forma que creemos un ranking por año de los países de Oceanía para la variable de interés. Para esto, creamos la variable `Ranking`, usando las funciones **filter()**, **group_by()**, **arrange()**, **mutate()** y **row_number()**, del paquete *dplyr*[11].

En la capa de ***Aesthetics*** agregamos las variables que queremos dibujar (mapear). En nuestro ejemplo, serán el año (`year`), `Ranking` y el país (`country`). La capa de ***Geometría*** es manipulada con las funciones **geom_line()** y **geom_point()**.

Una buena práctica para la realización de estos gráficos es organizar la escala del eje `y` (que muestra la posición en el ranking), de forma que el número uno se encuentre en la posición más alejada del eje `x`, mostrando la primera posición. Para esto, usamos la función **scale_y_reverse()** para cambiar la capa de ***Escalas***.

Una visualización de evolución de *Ranking* es muy útil para determinar fácilmente qué observación tiene el valor mayor en una variable; sin embargo, tiene la desventaja de que no muestra los valores individuales, solo el orden. Es decir, sabremos que la posición uno tiene un mayor valor que la segunda, pero no tenemos conocimiento de cuál es la diferencia entre ambas observaciones.

[11] Para más información al respecto, puedes consultar Alonso (2022).

9.3 Gráficos avanzados de composición

Cuando la naturaleza de los datos muestra cómo se distribuye la información en un solo período de tiempo, lo más apropiado es un gráfico de composición. Estos gráficos buscan mostrar en una foto las principales características de la información, como los subgrupos que hacen parte de la muestra en un momento específico de tiempo.

9.3.1 Treemap

Los *treemap*[12] muestran la distribución de los datos en un solo periodo de tiempo usando rectángulos para identificar qué observaciones componen los grupos y subgrupos.

Como en los ejemplos de composición que vimos en la sección 3, el área de los *treemap* es proporcional al valor que están graficando; pero tienen la ventaja de que es posible agregar subgrupos. Esto permite que la información sea más detallada que los gráficos simples de composición.

Es fácil identificar, según la Figura 9.9, que, para el año 2007, el país con la mayor población es China, como muestra el tamaño del rectángulo correspondiente a este país.

Para hacer este gráfico se utilizó la base de datos del paquete *gapminder* (Bryan, 2017) , y se realizaron los siguientes pasos:

- Primero debemos instalar dos librerías: *treemap* (Tennekes, 2023) y *treemapify* (Wilkins, 2021). Estas nos permitirán hacer *treemaps* usando la estructura de *ggplot2*. Es decir, esta librería incluye una capa de ***Geometría*** nueva (**geom_treemap**) que funciona como todas las otras capas vistas hasta ahora.
- Filtramos el objeto *gapminder* para que los datos nos muestren la composición de un solo año. En este caso, usaremos 2007[13].
- Realizamos la gráfica, teniendo en cuenta agregar en la geometría la función **geom_treemap()** del paquete *treemapify*.

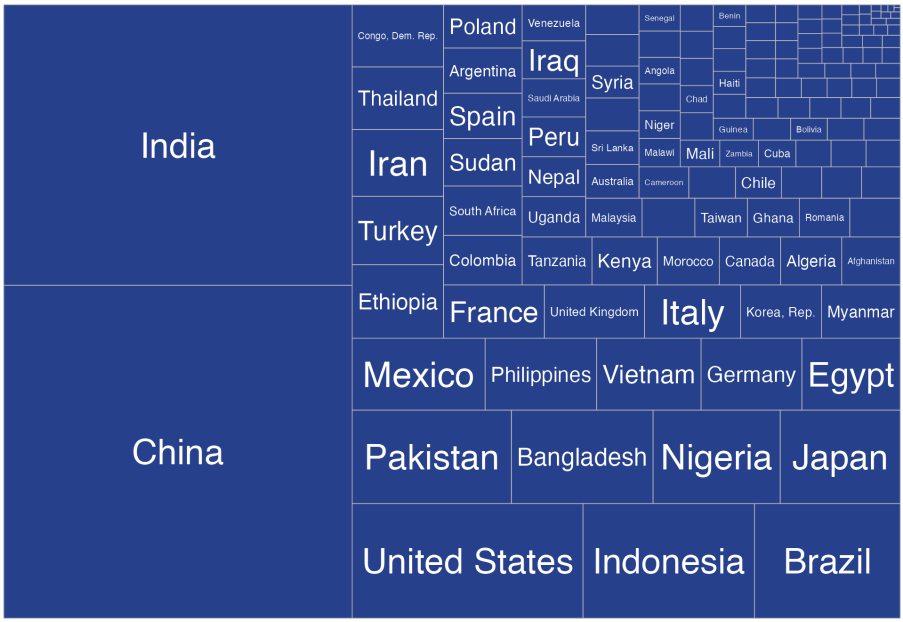
Para la realizar la gráfica 9.9 puedes correr las siguientes líneas de código:

```
#Se cargan los paquetes

library(ggplot2)
```

[12]Una traducción para este término es "mapeo de árboles". Pero es poco usual que se emplee este término en español para referirnos a esta visualización. Es más común emplear el término en inglés.

[13]Para esto, usamos la función **filter()** del paquete *dplyr*. Para más información al respecto, puedes consultar Alonso (2022).

Figura 9.9. Composición de la población por continente para el año 2007



Fuente: Datos del paquete gapminder.

```
library(gapminder)
library(treemap)
library(treemapify)
library(dplyr)


gapminder %>%
  filter(year==2007) %>%
ggplot(aes(area = pop, label = country)) +
  geom_treemap(fill = "royalblue4") +
geom_treemap_text(colour = "white", place = "centre")+
  theme_minimal() + theme(legend.position = 'bottom')
```

Hasta ahora, hemos revisado cómo realizar un *treemap* graficando una variable numérica (correspondiente al área del rectángulo) y una categórica (la observación representada por el nombre en el rectángulo). Sin embargo, podemos agregar otra variable numérica a través del color, como lo muestra la Figura 9.10.

En este ejemplo, estamos graficando la población (mapeada al área del rectángulo) y la expectativa de vida (mapeada al color del relleno) por país, representadas por el área y la intensidad del color del rectángulo, respectivamente.

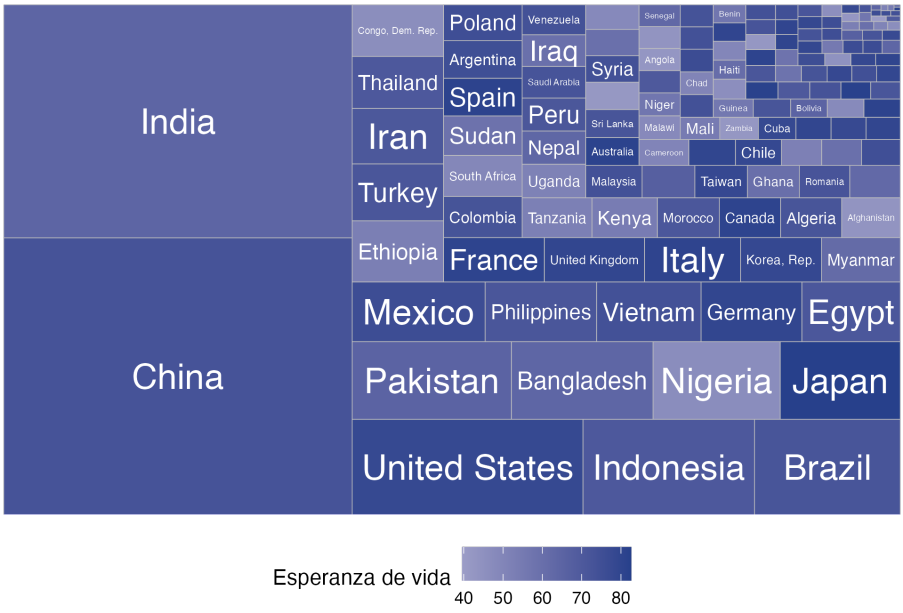
Para lograr la Figura 9.10, vamos a modificar un poco las líneas de código de la Figura 9.9, de la siguiente forma:

- Agregamos en la capa de ***Aesthetics*** el argumento *fill* y la variable que será representada a través del color. En este caso, es la expectativa de vida al nacer.
- Por defecto, el color del *treemap* más claro representará el valor más alto de nuestra nueva variable numérica. Para modificar esto, y a su vez elegir la escala de colores, cambiaremos la capa de ***Escalas*** a través de la función **scale_fill_gradient2()**, incluida en el paquete *ggplot2*.

Implementar las siguientes líneas de código nos permitirá que la expectativa de vida al nacer vaya del color blanco al color azul, correspondiendo el tono más fuerte a un mayor valor de la variable.

```
#Se cargan los paquetes

library(ggplot2)
library(gapminder)
library(treemap)
library(treemapify)
library(dplyr)
```

Figura 9.10. Composición y esperanza de vida por país para el año 2007

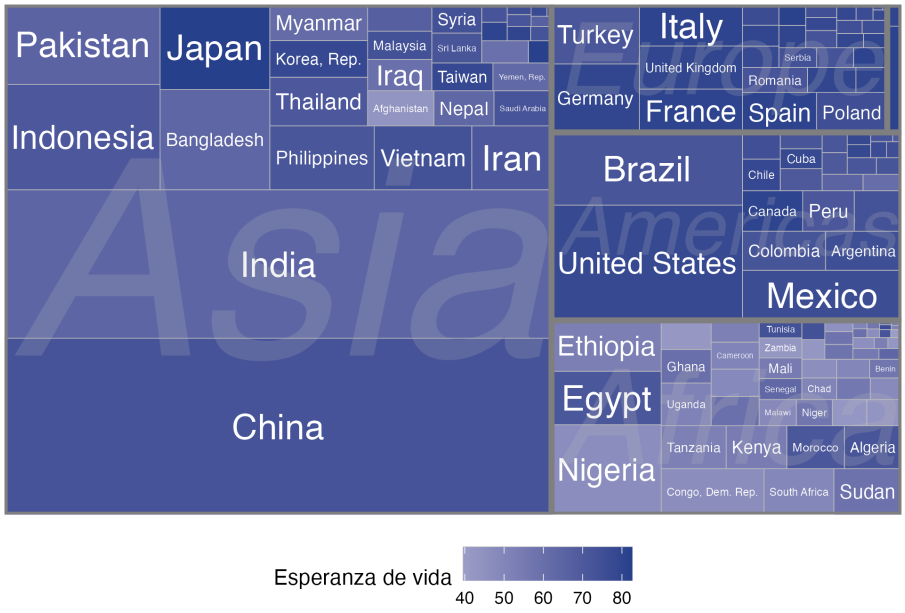


Fuente: Datos del paquete gapminder.

```
gapminder %>%
  filter(year==2007) %>%
ggplot(aes(fill = lifeExp, area = pop, label = country)) +
  geom_treemap() +
geom_treemap_text(colour = "white", place = "centre")+
  scale_fill_gradient2(low="white", high = "royalblue4") +
  labs(fill = "Esperanza de vida") +
  theme_minimal() + theme(legend.position = 'bottom')
```

Ya hemos logrado graficar dos variables numéricas y una categórica usando un *treemap;* pero esto no es todo lo que permite hacer este tipo de visualización. Como ya lo mencionamos, es posible agregar las observaciones en subgrupos. Por la naturaleza de los datos, tenemos que los continentes son la variable categórica disponible que vamos a utilizar, tal y como se muestra en la Figura 9.11.

Figura 9.11. Composición y esperanza de vida por país



Fuente: Datos del paquete gapminder.

Esto lo podemos lograr agregando en la capa de ***Aesthetics*** el argumento *subgroup*, especificando la variable continente, como se muestra en las siguientes líneas de código:

```
#Se cargan los paquetes

library(ggplot2)
library(gapminder)
library(treemap)
library(treemapify)

gapminder %>%
  filter(year==2007) %>%
ggplot(aes(fill = lifeExp, area = pop, label = country,
           subgroup = continent)) +
  geom_treemap() +
  geom_treemap_subgroup_border() +
geom_treemap_text(colour = "white", place = "centre")+
  scale_fill_gradient2(low="white", high="royalblue4") +
  geom_treemap_subgroup_text(place = "centre", grow = T,
                             alpha = 0.2, colour ="grey",
                             fontface = "italic") +
  labs(fill = "Esperanza de vida") +
  theme_minimal() + theme(legend.position = 'bottom')
```

Los *treemaps* son una buena herramienta para mostrar la composición de las observaciones y sus subgrupos cuando no hay muchos individuos que vayan a ser graficados. Otra desventaja de estas visualizaciones es que puede ser difícil comparar las áreas o la intensidad del color si los valores son muy parecidos entre sí.

9.3.2 Waffle

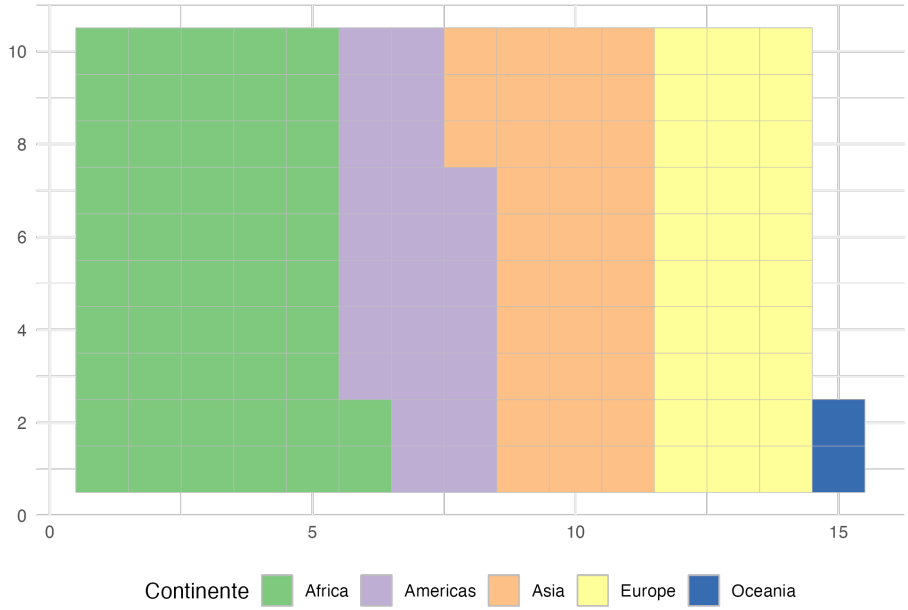
Los gráficos de *waffle* son otra alternativa para mostrar la composición de los datos en un solo período de tiempo. Cada observación está representada por un cuadrado, y el color de su área corresponde al grupo del que hace parte.

La Figura 9.12 muestra la cantidad de países que hay por continente en 2007, según la base de datos *gapminder* (Bryan, 2017). Como podrás ver, los *waffle* son una forma sencilla y poco detallada de mostrar composición, pues el color nos indica a qué continente pertenece cada país (cuadro).

Es importante notar que los *waffle* tienen sentido cuando se van a graficar variables clase **integer**, pues estamos contando las veces en las que las observaciones se encuentran en una categoría.

Para realizar la Figura 9.12, primero necesitamos instalar el paquete *waffle* (Rudis y Gandy, 2019), que contiene la geometría necesaria para crear el gráfico de *waffle*. Si no encuentras una versión del paquete

Figura 9.12. Número de países por continente



Fuente: Datos del paquete gapminder.

adecuada para tu versión de R en el repositorio de CRAN, puedes descargar uno del repositorio del creador del paquete (Ver código).

Antes de pasar a usar **ggplot()** para realizar nuestro *waffle*, debemos crear una base de datos a partir de *gapminder* que tenga el número de países por continente en un año. Para esto, primero usaremos la función **filter()** para seleccionar el año 2007; después, agruparemos por la variable `continent`, usando la función **group_by()**; finalmente, contaremos las observaciones agrupadas con la función **count()**. Las funciones anteriores están disponibles en el paquete *dplyr* (Wickham et al., 2023a)[14].

```
#Se cargan los paquetes
library(gapminder)
library(dplyr)

datos.waffle <- gapminder %>%
  filter(year==2007) %>%
  group_by(continent) %>%
  count()
```

Ahora tenemos el objeto `datos.wafle` con dos columnas: `continent` y `n`. Esta última corresponde al número de países que compone cada continente. Ahora, construyamos nuestra visualización especificando la capa ***Aesthetics*** con los argumentos **fill** para el color de relleno y **values** para el valor. Después, adicionaremos la capa de ***Geometría*** con la función **geom_waffle()** y terminamos personalizando nuestra visualización.

```
#Se cargan los paquetes

library(ggplot2)
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)
library(scales)


datos.waffle %>%
ggplot(aes(fill = continent, values=n)) +
  geom_waffle(colour="grey", na.rm = TRUE)+
  scale_fill_manual(values = c("#7fc97f","#beaed4",
                               "#fdc086","#ffff99","#386cb0"))+
  scale_y_continuous(breaks= pretty_breaks())+
    scale_x_continuous(breaks= pretty_breaks())+
```

[14]Para más información al respecto, puedes consultar Alonso (2022).

```
  labs(fill="Continente")+
  theme_minimal()+ theme(legend.position = 'bottom')
```

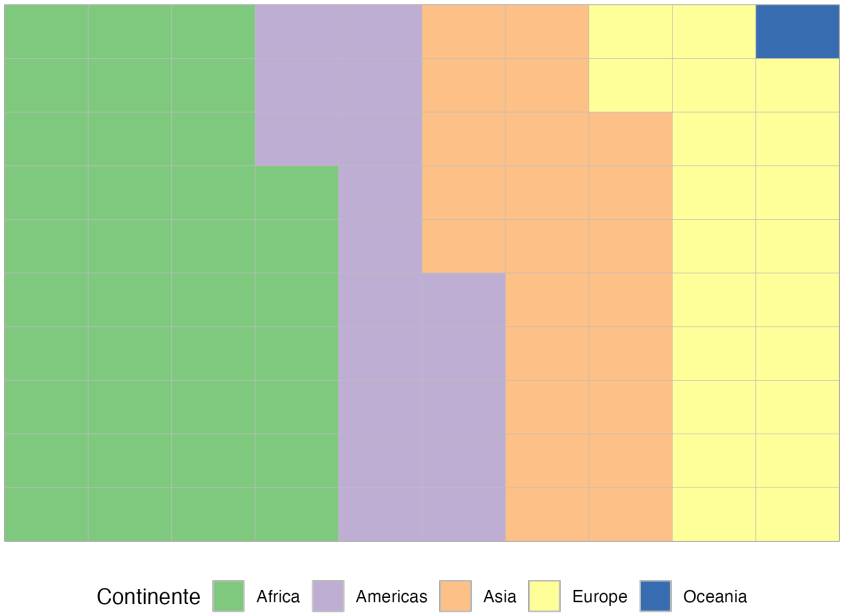
En la función **geom_waffle()** empleamos dos argumentos. El primer argumento es **colour** que define el color de las líneas que dividen los cuadrados. El segundo argumento, **na.rm** determina si se remueven o no los NA de los datos.

Adicionalmente, para personalizar nuestro gráfico, hemos decidido agregar un vector de colores de forma manual. Esto implica modificar la capa de ***Escalas*** por medio de la función **scale_fill_manual()** y empleando como argumento los colores que deseamos emplear. Otra sugerencia a la hora de emplear este tipo de visualización es asegurarse de que los ejes sean números enteros, pues no tiene mucho sentido utilizar otra escala para estos gráficos. Lo anterior lo resolvemos modificando la capa de ***Escalas***, usando la función **scale_y_continuous()** y **scale_x_continuous()**, del paquete *ggplot2*, y especificando el argumento **breaks** con la función **pretty_breaks()**, disponible en el paquete *scales* (Wickham y Seidel, 2022).

Noten que en este caso se cuenta con 142 países. Por eso, tenemos dos cuadros que parecen desalineados. Una variante del gráfico de *waffle* es expresar las categorías no como número (términos absolutos) sino como porcentaje (términos relativos). Pero será necesario aproximar los porcentajes a números enteros. La Figura 9.13 muestra los mismos datos de la Figura 9.12 pero en forma de porcentajes, aproximando a números enteros. Si bien se pierde precisión con esta aproximación, es visualmente "más" agradable que la primera versión (Figura 9.12) y da una idea aproximada de la composición de los países por continente. Esto se logra asignándole al argumento **make_proportional** el valor de TRUE en la función **geom_waffle()**. Este argumento es por defecto igual a FALSE.El código para esta figura se muestra a continuación.

```
#Se cargan los paquetes

library(ggplot2)
library(gapminder)
library(dplyr)
# install.packages("waffle", repos = "https://cinc.rud.is")
library(waffle)
library(scales)

datos.waffle %>%
ggplot(aes(fill = continent, values=n)) +
  geom_waffle(colour="grey", na.rm = TRUE,
              make_proportional = TRUE)+
```

Figura 9.13. Proporción de países por continente

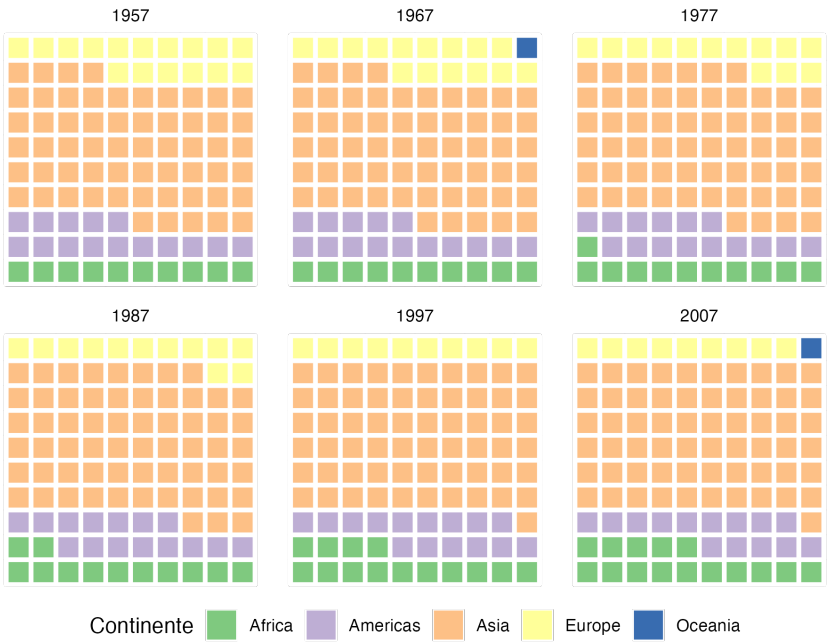


Fuente: Datos del paquete gapminder.

```
  scale_fill_manual(values = c("#7fc97f","#beaed4",
                               "#fdc086","#ffff99","#386cb0"))+
  scale_y_continuous(breaks= pretty_breaks())+
    scale_x_continuous(breaks= pretty_breaks())+
  labs(fill="Continente")+
  theme_void()+ theme(legend.position = 'bottom')
```

El gráfico de *waffle* tambien puede ser útil par mostrar la evolución de la composición. Por ejemplo, en la Figura 9.14 se presenta la evolución a través del tiempo de la participación porcentual en la población de cada uno de los continentes.

Figura 9.14. Evolución de la particiapción porcentual de cada continente en la población mundial



Fuente: Datos del paquete gapminder.

La Figura 9.14 se puede construir empleando el siguiente código:

```
# Se crea la base de datos
datos.wafle.pob <- gapminder %>%
  group_by(continent, year) %>%
  summarise(pop= sum(pop)) %>%
  filter(year %in% c(1957, 1967, 1977, 1987, 1997, 2007))
```

```
# Se crea la visualización
datos.wafle.pob %>%
ggplot(aes(fill = continent, values = pop)) +
  geom_waffle(color = "white", size = 1.125, n_rows = 10,
              flip = TRUE, na.rm = TRUE,
              make_proportional = TRUE) +
  facet_wrap(~year, ncol = 3) +
  scale_x_continuous(breaks= pretty_breaks())+
  scale_y_continuous(breaks= pretty_breaks())+
  scale_fill_manual(values = c("#7fc97f","#beaed4",
                               "#fdc086","#ffff99","#386cb0"))+
  coord_equal() +
  theme_void()+
  labs(fill="Continente") +
  theme(legend.position = 'bottom')
```

Nota que en este caso estamos empleando la capa de ***Facets*** para hacer repetir los gráficos de *waffle* por año.

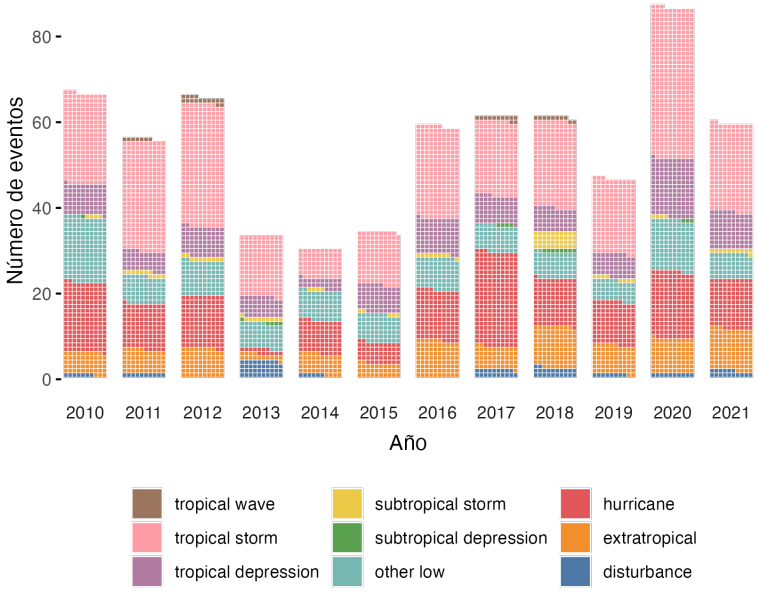
Otra forma interesante de emplear los gráficos de *waffle* para mostrar la evolución de composición se presenta en la Figura 9.15. En este caso, estamos empleando los datos de tormentas en el océano Atlántico (objeto `storms`) disponible en el paquete *dplyr*[15] (Wickham et al., 2023a) . En este caso, se presenta el número de eventos de tormentas (y no la participación). La visualización no solo permite ver la composición, sino también cómo en el año 2020 se presentaron más eventos que los otros años visualizados.

La Figura 9.15 se puede construir empleando el siguiente código:

```
# Se crea la base de datos
library(dplyr)
datos.storms <- storms %>%
                filter(year >= 2010) %>%
                 count(year, status)

# Se crea la visualización
datos.storms %>%
            ggplot(aes(fill = status, values = n)) +
            geom_waffle(color = "white", size = 0.15,
                        n_rows = 10, flip = TRUE, na.rm = TRUE) +
            facet_wrap(~year, nrow = 1, strip.position = "bottom") +
            scale_x_discrete() +
```

[15]Estos datos pertenecen a la base de datos de huracanes del Atlántico de la NOAA de los Estados Unidos. Puedes mirar la ayuda de este objeto para mayor información

Figura 9.15. Evolución de los diferentes tipos de tomentas en el océano Atlantico



Fuente: Datos del paquete dplyr.

```
            scale_y_continuous(breaks= pretty_breaks())+
            ggthemes::scale_fill_tableau(name=NULL) +
            coord_equal() +
            labs(x = "Año", y = "Número de eventos") +
             theme_minimal() +
             theme(legend.position = 'bottom',
                   panel.grid = element_blank(),
                   axis.ticks.y = element_line())  +
            guides(fill = guide_legend(reverse = TRUE,
                                       nrow = 3))
```

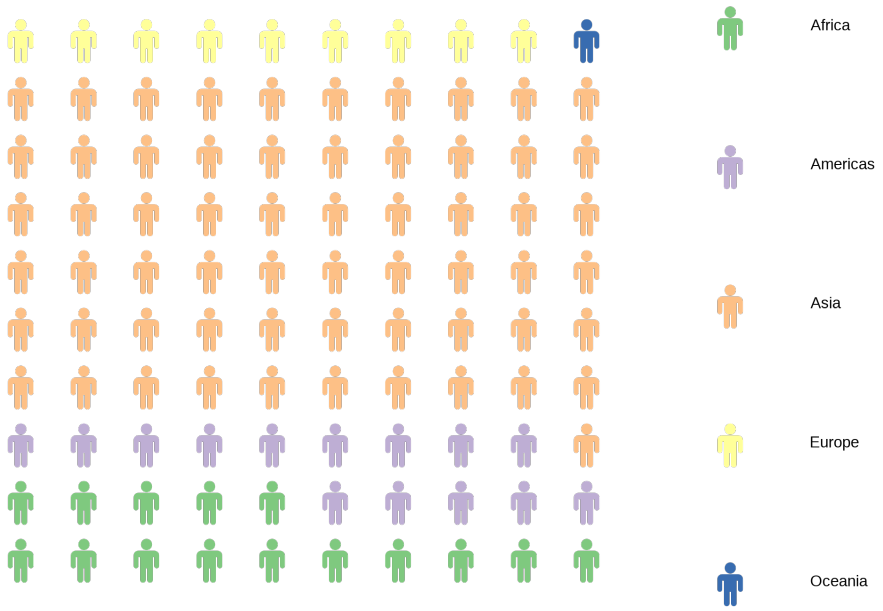
Finalmente, los gráficos de *waffle* se pueden convertir en pictogramas. Un pictograma utiliza íconos o imágenes en lugar de cuadrados. Los pictogramas son especialmente útiles para representar datos en una manera que sea fácil de entender para el público en general y son muy empleados en la construcción de infografías.

El paquete *waffle* (Rudis y Gandy, 2019) incluye la función **geom_pictogram()** que permite emplear en la capa de ***Geometría*** un pictograma. Por ejemplo, la Figura 9.16 presenta un pictograma para la composición de la población por continente para el año 2007.

La Figura 9.16 se puede construir empleando el siguiente código:

```
# Se carga el paquete para emplear los iconos
# si no tienes instalado el paquete instálalo
# install.packages("emojifont")
library(emojifont)
# se carga el tipo de letra necesario para los íconos
load.fontawesome()

# Se crea la base de datos

datos.pob.2007 <- gapminder %>%
  filter(year==2007) %>%
  group_by(continent) %>%
  summarise(pob = sum(pop))

# Se crea la visualización
datos.pob.2007 %>%
    ggplot(aes(label = continent, colour= continent,
               values=pob)) +
    geom_pictogram(n_rows = 10, make_proportional = TRUE,
                   family = 'fontawesome-webfont',
                   size = 25, flip = TRUE,
                   show.legend = TRUE) +
```

Figura 9.16. Pitograma de la participación porcentual en la población total de cada continente



Fuente: Datos del paquete gapminder.

```
    scale_colour_manual(values = c("#7fc97f","#beaed4",
                                   "#fdc086","#ffff99","#386cb0"))+
    scale_label_pictogram( values = c("male")) +
    scale_y_continuous(breaks= pretty_breaks())+
    scale_x_continuous(breaks= pretty_breaks())+
    theme_void()+ theme(legend.position = 'bottom',
                        legend.title = element_blank(),
                        legend.text = element_text(size = 25)) +
    guides(fill = guide_legend(reverse = TRUE))
```

Para emplear los diferentes íconos será necesario intalar el tipo de letra `font awesome-webfont`[16]. Nota que con la función **scale_label_pictogram()** estamos definiendo el ícono que empleamos en el pictograma. en este caso estamos empleando el ícono `male`. Podríanos emplear un icono diferente para cada categoría de la variable que mapeamos al argumento *label*. Empleando la función **search_fontawesome()** del paquete *emojifont* (Yu, 2021) se pueden buscar iconos que concuerden con el objetivo de nuestra visualización. Por ejemplo, el siguiente código busca los nombres de los íconos que contenga en su nombre "male":

```
# se buscan iconos que contengan los caracteres "male"
search_fontawesome("male", approximate = TRUE)
```

```
##  [1] "fa-balance-scale"      "fa-calendar"
##  [3] "fa-calendar-check-o"   "fa-calendar-minus-o"
##  [5] "fa-calendar-o"         "fa-calendar-plus-o"
##  [7] "fa-calendar-times-o"   "fa-female"
##  [9] "fa-file-image-o"       "fa-image"
## [11] "fa-male"               "fa-smile-o"
## [13] "fa-stumbleupon"        "fa-stumbleupon-circle"
```

Nota que el nombre del ícono corresponde a los caracteres que están después de "fa-"; este prefijo corresponde *font awesome* (fa). Ahora intenta usar 5 diferentes íconos para modificar la Figura 9.16, de tal manera que cada continente tenga un ícono diferente. Por ejemplo, corre el siguiente código.

```
datos.pob.2007 %>%
    ggplot(aes(label = continent, colour= continent,
               values=pob)) +
    geom_pictogram(n_rows = 10, make_proportional = TRUE,
```

[16]Esta letra se puede descargar del siguiente enlace: https://fontawesome.com/docs/web/setup/host-yourself/webfonts.

```
                  family = 'fontawesome-webfont',
                  size = 10, flip = TRUE,
                  show.legend = TRUE) +
    scale_colour_manual(values = c("#7fc97f","#beaed4",
                                   "#fdc086","#ffff99","#386cb0"))+
    scale_label_pictogram( values = c("balance-scale",
                                      "object-group",
                                      "apple", "object-ungroup",
                                      "calendar")) +
    scale_y_continuous(breaks= pretty_breaks())+
    scale_x_continuous(breaks= pretty_breaks())+
    theme_void()+ theme(legend.position = 'bottom',
                        legend.title = element_blank()) +
    guides(fill = guide_legend(reverse = TRUE))
```

9.3.3 Sankey

A diferencia de las anteriores visualizaciones, el gráfico de *Sankey* permite visualizar la composición de las observaciones según varias variables cualitativas. Aquí, cada característica (variable) de las observaciones compone un nodo, y va creando un flujo, a través de arcos, de una característica a otra. Esto permite mostrar la relación entre la composición cuando consideramos diferentes variables categóricas.

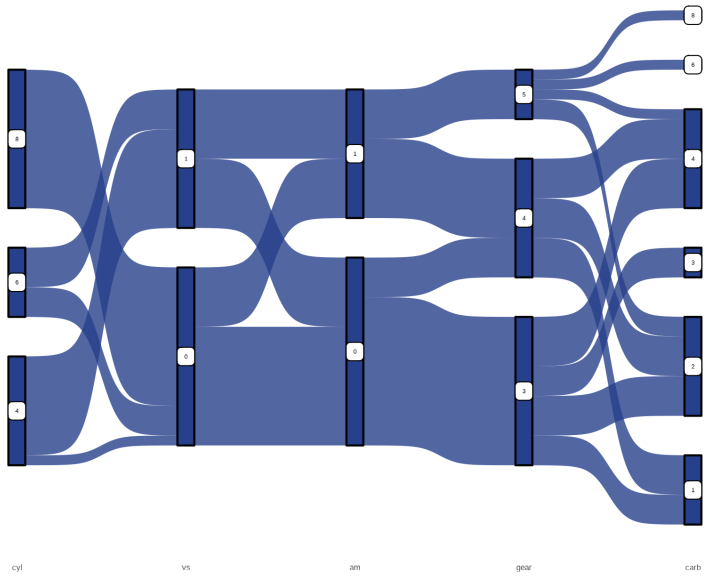
Hay que tener en cuenta que las características que se pueden seguir de una misma observación son las que están ubicadas a la derecha e izquierda de un nodo; esto quiere decir que puedo saber las características de una observación según mi punto de partida o el orden que le de al gráfico.

Como ejemplo, tenemos la Figura 9.17. Esta muestra unos datos extraídos de la revista *Motor Trend US* del año 1974, y están disponibles en el objeto `mtcars`, que hace parte del paquete base de R (R Core Team, 2018). Empezando por los primeros nodos, es fácil ver que, por ejemplo, todos los autos de ocho cilindros (variable `cyl`), corresponden al tipo de motor en forma de V (variable `vs` y categoría `0`)[17]. Y a su vez podemos ver como los carros con motor en V se distribuyen cuando consideramos el tipo de transmisión (variable `am`). Y así sucesivamente para las variables número de cambios (`gear`) y número de carburadores (`carb`).

Para hacer un gráfico de Sankey, es importante tener en cuenta que los datos deben tener una estructura específica. En nuestro caso, la función **make_long()** del paquete *ggsankey* (Sjoberg, 2023) nos permite

[17] Esto lo puedes verificar utilizando el diccionario de datos de la base, disponible en la descripción del objeto en la herramienta de ayuda de RStudio.

Figura 9.17. Composición de las características de los carros en la muestra



Fuente: Datos del paquete base.

llegar fácilmente al resultado deseado especificando las variables de las cuales vendrán los nodos de nuestra visualización.

Las siguientes líneas de código muestran cómo llegar a la Figura 9.17, con la única modificación de cambiar las leyendas para que las categorías sean más claras.

```
install.packages("remotes")
remotes::install_github("davidsjoberg/ggsankey")
library(ggsankey)
library(ggplot2)
library(dplyr)


mtcars$vs<-recode(mtcars$vs, "0"="V-shaped", "1"="straight" )
mtcars$am<-recode(mtcars$am, "0"="automatic", "1"="manual" )


ggplot(mtcars %>%
  make_long(cyl, vs, am, gear, carb),
  aes(x = x,
      next_x = next_x,
      node = node,
      next_node = next_node,
      fill = factor(node),
      label = node)) +
  geom_sankey(flow.alpha = 0.8, node.color = 1) +
  geom_sankey_label(size = 3.5, color = 1,
                    fill = "white",  show.legend = NA) +
  scale_fill_manual(values=replicate(11,"royalblue4"))+
  theme_sankey(base_size = 16) +
  theme(legend.position = "none")+
  labs(x="")
```

Primero, es necesario instalar el paquete *remotes* (Csárdi et al., 2021), el cual permite que R instale paquetes desde repositorios remotos, como Github. Después, usamos la función **install_github()** para descargar e instalar el paquete *ggsankey* (Sjoberg, 2023), el cual contiene las funciones necesarias para realizar nuestra visualización.

Cargamos los paquetes, y, a continuación, hacemos el cambio en las leyendas del objeto `mtcars` (R Core Team, 2018). Utilizando la función **recode()** , modificamos las etiquetas de las variables `vs` y `am`.

Posteriormente, debemos pasar los datos de formato ancho (wide) a formato largo (long). Esto lo logramos usando la función **make_long()**, disponible en el paquete *ggsankey* . Por defecto, esta función creará los

flujos del *Sankey*. Cada observación determinará de dónde a dónde irá el arco. En la capa de ***Aesthetics*** es importante especificar de dónde a dónde irán los flujos, con los argumentos *x*, *next_x*, *node* y *next_node* (nota que la función **make_long()** cambia el nombre de las variables por los argumentos que acabamos de mencionar). Lo siguiente que debemos agregar es la capa de ***Geometría***, utilizando la función **geom_sankey()** (Allaire et al., 2017), para especificar el tipo de gráfico que queremos.

9.4 Comentarios finales

Este capítulo desarrolló visualizaciones que requerían un nivel mayor de comprensión de líneas de código. Dependiendo de tu objetivo y tu público, pueden ser útiles para mostrar fielmente tu información.

Ahora tienes disponible estas visualizaciones en tu caja de herramientas para descrestar a tu público. Esperamos explores en la comunidad de usuarios de R todas las visualizaciones que están disponibles en *ggplot2* o en paquetes que adicionan capas de ***Geometría***. Si quieres ver alguna visualización en especial en futuras ediciones de este libro, no dudes en escribirnos.

Esperamos que con las herramientas aprendidas puedas generar visualizaciones de alto impacto. Recuerda: ¡la imaginación es el límite!

Bibliografía

Abela, A. (2008). *Advanced presentations by design: Creating communication that drives action*. John Wiley & Sons.

Aden-Buie, G. (2023). *ggpomological: Pomological plot themes for ggplot2*. R package version 0.1.2.

Allaire, J., Gandrud, C., Russell, K., y Yetman, C. (2017). *networkD3: D3 JavaScript Network Graphs from R*. R package version 0.4.

Alonso, J. C. (2021). Una introducción a los Loops en R (y algunas alternativas). Icesi Economics Lecture Notes 019408, Universidad Icesi.

Alonso, J. C. (2022). *Empezando a transformar bases de datos con R y dplyr*. Universidad Icesi.

Alonso, J. C. y González, A. (2012). Ggplot: gráficos de alta calidad. *Apuntes de Economía*, (33):29.

Alonso, J. C. y Ocampo, M. P. (2022). *Empezando a usaR: Una guía paso a paso.* Universidad Icesi.

Arnold, J. B. (2021). *ggthemes: Extra Themes, Scales and Geoms for 'ggplot2'*. R package version 4.2.4.

Bryan, J. (2017). *gapminder: Data from Gapminder*. R package version 0.3.0.

Csárdi, G., Hester, J., Wickham, H., Chang, W., Morgan, M., y Tenenbaum, D. (2021). *remotes: R Package Installation from Remote Repositories, Including 'GitHub'*. R package version 2.4.2.

Gross, C. y Ottolinger, P. (2016). *ggThemeAssist: Add-in to Customize 'ggplot2' Themes*. R package version 0.1.5.

Henry, L. y Wickham, H. (2020). *purrr: Functional Programming Tools*. R package version 0.3.4.

Moreno, D. (2015a). *colmaps: Colombian maps*. R package version 0.0.0.9515.

Moreno, D. (2015b). *homicidios: Colombian homicides data*. R package version 0.0.0.9000.

Müller, K. y Wickham, H. (2021). *tibble: Simple Data Frames*. R package version 3.1.6.

Pedersen, T. L. y Robinson, D. (2020). *gganimate: A Grammar of Animated Graphics*. R package version 1.0.7.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rudis, B. (2020). *hrbrthemes: Additional Themes, Theme Components and Utilities for 'ggplot2'*. R package version 0.8.0.

Rudis, B. y Gandy, D. (2019). *waffle: Create Waffle Chart Visualizations*. R package version 1.0.1.

Sievert, C. (2020). *Interactive Web-Based Data iczation with R, plotly, and shiny*. Chapman and Hall/CRC.

Sjoberg, D. (2023). *ggsankey: Sankey, Alluvial and Sankey Bump Plots*. R package version 0.0.99999.

Tennekes, M. (2023). *treemap: Treemap Visualization*. R package version 2.4-4.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wickham, H. (2019). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.4.0.

Wickham, H. (2020). *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 0.5.0.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., y Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686.

Wickham, H., François, R., Henry, L., Müller, K., y Vaughan, D. (2023a). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.1.

Wickham, H. y Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*. .ºReilly Media, Inc.".

Wickham, H., Hester, J., y Francois, R. (2018). *readr: Read Rectangular Text Data*. R package version 1.3.1.

Wickham, H. y Seidel, D. (2022). *scales: Scale Functions for Visualization*. R package version 1.2.1.

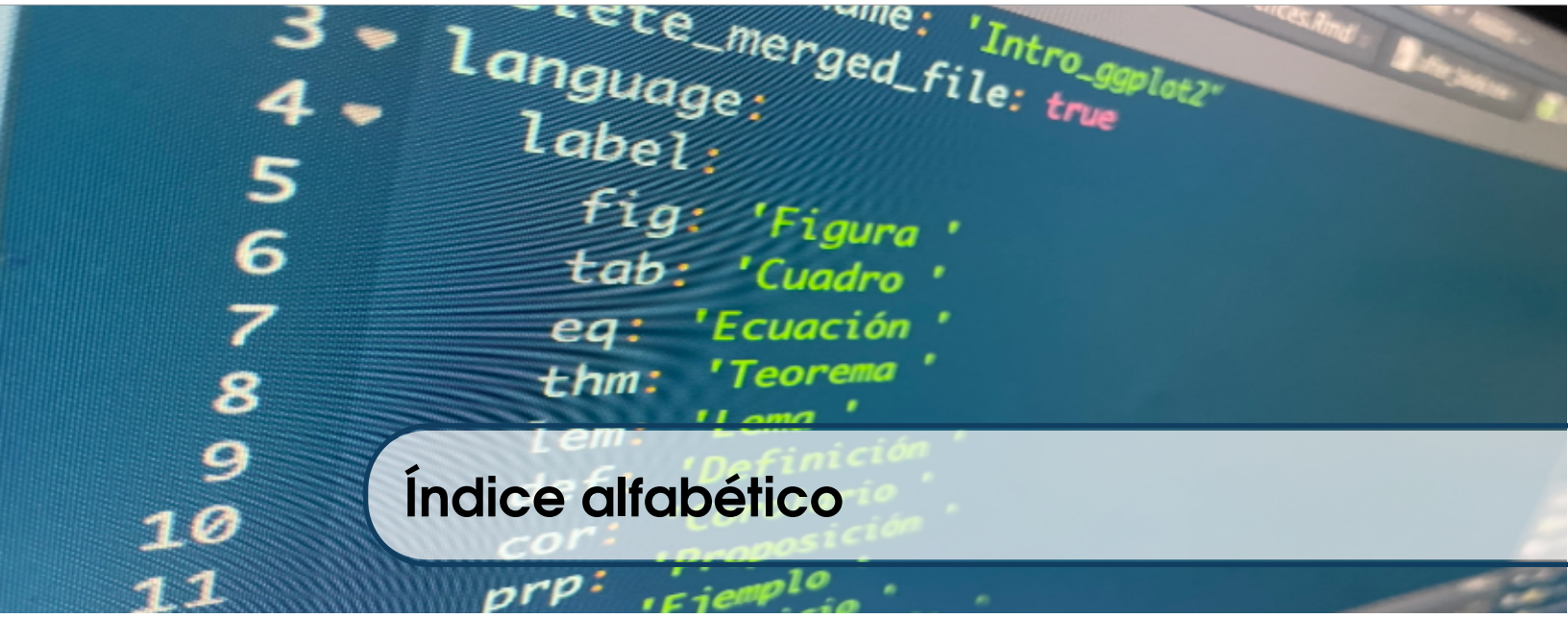
Wickham, H., Vaughan, D., y Girlich, M. (2023b). *tidyr: Tidy Messy Data*. R package version 1.3.0.

Wilkins, D. (2021). *treemapify: Draw Treemaps in 'ggplot2'*. R package version 2.5.5.

Wilkinson, L. (2012). The grammar of graphics. In *Handbook of computational statistics*, pages 375–414. Springer.

Yu, G. (2021). *emojifont: Emoji and Font Awesome in Graphics*. R package version 0.5.5.

Yutani, H. (2020). *gghighlight: Highlight Lines and Points in 'ggplot2'*. R package version 0.3.1.



Índice alfabético

Editorial Universidad Icesi

UNIVERSIDAD ICESI

Si estás leyendo *Empezando a visualizar datos con R y ggplot2*, es porque haces parte de la comunidad que emplea R para analizar datos. Este libro tiene como objetivo presentar una primera aproximación a visualizar datos empleando el paquete *ggplot2*. Este paquete permite realizar visualizaciones rápidamente y de manera más intuitiva que la base de R. Si eres nuevo en el universo de R o en el uso del paquete ggplot2, este libro puede ser un buen punto de arranque. Si ya eres usuario de *ggplot2*, seguramente este libro no te aportará nuevos conocimientos, pero puede ser una herramienta de consulta de algunos conceptos básicos.

Herramientas del BIG DATA y analytics