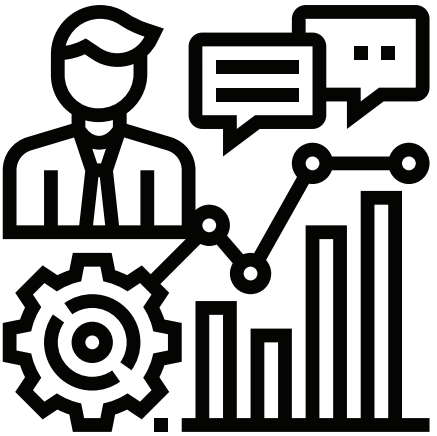
DERECHO Y CIENCIA POLÍTICA

JORDI MAS ELIAS

ANÁLISIS DE DATOS

CON R EN ESTUDIOS INTERNACIONALES



EDITORIAL UOC

13 h

Análisis de datos con R en estudios internacionales

Jordi Mas Elias

EDITORIAL UOC

Diseño de la colección: FUOC
Diseño de la cubierta: Natàlia Serrano
Pictograma de cubierta: icono hecho por Eucalip de www.flaticon.com

Primera edición en lengua castellana: octubre 2020
Primera edición en formato digital (PDF): octubre 2020

© Jordi Mas Elias, del texto

© Fundació per a la Universitat Oberta de Catalunya, de esta edición, 2020
Av. Tibidabo, 39-43, 08035 Barcelona
Marca comercial: Editorial UOC
www.editorialuoc.com

Realización editorial: FUOC

ISBN: 978-84-9180-758-2

La UOC queda facultada expresamente por el/la autor/a para digitalizar y publicar la Obra en un repositorio en línea que será accesible al público bajo licencias Creative Commons, incluyendo la licencia ReconocimientoNo-Comercial-SinObraDerivada (BY-NC-ND), v.4.0 Internacional (jurisdicción internacional), que permitirá copiar, distribuir y transmitir públicamente la Obra siempre citando la autoría y la fuente, sin hacer un uso comercial y sin hacer obra derivada. Si la Obra es transformada, la obra generada estará sometida a una licencia Creative Commons similar o compatible con la licencia mencionada.

Autor

Jordi Mas Elias
Profesor de los Estudios de Derecho y Ciencia Política en la Universitat Oberta de Catalunya (UOC). Se doctoró en 2018 en Ciencia Política, Políticas Públicas y Relaciones Internacionales por la Universidad Autónoma de Barcelona (UAB). Tiene un máster en Economía Política por la London School of Economics y otro en Relaciones Internacionales, Seguridad y Desarrollo por la UAB. Sus principales intereses tanto docentes como investigadores se centran en la Economía Política Internacional y la metodología en Ciencias Sociales.

Índice

Introducción 11

Capítulo I. El lenguaje de R 15
1. Introducción 15
2. La estructura de datos de R 16
   2.1. Objetos y atributos 17
   2.2. Funciones y argumentos 32
3. Variables y tipos 39
   3.1. Variables categóricas nominales 43
   3.2. Variables categóricas ordinales 54
   3.3. Variables numéricas 58

Ejercicios de autoevaluación 64
Solucionario 68
Glosario 70
Bibliografía 73

Capítulo II. Explorar, transformar y visualizar 75
1. Introducción 75
2. Explorar 78
   2.1. Exploración general 79
   2.2. Exploración específica 82
3. Transformar 87
   3.1. Filtrar 90
   3.2. Ordenar 94
   3.3. Seleccionar 96
   3.4. Mutar 98

3.5. Resumir 101
3.6. Agrupar 104
3.7. Recapitulando dplyr 106
4. Visualizar 107
4.1. Las primeras capas 111
4.2. Otras capas 119
4.3. Combinando dplyr y ggplot2 124
Ejercicios de autoevaluación 126
Solucionario 130
Glosario 131
Bibliografía 133

Capítulo III. Análisis univariante 135
1. Introducción 135
2. Representaciones gráficas 136
2.1. Histograma 137
2.2. Diagrama de líneas 143
2.3. Diagrama de barras 146
2.4. Diagrama de cajas 152
2.5. Diagrama de dispersión 155
2.6. Retoques finales 158
3. Medidas de análisis univariante 161
3.1. Tabla de frecuencias 162
3.2. Medidas de centralidad 164
3.3. Medidas de dispersión 173
Ejercicios de autoevaluación 180
Solucionario 184
Glosario 185
Bibliografía 186
Anexo 187

Capítulo IV. Análisis bivariante 189
1. Introducción 189
2. Relación entre variables categóricas 193
   2.1. Visualización con tabla 194
   2.2. Visualización gráfica 197
   2.3. Cuantificar 199
3. Relación entre variable categórica y numérica 205
   3.1. Visualizar 207
   3.2. Cuantificar 211
4. Relación entre variables numéricas 214
   4.1. Visualizar 216
   4.2. Cuantificar 220
   4.3. Modelar 226
5. Regresión múltiple 232
   5.1. Cuantificar 233
   5.2. Caso de estudio 235

Ejercicios de autoevaluación 239
Solucionario 242
Glosario 243
Bibliografía 244
Anexo 245

Capítulo V. Importar, limpiar, unir 251
1. Introducción 251
2. Importar datos 252
   2.1. Importar un paquete 253
   2.2. Importar un archivo 257
   2.3. Guardar el marco de datos 265
3. Limpiar bases de datos 266
   3.1. Limpiar con tidyr 267
   3.2. Convertir variables 274

3.3. Eliminar filas y columnas 275
3.4. Anomalías en los datos 276
4. Unir marcos de datos 282
Ejercicios de autoevaluación 291
Solucionario 293
Bibliografía 294
Anexo 295

Conclusión 297
Bibliografía 299

Introducción

En las últimas décadas, las nuevas tecnologías han permitido una proliferación de bases de datos internacionales en el ámbito de las Ciencias Sociales, producidas tanto por organizaciones intergubernamentales como por centros de investigación y universidades. Ejemplo de ello son los repositorios de datos socioeconómicos del Banco Mundial y del Fondo Monetario Internacional, las bases de datos de democracia como Freedom House y Varieties of Democracy (V-Dem), o las de relaciones entre países como Correlates of War (CoW). Esta cantidad creciente de información cuantitativa disponible sobre temas sociales complejos en ámbitos internacionales lleva a la necesidad de formar a los estudiantes en programas de gestión de datos que les permita tratarla y analizarla. Tal es el propósito de este libro.

En esta obra, el estudiante aprenderá, de manera práctica, a utilizar un programa de gestión de datos y conocerá y experimentará con distintas bases de datos, como las del Banco Mundial, la UN Votes o la Democracy-Dicatorship (DD). El entorno de datos que se utilizará durante el libro es R, un programa libre especialmente popular en Ciencias Sociales y con una comunidad de usuarios muy activa que permite resolver las dudas de funcionamiento durante el aprendizaje. R y el programa derivado que utilizaremos, RStudio, trabaja con código.[1] A pesar de que esto

1. El programa R se puede descargar en https://cran.r-project.org/ y RStudio en https://rstudio.com/. Ambos deben estar instalados, aunque utilizaremos solo RStudio. Para más información y ayuda, consultad Stack Overflow (https://stackoverflow.com/), la comunidad de usuarios de RStudio (https://

puede representar un obstáculo inicial para el aprendizaje, utilizar un lenguaje de código comporta grandes beneficios para el estudiante y su investigación.[2]

El primer capítulo del libro muestra el lenguaje de programación básico para iniciarse en el análisis de datos con R y cómo construir pequeñas bases de datos con distintas variables y observaciones. La estructura del capítulo está pensada para que el estudiante aprenda primero a realizar operaciones sencillas y acabe siendo capaz de moverse con relativa facilidad entre bases de datos de tamaño considerable. En el segundo capítulo, se aprende a explorar un marco de datos, a identificar sus características principales y a transformar los datos para poder visualizarlos posteriormente. En este proceso, utilizaremos los conocidos paquetes de R dplyr y ggplot2. El tercer capítulo muestra las principales técnicas de análisis univariante, mientras que en el cuarto se aprenden las principales técnicas de análisis bivariante y multivariante. Finalmente, el último capítulo muestra un compendio de bases de datos de ámbito internacional en Ciencias Sociales y ofrece distintas herramientas para importar y limpiar

community.rstudio.com/) o la de DataCamp (https://www.datacamp.com/community/), donde podéis acceder a tutoriales y guías del programa.

2. ¿Por qué trabajar con código? Imaginémonos que descargamos una base de datos de internet, hacemos un análisis de datos y presentamos estos datos en público con unas diapositivas. ¿Qué ha pasado en el ínterin? Nadie sabe qué hemos hecho con los datos en nuestro pequeño laboratorio. Se asume que hemos sido honestos y no hemos manipulado los datos de forma sospechosa para que cuadre lo que queríamos demostrar. El código permite tener un registro de todos nuestros pasos, lo cual facilita la reproducción de la investigación. Todo lo que hemos generado a partir de unos datos podemos transferirlo a otras personas para que puedan obtener los mismos resultados a los que nosotros hemos llegado. Esto mejora la transparencia de nuestros análisis y el rigor de nuestros resultados.

las bases de datos, entre ellas la que permite unir en un mismo marco de datos variables procedentes de distintas bases de datos.

Se asume que el estudiante tiene un dominio mínimo inicial de R: se sabe orientar dentro del programa, conoce la utilidad de cada elemento de la interfaz y sabe cómo reproducir un código. Las líneas de código que aparecen en las páginas siguientes están pensadas para ser copiadas y pegadas a RStudio para que el estudiante pueda reproducirlas en el programa.[3] Normalmente, el código se muestra dentro de una caja gris. Cuando dentro de dicha caja aparecen encabezados con negrita y el símbolo **>**, significa que dentro de la misma caja mostramos imprimido el resultado del código. Es posible que los resultados que obtenga el lector al aplicar los ejemplos que se muestran en este libro varíen respecto a los resultados que se ofrecen, ya que los datos pueden proceder de bases de datos actualizadas. Para terminar, una aclaración terminológica: nos referimos a una *base de datos* como una ubicación genérica para almacenar datos. En cambio, nos referimos a un *marco de datos* como un objeto específico que R utiliza para almacenar datos.

3. Para cualquier duda en la aplicación de los códigos, puede consultarse la sección de cursos en la página web del autor (www.jordimas.cat/courses).

Capítulo I
El lenguaje de R

1. Introducción

Para aprender a utilizar programas de análisis de datos como R, lo más importante que nos hace falta inicialmente es dominar el lenguaje. En general, conocer un lenguaje de programación equivale a ser competente en un idioma. Si compartimos el mismo idioma, podremos comunicarnos con el programa y hacer que este entienda nuestras órdenes. Pasa lo mismo cuando queremos aprender alemán: tendremos que dominar la gramática alemana, sus palabras básicas y saber cómo se conjugan los verbos. Además, tarde o temprano, tendremos que ser capaces de defendernos oralmente y tener una buena pronunciación.

Con lenguajes de programación como R, la suerte que tenemos es que son lenguajes escritos y no orales. Por lo tanto, esto nos ahorra una parte importante del aprendizaje, puesto que solo tendremos que aprender la parte escrita. El entorno en el que se utiliza el lenguaje de R es también muy limitado: no tenemos que usarlo ni en el aeropuerto ni en el supermercado, ni para presentarnos a otras personas, tal como pasa en la mayoría de los idiomas, que tenemos que aprender a usarlos en una gran cantidad de contextos. Estos lenguajes solo son necesarios en situaciones muy concretas, de modo que no tenemos que conocer muchas palabras puesto que constantemente repetimos las mismas ideas, por ejemplo:

«filtra las columnas de una base de datos»

«construye un gráfico a partir de estas variables»
«suma estos dos valores»
«di cuál es la media de los valores de esta variable»...

Si sabemos denominar los principales objetos de los que disponemos (*variable, base de datos, valor*...) y los principales verbos que utilizaremos (*filtrar, ordenar, crear gráfico*), y tenemos los conocimientos gramaticales necesarios para construir frases con sentido, R nos entenderá y podremos hacer maravillas con los datos.

¿Cómo le hablamos, pues, a R? Siempre solemos decir «[R] haz esto», «[R] haz aquello». Por lo tanto, todas las órdenes ya llevan implícitas el sujeto, puesto que es R quien lo hace, lo cual es otra ventaja. Si, en la mayoría de idiomas, las frases suelen tener sujeto y predicado, en la gramática de R no nos hará falta sujeto porque R ya entiende quién tiene que hacer la acción. Lo que sí nos hace falta es el predicado, razón por la que en el lenguaje de R hay sobre todo verbos, adverbios y complementos del verbo:

1) a los verbos los denominaremos *funciones*,
2) a los adverbios los denominaremos *argumentos*,
3) a los complementos del verbo los denominaremos *objetos*, y
4) a la información complementaria asociada a los objetos la denominaremos *atributos*.

2. La estructura de datos de R

Para el propósito del análisis de datos, deberemos conocer principalmente cuatro elementos fundamentales de la estructura de datos de R. A continuación, detallamos, en primer lugar, los

objetos y los atributos y, en segundo lugar, las funciones y los argumentos.

2.1. Objetos y atributos

Por *objeto* nos referimos a cualquier dato que tengamos guardado dentro de R, mientras que por *atributo* entendemos la información complementaria asociada a estos datos.

Un objeto de R[1] puede tomar la forma de un número, de una cadena de valores o de un marco de datos, entre otras. Para crear un objeto tenemos que usar el símbolo `<-`, donde, en primer lugar, ponemos el nombre que tendrá el objeto, seguido de `<-` y, finalmente, la forma del objeto. A continuación, hemos generado varios códigos para crear objetos. Primero, hemos creado el objeto `tres`, que está formado por el número tres. En segundo lugar, hemos pedido a R que nos guarde, con el nombre de `operacion`, el resultado de `(6 + 4) / 2`. Lo mismo le pedimos con `operacion_nueva`, donde nos multiplica por tres el objeto `operacion` que hemos creado hace un momento. El cuarto código y el quinto nos muestran objetos algo más sofisticados, que estudiaremos enseguida. El objeto `paises` contiene un tipo de objeto llamado *vector*, de longitud cinco, puesto que está formado por una cadena de cinco

1. Hay otros tipos de objetos que no veremos. Los más comunes son la lista, la matriz (*matrix*) o las variables indexadas (*array*).

valores que contiene nombres de varios países. Hay varios tipos de vectores. A este lo denominaremos *vector de carácter*, ya que en lugar de guardar números como en los ejemplos anteriores, guarda caracteres, que siempre irán separados por comillas. Finalmente, el último objeto que vemos es un marco de datos (*data frame*), que hemos denominado `md_hdi`. El marco de datos se caracteriza por que agrupa varios vectores de la misma longitud. Para conocer la longitud de un vector tenemos que utilizar la función `length()`. Por ejemplo, `length(paises)` nos mostrará la longitud del vector `paises` que acabamos de crear. El primer vector `pais` es un vector de carácter, el segundo y el tercero, `pnb` y `e_vida`, son vectores numéricos, mientras que el último vector `dem` es un vector lógico, que puede adoptar los valores verdadero (`TRUE`) o falso (`FALSE`).

```
tres <- 3
operacion <- (6 + 4) / 2
operacion_nueva <- operacion * 3
paises <- c("Alemania", "Argentina", "España", "Marruecos",
"Sudan")
md_hdi <- data.frame(pais = c("Alemania", "Argentina", "España",
"Marruecos", "Sudan"),
pnb = c(36.2, 15.5, 28.3, 10.2, 2.8),
e_vida = c(78, 73, 79, 67, 54),
dem = c(TRUE, TRUE, TRUE, FALSE, FALSE))
```

Los vectores y los marcos de datos serán los dos principales objetos que utilizaremos con R. Como observamos en el marco de datos `md_hdi` que acabamos de crear, todos los vectores que lo forman tienen la misma longitud. En este caso, es un marco de datos formado por cuatro columnas (el número de vectores) y cinco observaciones (la longitud de los vectores).

Antes de repasar más a fondo estos dos tipos de objetos, tenemos que hacer dos consideraciones relacionadas con su creación:

1) Debemos tener en cuenta que, cuando creamos un objeto, la consola no nos dará ninguna señal de que lo hayamos creado, puesto que lo único que hacemos es almacenar el objeto en la memoria. Si queremos visualizar el objeto una vez creado, lo que tenemos que hacer es teclear su nombre. Si, por el contrario, queremos visualizar el objeto a la vez que lo creamos, podemos poner toda la línea de mando entre paréntesis. En este caso, R no solo nos creará el objeto `tres`, sino que también nos reproducirá su contenido en la consola.

Visualizar objetos en la consola

Si tecleamos `md_hdi` después de guardarlo con este nombre, podremos ver el objeto imprimido en la consola. Para visualizar el objeto a la vez que lo creamos, lo tenemos que poner entre paréntesis, como por ejemplo `(tres <- 3)`. Muchas veces utilizaremos el término *imprimir el objeto*, que significará teclear el nombre para visualizarlo en la consola.

Poner nombre a un objeto

A un objeto le podemos dar casi casi cualquier nombre. Las únicas limitaciones son que no puede empezar con una cifra (por ejemplo, `1objeto <- 34`) ni puede contener ninguno de los símbolos siguientes: ^, !, $, @, +, -, /, *. Si denominamos un objeto con el nombre de otro objeto creado previamente, R nos sobrescribirá el objeto sin avisarnos antes. R es sensible a las minúsculas y a las mayúsculas, de modo que entenderá `Casa` y `casa` como dos objetos diferentes.

Recomendamos utilizar minúsculas siempre que se pueda y usar la barra baja en caso de querer separar palabras para denominar un mismo objeto.

2) Tenemos que saber que cuando guardamos un objeto este nos aparecerá en el panel Environment de RStudio acompañado de una descripción breve. Podemos visualizar los objetos directamente en el panel Environment o bien también podemos consultar un listado de los objetos creados tecleando indistintamente las funciones `ls()` u `objects()`.

2.1.1. Vectores: objetos de una dimensión

Un *vector* es una cadena de valores, ordenados en una sola dimensión, que puede tener una longitud diversa, desde un solo valor hasta miles de valores.

Anteriormente hemos creado un pequeño marco de datos, `md_hdi`, en que cada vector es una variable diferente del marco de datos. La mayoría de objetos de R se organizan a partir de vectores, y, por lo tanto, tener muy claro qué es un vector, qué función hace y qué tipos de vectores podemos crear con R nos facilitará mucho el trabajo como analistas.

El *vector* es la estructura básica de R y la forma que en R toma una variable dentro de un marco de datos. En la tabla 1 hemos creado varios vectores y hemos asignado un nombre diferente a cada uno de ellos. Los vectores pueden almacenar hasta seis tipos

diferentes de datos, aunque en este solo utilizaremos cuatro: numérico, entero, carácter y lógico.

Tabla 1. Tipos de datos que puede almacenar un vector

Numérico o doble	`vector_numerico <- c(78.2, 56.3, 72.4, 64.6, 84.1)`
Entero	`vector_entero <- c(1L, 5L, 7L, 4L, 4L, 4L, 7L, 8L)`
Carácter o *string*	`vector_caracter <- c("azul", "amarillo", "verde", "azul")`
Lógico	`vector_logico <- c(TRUE, FALSE, FALSE, FALSE, TRUE)`

Fuente: elaboración propia.

La L del vector entero

Aunque introducimos una L mayúscula cuando creamos un vector entero, R no nos visualizará la L. Solo entenderá que este vector lo tiene que guardar como entero. Una opción más fácil para crear un vector entero es crear un vector numérico sin decimales de la manera siguiente: `as.integer(c(3, 5, 1, 7))`.

Para crear un vector que contenga más de un valor, pondremos los valores entre paréntesis encabezados por la función `c()`, que es una abreviación de *concatenado*. Dentro de la función, introduciremos los valores correspondientes según el tipo de vector que queramos crear. El vector numérico y el entero tienen una apariencia muy parecida. Los dos almacenan números, pero, mientras que el vector numérico acepta decimales, el entero (*integer,* en inglés) solo acepta números enteros. Por defecto, R nos almacenará cualquier número como vector numérico, tenga

decimales o no. Si queremos que nos lo guarde como entero, lo tendremos que especificar poniendo una L mayúscula delante o transformando el vector numérico en entero. Ahora no podemos apreciar muy bien la diferencia práctica entre numérico y entero, pero, más adelante en este módulo, veremos que esta distinción nos va a ser muy útil.

El vector de carácter, que veremos muy a menudo con el nombre de *string*, almacena texto. Este texto tiene que ir siempre entre comillas y dentro de las comillas podemos guardar letras, números, caracteres especiales, etc. Finalmente, el vector lógico (también llamado *vector booleano*) nos guarda valores que pueden ser verdaderos o falsos. Los valores que son verdaderos los almacenamos como `TRUE` o sencillamente con una `T`, mientras que los que son falsos los guardaremos como `FALSE` o con una `F` (siempre indicado en letras mayúsculas). Es importante saber que en un mismo vector no podemos mezclar vectores que tengan diferentes tipos de datos. Si lo hacemos, R guardará los datos con el tipo de vector que permita conservar la mayor cantidad de información posible.

Vectores de carácter que no lo parecen

Un valor en un vector de carácter podría ser `"22"` o bien `"•=%/"`. Solo vale que vaya indicado entre comillas.

Ejercicio 1. Coerción de los datos

Intentad hacer el ejercicio siguiente para averiguar cómo R elige el vector con el que guardará la información en caso de que se hayan introducido varios tipos de valores. Primero, almacenad cada uno de

los vectores y después mirad qué tipo de vector habéis creado con la función `class()`.

```
num_log <- c(34, TRUE)
car_log <- c("Hello", TRUE)
num_car <- c(34, "Hello")
num_car_log <- c(34, "Hello", TRUE)
```

¿Qué descubrimos con este ejercicio? Si intentamos poner diferentes tipos de datos en un solo vector, R convertirá los elementos del vector en valores de un solo tipo. ¿Cómo decide R esta conversión? R siempre intentará preservar el máximo de datos posible, y la manera de perder menos información es en un vector de carácter. Dentro de las comillas podemos guardar todo lo que queramos. En cambio, en un vector numérico no podemos almacenar letras. Los caracteres, pues, siempre tendrán prioridad sobre los otros tipos de vector en caso de conflicto. Entre numéricos y lógicos, R guardará un vector numérico, puesto que siempre podemos interpretar `TRUE` como 1 y `FALSE` como 0. Entre numéricos y enteros, R guardará numéricos.

Para trabajar con vectores, sean del tipo que sean, nos será muy útil aprender a seleccionar solo una parte de sus elementos. La manera de seleccionarlos es mediante corchetes `[ ]` después del nombre del vector. Dentro de los corchetes, indicaremos la posición de los valores que queremos seleccionar. El símbolo `:` nos servirá para indicar una selección en cadena, en la que a la izquierda del símbolo pondremos la posición del primer valor y a la derecha, la del último valor. El símbolo negativo excluye una selección. Siguiendo esta lógica, en la tabla 2 podemos ver ejemplos de cómo funciona la selección de elementos de un vector hipotético.

Tabla 2. Selección de elementos de un vector

`vector[1:3]`	Del primer al tercer valor
`vector[c(2,4)]`	El segundo y el cuarto valor del vector
`vector[c(1:3,6)]`	Del primer al tercer valor y el sexto valor del vector
`vector[-c(5,8)]`	Todos los valores del vector menos el quinto y el octavo

Fuente: elaboración propia.

Hay que tener en cuenta que las operaciones que hacemos sobre los vectores se aplicarán a todos los valores del vector. Si, en cambio, multiplicamos vectores de igual longitud, multiplicará el primer valor del primer vector por el primer valor del segundo vector, el segundo valor del primer vector por el segundo valor del segundo vector, y así sucesivamente. Si hacemos operaciones con vectores de longitud diferente a 1 y a la longitud del vector, nos dará error.

Ejemplo

Si multiplicamos el número 3 por el vector `c(3,4,5)` nos multiplicará el número 3 por cada uno de los números de dentro del vector y dará como resultado 9, 12 y 15. En cambio, si multiplicamos `c(3,4,5)` por `c(3,4,5)` dará 9, 16 y 25.

2.1.2. Marcos de datos: objetos de dos dimensiones

Un *marco de datos* es un conjunto de vectores de igual longitud agrupados en una tabla de dos dimensiones formada por columnas y filas. En la dimensión de las filas, cada elemento representa una observación de los datos. En la dimensión de las columnas, cada columna está formada por un vector.

El marco de datos es la estructura principal que usaremos para el análisis de datos. Antes de iniciarnos en R, habremos visto estructuras parecidas en hojas de cálculo de Excel o en otros programas informáticos. Solemos decir *tabla*, *base de datos*, *cuadro*... Sin embargo, en R diremos *marco de datos*, entendido como un conjunto de vectores de igual longitud agrupados en una tabla de dos dimensiones formada por columnas y filas. Como ya sabéis, cada vector solo puede incluir un único tipo de datos (numéricos, lógicos, caracteres...); en cambio, un marco de datos puede incluir vectores de diferentes tipos.

Para crear un marco de datos tenemos que utilizar la función `data.frame()` e introducir los vectores que queremos que incorpore, separados por comas, en el orden en que los visualizaremos, de izquierda a derecha. Antes de cada vector, podemos especificar el nombre que tendrá la columna seguido del símbolo igual. En la tabla 3, tenemos representado el marco de datos `md_agr` y, en la parte inferior, el código que hemos utilizado para crearlo.

Crear un marco de datos

Dentro de la función `data.frame()` hemos creado primero el vector `pais`. Al repetir dos observaciones para cada país, una en 1980 y la otra en 2010, tenemos que repetir dos veces el nombre del mismo país. Recuerda que tienes que poner primero el nombre de la columna, seguido de un igual y un concatenado con los valores correspondientes, y, sobre todo, que para pasar al vector siguiente hay que cerrar paréntesis después de introducir todos los valores de cada vector.

Tabla 3. Creación de un marco de datos

país	año	dem	pib_cap	agr
<carácter>	<numérico>	<lógico>	<numérico>	<numérico>
"Francia"	1980	TRUE	12672.18	56.8
"Francia"	2010	TRUE	40638.33	52.8
"Reino Unido"	1980	TRUE	10032.06	76.8
"Reino Unido"	2010	TRUE	38893.02	71.2
"Polonia"	1980	FALSE	5241.75	64.5
"Polonia"	2010	TRUE	12597.86	47.2
"Congo"	1980	FALSE	927.10	30.8
"Congo"	2010	FALSE	2737.34	31

```
md_agr <- data.frame(pais = c("Francia", "Francia",
                              "Reino Unido", "Reino Unido",
                              "Polonia", "Polonia", "Congo",
                              "Congo"),

                     ano = c(1980, 2010, 1980, 2010, 1980, 2010,
                     1980, 2010),
```

```
                    dem = c(TRUE, TRUE, TRUE, TRUE, FALSE,
                    TRUE, FALSE, FALSE),

                    pib_cap = c(12672.18, 40638.33, 10032.06,
                    38893.02, 5241.75, 12597.86, 927.10,
                    2737.34),

                    agr = c(56.8, 52.8, 76.8, 71.2, 64.5, 47.2,
                    30.8, 31), stringsAsFactors = FALSE)
```

Fuente: elaboración propia.

Si imprimimos el marco de datos que acabamos de crear tecleando `md_agr`, veremos que consta de cinco columnas correspondientes a cada uno de los vectores que hemos creado. El vector `pais` de la primera columna es de carácter, puesto que R ha entendido que, si los valores están entre comillas, tendrá que crear un vector que será de carácter.[2] R ha interpretado los vectores `año`, `pib_cap` y `agr` (que representa el porcentaje de tierra cultivable) como variables numéricas, mientras que ha interpretado el vector `dem` (democracia) como vector lógico. Podemos ver algunas características adicionales del marco de datos si pedimos la estructura con `str(md_agr)`. Esta función nos permite observar que se trata de un marco de datos con cinco variables o vectores, y ocho observaciones. También podemos comprobar cuál es la tipología de cada variable o vector (*chr* por carácter, *num* por numérico y *logi* por lógico).

```
> str(md_agr)

'data.frame': 8 obs. of 5 variables:
```

2. El primer vector también es de carácter porque hemos introducido el argumento `stringsAsFactors = FALSE` al final del código. Siempre tendremos que poner este argumento si queremos que nos mantenga los vectores entre comillas como caracteres. Más adelante en este módulo veremos la diferencia entre *strings* y factores.

```
$ pais     :  chr  "Francia"  "Francia"  "Reino  Unido"  "Reino
           Unido" ...
$ año      : num 1980 2010 1980 2010 1980 2010 1980 2010
$ dem      : logi TRUE TRUE TRUE TRUE FALSE TRUE ...
$ pib_cap  : num 12672 40638 10032 38893 5242 ...
$ agr      : num 56.8 52.8 76.8 71.2 64.5 47.2 30.8 31
```

A la hora de trabajar con los marcos de datos de R, debemos tener en cuenta dos consideraciones importantes. Si os fijáis, cuando imprimimos el marco de datos, las observaciones están en las filas y los vectores en las columnas. Sin embargo, cuando imprimimos la estructura con `str()`, lo encontramos a la inversa: los nombres del vector están en las filas, mientras que las observaciones están desplegadas en horizontal. Esto es así por cuestiones prácticas: normalmente trabajaremos con marcos de datos enormes, con miles de filas, pero, en cambio, con pocos vectores. Por esto, nos resulta más cómodo imprimir la estructura para hacer una visualización rápida en la consola, dar un vistazo a los nombres de las variables y su tipo, y ver una muestra de los datos que contiene la variable. Para visualizarlo, es más intuitivo desplazarse por la consola en vertical que en horizontal. La segunda consideración, relacionada con la primera, es que, a diferencia de otros programas informáticos, nos tendremos que acostumbrar a trabajar con los datos, pero sin los datos. Es decir, el marco de datos estará en nuestra cabeza, pero no en la pantalla. Como analistas de datos veréis que acabará siendo una manera más eficiente de trabajar, puesto que no hay ninguna necesidad de tener los datos visualmente disponibles cuando trabajamos con bases de datos de grandes dimensiones.

Visualización clásica de una tabla

La función `View()` nos permite visualizar todo un marco de datos a la manera clásica, pero avisamos que en pocos días no la echaremos de menos. En este módulo aprenderemos a trabajar sin necesidad de tener disponibles todos los datos en la pantalla.

Tibble: la nueva generación de marcos de datos

Un *tibble* es un tipo especial de marco de datos, que ofrece, entre otras ventajas, mejores visualizaciones en la consola que un marco de datos normal. El *tibble* adapta el marco de datos a la consola, de modo que según la anchura que tenga la pantalla de nuestro ordenador podremos visualizar más o menos observaciones. Todo lo que no nos pueda enseñar nos lo mostrará de manera resumida al final (filas, variables y tipos que faltan). En definitiva, tener un marco de datos en formato *tibble* es preferible a un marco de datos normal. Podemos probar a cambiar el marco de datos `md` a *tibble,* indistintamente, mediante la función `tibble()` o `tbl_df()` que se encuentra dentro del paquete de dplyr . La función `as.data.frame()` hace la operación inversa y cambia el *tibble* a un marco de datos normal.

```
md <- tbl_df(md)
```

Otra de las grandes ventajas de *tibble* es que nos permite crear marcos de datos de manera horizontal con la función `tribble()`. Como vemos en el ejemplo siguiente, con *tribble* podemos introducir los marcos de datos por filas en lugar de por columnas, de modo que nos permite visualizar el código más intuitivamente.

```
tribble(~country,       ~year,    ~number,
        "Argentina",    1980,     2304,
        "Brazil",       1990,     2045)
```

Para seleccionar solo una parte del marco de datos, también nos pueden ayudar los corchetes. La única diferencia en relación con la selección de vectores que hemos visto anteriormente es que, en el caso de los marcos de datos, hay una coma dentro de los corchetes que separa las indicaciones de la posición de las filas de las indicaciones de la posición de las columnas que queremos seleccionar (por ejemplo, `md[filas,columnas]`). Si dejamos un lado de la coma en blanco, R entenderá que queremos conservar todas las filas o las columnas.

Si introducimos un símbolo negativo, R entenderá que queremos toda la selección excepto el valor o valores indicados. Ilustramos estas características en la tabla 4.

Tabla 4. Selección de filas y columnas de un marco de datos

`md[3, ]`	La tercera fila y todas las columnas
`md[2:4,c(1,4)]`	De la segunda a la cuarta fila y la primera y cuarta columna
`md[ ,-4]`	Todas las filas y todas las columnas excepto la cuarta.
`md[-c(2:4,6), "pib"]`	Todas las filas excepto de la segunda a la cuarta y la sexta, la columna con nombre «pib». Nos devolverá un vector.

Fuente: elaboración propia.

Todo este abanico de objetos que hemos estudiado hasta ahora no solamente tiene almacenado en su interior los datos propiamente dichos, sino que también tiene ligada información

adicional. A esta información la llamamos *atributos*, que podemos conocer por medio de la función `attributes()`. Si cuando pedimos los atributos a la consola R nos devuelve `NULL`, querrá decir que el objeto no tiene ningún atributo. Si tiene alguno, R nos devolverá una lista con diferentes vectores para cada tipo de atributo.

Observemos, por ejemplo, los atributos del marco de datos `md_agr`. En primer lugar, vemos que los nombres de las columnas son atributos. Es decir, no forman parte directamente de los valores del marco de datos, pero sí que son información adicional sobre el marco. Los nombres de las columnas también los podemos obtener con la función `names()`, que nos será de gran utilidad cuando queramos cambiar los nombres de columna. Como muestra el código siguiente, podemos cambiar todos los nombres de columna del marco de datos `md_agr` si creamos un vector de la misma longitud que la cantidad de columnas y lo insertamos en los nombres del marco de datos. Si solo nos interesa cambiar una sola columna, podemos seleccionar el número de columna con los corchetes y hacer la misma operación.

```
names(md_agr) <- c("País", "Año", "Dem", "PIB_cap", "Agr")
names(md_agr)[3] <- c("Democracia")
```

Otros atributos que encontraremos ligados a los objetos son la clase de objeto, que podemos obtener directamente con la función `class()`. Esta función es muy útil para determinar el tipo de vector con el que estamos trabajando. La función `typeof()` nos devuelve una descripción más genérica del tipo de objeto. Finalmente, también podemos obtener el nombre de las filas con `row.names()`, aunque es una función que utilizaremos en contadas ocasiones.

2.2. Funciones y argumentos

Las *funciones* son los verbos de R y nos permiten hacer operaciones con sus objetos.

Ya hemos podido ver algunas funciones en el apartado anterior, como `data.frame()` o `str()`, que hacen acciones como crear un marco de datos o visualizar la estructura del marco. R tiene miles de funciones que hacen operaciones muy diversas: desde tareas sencillas como redondear un número con decimales (`round()`), sumar los valores de un vector (`sum()`) o decirnos cuál es el valor más alto de un vector (`max()`). Utilizar este tipo de funciones es muy sencillo. Solo hay que poner dentro del paréntesis el objeto sobre el cual queremos ejecutar la función. Mediante estas funciones, en el primer caso hemos redondeado los números del `vector_numerico` que hemos creado anteriormente; en el segundo caso, hemos sumado los números de un vector, y, en el último, hemos buscado el número máximo del `vector_entero`.

```
round(vector_numerico)
sum(c(5, 1, 9, 8, 2))
max(vector_entero)
```

Ejercicio 2. Funciones

Con las funciones que conocéis, intentad redondear la división de 17 entre 3. Haced la suma de una cadena de vectores entre el 1 y el 10.

Buscad el número más alto del `vector_numerico`. Finalmente, primero imprimid en la consola el número `pi` que R ya tiene guardado por defecto y después redondeadlo.

Las funciones también pueden operar dentro de otras funciones. Es decir, la función operará a partir del resultado de una función que ha operado previamente. Pongamos por caso que introducimos el siguiente código: `round(sum(vector_numerico))`. R primero nos sumará todos los valores del vector numérico y, a continuación, nos redondeará el resultado, sacándole los decimales.

Dentro de las funciones hay argumentos, que serían los complementos del verbo. Un *argumento* es un elemento que la función necesita para desarrollar una acción.

Las funciones pueden tener más de un argumento y siempre irán separadas por comas dentro del paréntesis de la función. En muchos casos, solo necesitaremos un argumento para hacer la acción. En las funciones del cuadro anterior, por ejemplo, ha bastado con situar un objeto dentro del paréntesis. Sin embargo, en otras ocasiones tendremos que incluir más de un argumento para hacer una acción. Fijémonos en la función `sample()`, que nos permite obtener una muestra aleatoria de un conjunto de datos. Si el único argumento que introducimos dentro de la función es un número, R nos devolverá aleatoriamente un vector de números enteros diferentes con longitud igual al número que hemos indicado. Si introducimos un vector como `vector_numerico`,

la función nos devolverá los elementos del vector ordenados de manera aleatoria.

Ejercicio 3. La función `sample()`

Intentad repetir las funciones del cuadro más de una vez. Veréis que cada vez que se ejecuta la acción el programa devuelve los datos con orden diferente.

```
sample(10)
sample(vector_numerico)
sample(10, 3)
sample(vector_numerico, 2)
```

Intentemos ahora introducir un segundo argumento. En el tercer ejemplo del cuadro, le decimos a R que tome los números enteros del 1 al 10 y seleccione solo tres aleatoriamente. Lo mismo podemos hacer con los valores de `vector_numerico`. R nos devolverá dos valores al azar del vector.

Cada función, pues, puede tener varios argumentos. Normalmente, como en el caso de `sample()`, el primer argumento será el objeto sobre el cual queremos que se aplique la función. Otros muchos argumentos pueden funcionar de manera implícita, puesto que cada función acostumbra a estar diseñada con varios argumentos por defecto, que operan sin que los veamos. Por ejemplo, `sample()` tiene como segundo argumento implícito que nos devuelva un vector de longitud igual al objeto del primer argumento.

Cada función tiene sus propios argumentos y, por lo tanto, para poder sacar el máximo rendimiento de una función, hay que cono-

cer la estructura interna. La manera más directa de conocer los argumentos que tiene una función determinada es aplicar `args()`.

```
> args(sample)
function (x, size, replace = FALSE, prob = NULL)
```

Si queremos conocer los argumentos de un manera más extensa, podemos poner el signo de interrogación `?` antes de la función o bien podemos utilizar la función `help()`.[3] Ambas opciones abren una ficha de información sobre la función en la pestaña de ayuda situada en la parte inferior derecha de RStudio.

En la tabla 5 no reproducimos toda la información que aparece en la ficha de información, pero sí que describimos los elementos más relevantes.

Tabla 5. Cuadro de ayuda resumido de la función `sample`

> ?sample() **> help(sample)**	
Usage	
`sample(x, size, replace = FALSE, prob = NULL)`	
Arguments	
`x`	Vector de uno o más elementos
`size`	Número no negativo de ítems que seleccionará de la muestra
`replace`	Si vuelve a incluir en la muestra los números que ya han salido
`prob`	Vector de probabiblidad que indica los pesos del vector `x`
Por defecto, `size` tiene la misma longitud que `x`	

Fuente: elaboración propia.

3. También tenemos el doble interrogante `??`, que, situado delante de cualquier nombre, nos hace una búsqueda por la palabra clave que indicamos y nos reproduce un listado en el cuadro de ayuda con los resultados más relevantes que ha encontrado.

Utiliza constantemente el cuadro de ayuda

Usar el cuadro de ayuda es muy útil para obtener más información o para ayudarnos a desencallar algún problema que podamos tener. Es imposible memorizar todos los argumentos que tiene cada función, por lo cual nos será muy útil consultar el cuadro de ayuda cada vez que necesitemos refrescar cuáles son los elementos principales de una función determinada. Por eso es muy importante prestar atención y entender las fichas de ayuda.

Como vemos en la tabla, el apartado Usage nos muestra los argumentos principales de la función `sample()`: *x*, *size*, *replace* y *prob*. Si queremos saber más detalles, podemos ir al apartado Argumentos, que nos ofrece una descripción de cada argumento. En *x* tenemos que poner el objeto en cuestión y en *size*, el número no negativo de ítems que seleccionará de la muestra. Fijémonos en que estos dos elementos no van acompañados en Usage del símbolo = y otro parámetro, pero sí que van acompañados los argumentos *replace* y *prob*. Esto quiere decir que *replace* y *prob* tienen asignado un valor por defecto. En otras palabras, si no indicamos lo contrario, R entenderá que *replace* es falso (por lo tanto, no volverá a incluir el número en el bombo una vez que haya salido) y que *prob* es nulo (todos los elementos tienen la misma probabilidad de salir seleccionados). En *Details,* vemos que *size* también tiene un valor asignado por defecto, pero este no es fijo, sino que depende del valor de *x*. Esto quiere decir que en el supuesto de que introduzcamos un solo argumento en la función `sample()`, R nos devolverá tantos valores como elementos haya en el valor en cuestión. En caso de que queramos que nos devuelva un número diferente de valores, lo tendremos que indicar expresamente. Al principio nos costará entender estas

páginas de ayuda,[4] pero poco a poco, a medida que nos vayamos familiarizando con la lógica de las funciones, nos resultarán cada vez más útiles e indispensables.

Mismo nombre de función, diferentes paquetes

Nos podemos encontrar, como en el caso de la función `sample()`, con que varios paquetes tengan una función con el mismo nombre. Para especificar que queremos la función de un paquete concreto, podemos utilizar el símbolo `::`. Por ejemplo, `base::sample()`.

Ejercicio 4. Consultad los argumentos de otras funciones

Intentad investigar las funciones `seq()`, `rep()` y `sort()` mediante el cuadro de ayuda. Intentad construir un código para cada función para que os devuelva a la consola el resultado que se indica a continuación. En las dos últimas funciones tendremos que indicar, como argumento *x*, el objeto `q <- c(3, 9, 5, 6)`, e intentar averiguar el resto de argumentos.

Función	Resultado
`seq()`	`7 12 17 22 27 32`
`rep(x = q)`	`3 3 9 9 5 5 6 6 3 3 9 9`
`sort(x = q)`	`9 6 5 3`

4. Hay otros apartados en la ayuda que no hemos mirado tan detalladamente, como *Description* (breve sumario de lo que hace la función), *Details* (descripción avanzada de cómo está programada la función) o *See also* (nos da la lista de funciones similares a la que estamos inspeccionando). El último apartado, *Examples*, nos puede servir de gran ayuda para hacernos una idea de cómo operan las funciones en cuestión con códigos reales.

Un último aspecto importante que hay que tener en cuenta cuando especificamos los argumentos dentro de una función es el orden con el que los ponemos. R espera que, si no indicamos lo contrario, introduzcamos los argumentos con el mismo orden en el que nos los muestra en el cuadro de ayuda. Es decir, como en la primera línea del código siguiente, nos leerá la función de modo que `vector_numerico` corresponde a la *x*, 2 corresponde a *size* , `TRUE` corresponde a *replace* y el vector que hemos creado, que es igual a la longitud del vector indicado a *x*, señala las probabilidades que tendrá cada elemento de *x* de salir elegido. También podemos poner los nombres de los argumentos como vemos en la segunda línea de código, pero, si seguimos el orden predeterminado, no sería necesario introducirlos. Del mismo modo, también podemos alterar el orden de los argumentos, pero en este caso sí será necesario indicar a R cuál es cada argumento.

```
sample(vector_numerico, 2, TRUE, c(0.3, 0.1, 0.3, 0.2, 0.1))
sample(x = vector_numerico, size = 2, replace = TRUE,
       prob = c(0.3, 0.1, 0.3, 0.2, 0.1))
sample(replace = TRUE, size = 2,
       prob = c(0.3, 0.1, 0.3, 0.2, 0.1), x = vector_numerico)
```

Si siempre indicamos los nombres de los argumentos, acabaremos teniendo códigos más largos, pero tendremos la ventaja de que será más fácil y rápido leer e interpretar los códigos, tanto para nosotros como para terceras personas. Indicar el nombre de los argumentos también ayuda a detectar y prevenir errores.

3. Variables y tipos

Una *variable* es una característica del fenómeno que estudiamos que varía entre los casos y, por lo tanto, puede tomar varios valores.

Supongamos que tenemos una base de datos de todos los países del mundo y estamos recopilando varias características de estos países. ¿Qué podría ser y qué no podría ser una variable en este estudio? Un color cualquiera, el planeta del país, la gravedad o la presencia de oxígeno no serían una variable. Un color cualquier no es ninguna propiedad del fenómeno que estudiamos. Todos los países se encuentran en el mismo planeta, por lo que el valor de planeta siempre será el mismo: la Tierra. Como no habrá variación, planeta no es una variable. En esencia, la gravedad también es la misma entre países y también en todos los países hay presencia de oxígeno. Así pues, tampoco son variables. Sin embargo, sí que serían variables el nombre del país, su extensión territorial, el número de habitantes o la edad media de estos habitantes. Estos elementos son variables porque tendremos países con nombres diferentes, de tamaños territoriales diferentes, más o menos poblados y con unos habitantes más o menos envejecidos.

No todas las variables varían igual, y es por eso por lo que podemos encontrar variables de diferentes tipos. Normalmente, se suele establecer una primera distinción entre variables categóricas y numéricas.

1) Las llamaremos *variables categóricas* si los valores que varían no son números, sino categorías, como el país, la raza de perro o el color del pelo.

2) Las llamaremos *variables numéricas* si los valores que varían son números, como la temperatura o la edad.

Dentro de cada grupo de variables, también podemos hacer otras dos distinciones, tal como vemos en la tabla 6. Denominaremos *variables categóricas nominales* a las variables categóricas, discretas y no ordenables. Denominaremos *variables categóricas ordinales* a las variables categóricas, discretas y ordenables. Entre las variables numéricas, distinguiremos entre las *variables numéricas discretas* y las *variables numéricas continuas*.

Tabla 6. Tipo de variables

Variable	Características	Operaciones
Categórica nominal	Categórica, discreta, no ordenable	`==, !=`
Categórica ordinal	Categórica, discreta, ordenable	`<, <=, >, >=, !=, ==`
Numérica discreta	Numérica, discreta, ordenable	`<, <=, >, >=, !=, ==, +, -, *, /, etc.`
Numérica continua	Numérica, no discreta, ordenable	`<, <=, >, >=, !=, ==, +, -, *, /, etc.`

Fuente: elaboración propia.

¿Qué quiere decir exactamente que las variables sean ordenables o discretas? La ordenabilidad es la diferencia principal entre los dos tipos de variables categóricas. No se pueden ordenar

variables como el sexo o los países,[5] puesto que no hay manera lógica de establecer un orden de categorías que vaya de mayor a menor. Las categóricas nominales almacenan nombres y solo podemos hacer operaciones de igualdad entre estas variables (decir si son iguales o no lo son).

Sin embargo, las categóricas ordinales sí que se pueden ordenar de manera lógica. Según una de las clasificaciones del Banco Mundial, un país puede ser muy pobre, pobre, rico o muy rico. Estas cuatro categorías tienen una manera lógica de ordenarse, de menos a más. Muy pobre sería la categoría más baja y muy rico la más alta. Otra variable ordenable sería la clasificación de potencias mundiales que ordena los países según sí son potencias medianas, grandes potencias, superpotencias, etc. Además de poder decir si estas variables son iguales o no entre ellas, también podremos hacer otras operaciones como determinar si una categoría es superior o inferior a la otra.

Las variables numéricas siempre las podremos ordenar de mayor a menor. Podremos hacer operaciones de igualdad, decir si un valor es superior o inferior a otro y, además, hacer todo tipo de operaciones algebraicas como sumar, dividir o elevar al cuadrado. La diferencia principal entre los dos tipos de variables numéricas radica en si son o no discretas. Que una variable sea discreta no quiere decir que no haga ruido cuando varía. La palabra *discreto* tiene otra acepción que significa que tiene un número finito de categorías. Para saber si una variable es discreta solemos observar si puede contener decimales. Las personas pueden tener

5. No podemos ordenar `"Argentina"` y `"Brasil"` por su nombre para establecer qué valor es superior al otro. Solo son nombres de países y, como máximo, podremos clasificarlos alfabéticamente. Sí que los podríamos ordenar por el PIB, por ejemplo, pero entonces no utilizaríamos la variable *país*, sino la variable *PIB* para ordenarlos.

un hijo o dos, pero no un hijo y medio. El número de hijos de una persona es una variable numérica discreta. También la población de un país o el número de armas nucleares son variables discretas. No podemos tener media arma nuclear. La riqueza de un país, en cambio, medida normalmente por medio del PIB per cápita, puede tener un número infinito de valores puesto que la distancia entre un dólar y dos dólares es grandiosa y puede aceptar todos los decimales que sean necesarios. La temperatura o la esperanza de vida también serían, en este caso, variables numéricas continuas.

Como veremos en este apartado, es muy importante que R sepa con qué tipo de variables tratamos. La manera en que definamos el tipo de variable marcará algunas configuraciones por defecto de R, el tipo de operaciones que podemos hacer y las visualizaciones, y también determinará los métodos estadísticos más apropiados para responder a las preguntas que nos formulamos sobre los datos. Como ya os habréis dado cuenta, una variable en R toma la forma de un vector, de modo que podremos determinar el tipo de variable indicando si es un vector numérico, entero, de carácter o lógico. Para utilizar el vector como variable dentro de un marco de datos, lo tendremos que llamar con el nombre del marco de datos, seguido del símbolo `$` y, finalmente, el nombre de la variable[6] (por ejemplo, `marcodatos$variable`).

6. Podemos hacer la prueba de visualizar algunas variables del marco de datos `md_agr` que hemos creado en el apartado anterior e imprimirlas en forma de vector con `md_agr$pais`, `md_agr$año`, `md_agr$dem`.

3.1. Variables categóricas nominales

Las *variables categóricas nominales*, también denominadas *cualitativas*, son variables discretas no ordenables.

Este tipo de variables son discretas porque toman un número limitado de categorías. Cuando rellenamos un cuestionario, no podemos elegir entre sexos infinitos, solo tenemos las casillas de hombre o mujer. Si nos preguntan por nuestro estado civil, tampoco podemos elegir entre diversos valores. Normalmente oscilarán entre cuatro: soltero/a, casado/a, divorciado/a o viudo/a. En los estudios internacionales, los países, los continentes o el tipo de régimen (democracia o no) son variables categóricas nominales. Además de discretas, estas variables no son ordenables porque no podemos decir si un valor es más alto o más bajo que el resto. Podemos clasificar los países por continente o por país, pero no podemos decir si el hecho de pertenecer a un continente o a un país determinado es menor o mayor. Por lo tanto, con variables categóricas nominales podemos hacer pocas operaciones.[7] No las podemos sumar ni restar. Tampoco podemos decir si una es más grande que otra. Lo único que podemos hacer con ellas es clasificarlas con criterios de igualdad.

Las variables categóricas nominales se pueden almacenar en R como vectores de carácter, como vectores lógicos o como vectores

7. Operaciones con variables categóricas nominales: igual: ==; no igual: !=.

enteros en forma de factores. Normalmente, almacenaremos las variables categóricas como caracteres o como factores, aunque en determinados casos también lo podremos hacer con vectores lógicos.

3.1.1. Carácter

Como sabemos, los vectores de carácter o *strings* almacenan todo tipo de caracteres, desde letras hasta símbolos especiales. Siempre utilizaremos comillas para referirnos a estos vectores. El único tipo de variable que pueden representar los vectores de carácter es la variable categórica nominal. Al no ser ordenables por su valor, R nos ordena los valores por orden alfabético.

Lo más importante que tenemos que saber sobre los vectores de carácter es que hay un paquete en R, denominado *stringr*, que nos puede ayudar a manipularlos con facilidad. A menudo puede ser que queramos cambiar todos los caracteres en letra minúscula o que haya un nombre en particular que queramos denominar de otro modo. Las funciones de este paquete, que tienen la particularidad de empezar todas con *str*, nos pueden ayudar a hacer estos cambios. En la tabla 7 podemos ver algunas de las funciones principales. La sintaxis es muy intuitiva, puesto que en el primer argumento indicamos el nombre del vector, en el segundo los caracteres que seleccionamos y en el tercero, si hace falta, los nuevos caracteres.

Tabla 7. Funciones principales del paquete stringr

`str_detect()`	Devuelve un vector lógico que indica si ha encontrado el conjunto de caracteres que hemos especificado.
`str_replace()`	Cambia en un vector un conjunto de caracteres por otro.
`str_remove()`	Elimina el conjunto de caracteres que indicamos.

`str_to_upper()`	Convierte todos los caracteres en mayúsculas.
`str_to_lower()`	Convierte todos los caracteres en minúsculas.

Fuente: elaboración propia.

Para ilustrar cómo funcionan estas funciones del paquete stringr, podemos crear un marco de datos de nombre `eastg` con un vector de carácter: la columna *country* que tiene las categorías `"East Germany"`, `"Germany"` y `"France"`.[8] Si queremos que `"East Germany"` y `"Germany"` estén en la misma categoría, las tres primeras funciones nos permiten hacerlo mediante procesos diferentes. La función `str_detect()` nos crea vectores lógicos, por los cuales podemos crear el objeto `cambios`, que nos indicará `TRUE` todos los valores del vector que contengan `"East Germany"`. A continuación, utilizaremos los corchetes para seleccionar las columnas de la variable *eastg$country* que sean `TRUE` y le añadiremos un vector con el nombre de `Germany`. Con `str_replace()`, el proceso es mucho más directo, puesto que simplemente introducimos el nombre del vector, el nombre que queremos reemplazar y el nombre nuevo. Finalmente, `str_remove()` también nos permite hacer el mismo proceso, pero en este caso tenemos que indicar qué caracteres tiene que eliminar con `"East"`.

```
cambios <- str_detect(eastg$country, "East Germany")

eastg$country[cambios] <- c("Germany")

str_replace(eastg$country, "East Germany", "Germany")

str_remove(eastg$country, "East")
```

8. Lo crearemos en formato *tibble* con este código: `eastg <- tibble(country = c("East Germany", "Germany", "France"))`. Recordad que debéis tener el paquete dplyr cargado.

Fíjaos que si tuviéramos otras categorías que también empezaran por `"East"` no podríamos utilizar `str_remove()`, puesto que también nos eliminaría los caracteres en estas categorías. Todas estas funciones tienen su utilidad según la situación en la que nos encontremos.

3.1.2. Factores

Un *factor* es un vector entero que almacena información categórica.

Comprender qué es exactamente un factor exige paciencia. Podemos pensar que su naturaleza está entre un vector entero y un vector de carácter. Sobre los vectores enteros, sabemos que no pueden contener decimales y que son ordenables. Sobre los vectores de carácter, sabemos que pueden contener cualquier texto que queramos. Pues bien, un factor es un vector entero camuflado como si fuera un vector de carácter, donde cada número es una categoría diferente que tiene etiquetada información categórica. Por ejemplo, supogamos que creamos un vector entero y asociamos el número 1 a la etiqueta `"Argentina"`, el número 2 a `"Brasil"`, el 3 a `"Colombia"`, y así sucesivamente.

Diferencia entre *factores* y *caracteres*

A diferencia de los vectores de carácter, los factores son una clase de vector mucho más versátil, puesto que nos permiten almacenar a la vez

categorías y números. Los factores pueden llevar a confusión porque tienen la apariencia de un carácter, pero realmente son un número entero con una etiqueta con caracteres. Esta diferencia debemos tenerla muy en cuenta no solamente cuando hacemos operaciones, sino cuando descargamos bases de datos. Un factor con caracteres puede ser interpretado fácilmente por R como un vector de carácter cuando realmente nosotros lo queremos utilizar como un factor.

Para crear un factor, utilizaremos las funciones `factor()` o `as.factor()`. Normalmente, los factores se crean a partir de los vectores de carácter, pero para comprender bien la naturaleza de un factor ilustraremos primero cómo se crea a partir de un vector entero.

El argumento *stringsAsFactors*

Muchas funciones tienen incorporado un argumento que nos permite elegir si queremos que las variables categóricas sean *strings* o factores. Tanto `data.frame()`como la mayoría de funciones para importar marcos de datos como `read.csv()` tienen el argumento *stringsAsFactors*. Por defecto, este argumento es `TRUE`, de modo que convertirá automáticamente los caracteres en factores. Otras funciones, en cambio, como las del paquete *readr*, por defecto no convierten los caracteres en factores.

Crear un factor a partir de un vector entero

Fijémonos en el siguiente código: hemos creado un vector entero de nombre `entero`. Si pedimos tipo, clase y lo imprimimos, no veremos nada significativo: es un vector entero formado por números enteros. Ahora transformémoslo en factor. Si volvemos a pedir tipo y clase,

vemos que sigue siendo un vector entero, pero ahora R lo reconoce como factor. Si lo imprimimos, R ahora ya sabe que es un factor y nos dice que el vector tiene cuatro niveles, correspondientes a las cuatro «categorías» diferentes del vector. A continuación, asociamos un nombre categórico a cada uno de estos cuatro niveles con `levels()`. Si imprimimos de nuevo el vector, veremos que la consola nos enseña los niveles del vector en lugar de los números. El valor 1 está asociado al nivel `"Argentina"`, de modo que visualizaremos todos los valores 1 como Argentina, en concreto los dos primeros valores de la serie. Fijaos también en que el orden de los factores estará asociado al orden de los números enteros, no al orden alfabético de los niveles.

```
entero <- c(1L, 1L, 2L, 3L, 3L, 4L)
typeof(entero)
class(entero)
entero
entero <- as.factor(entero)
typeof(entero)
class(entero)
entero
levels(entero) <- c("Argentina", "Brazil", "Bolivia", "Colombia")
entero
attributes(entero)
```

Sin embargo, es poco probable que construyamos un factor a partir de un vector entero. El ejemplo anterior nos ha servido para comprender mejor la naturaleza de un factor. Pero para nuestra comodidad normalmente crearemos los factores a partir de vectores de carácter. En el ejemplo siguiente, hemos creado el vector `nucleares`, que responde a una variable categórica binaria que clasifica seis estados indeterminados según si son

potencias nucleares o no mediante dos categorías: `"Nuclear"` y `"No Nuclear"`. Podemos repetir de manera parecida las operaciones anteriores, primero examinando que efectivamente es un vector de carácter con `typeof()` y `class()`. A continuación, lo convertimos en factor, en este caso, creando un nuevo objeto llamado `factor_nucleares`. Con `typeof()` podemos comprobar que, efectivamente, este nuevo objeto es un vector entero y con `class()` veremos que es un factor.

```
nucleares <- c("Nuclear", "Nuclear", "No Nuclear", "Nuclear",
"No Nuclear", "Nuclear")
typeof(nucleares)
class(nucleares)
factor_nucleares <- factor(nucleares)
typeof(factor_nucleares)
class(factor_nucleares)
factor_nucleares
attributes(factor_nucleares)
```

Fijaos en que, como venimos de un vector de carácter, esta vez nos ha creado los niveles por orden alfabético. El primer nivel es `"No Nuclear"`, a pesar de que no es el valor que aparece primero en la secuencia del vector, mientras que el segundo nivel es `"Nuclear"`. Entonces, podemos decir que R, por orden alfabético, ha asignado el valor 1 a `"No Nuclear"` y el valor 2 a `"Nuclear"`.[9]

9. Hay que tener en cuenta que, aunque cada etiqueta corresponda a un número, eso no quiere decir que R entienda que la categoría asignada al número 1 es inferior a la categoría asignada al número 2. Por ahora, R solo ha puesto etiquetas a los números, de modo que estas variables siguen siendo categóricas nominales. Más adelante aprenderemos cómo ordenarlas de mayor a menor.

Podemos cambiar los nombres de los factores de dos maneras. Para ilustrarlo, hemos creado el factor `paises` con las categorías siguientes: `"BRA"`, `"ARG"`, `"VEN"`, `"URU"` y `"PAR"`. Si queremos cambiar estos nombres, podemos hacerlo, en primer lugar, con la función `levels()`. Con `levels(paises)` miramos primero el orden de los niveles. Vemos cómo están ordenados alfabéticamente. Una vez sabemos el orden, tenemos que crear un vector con los nombres nuevos, en el mismo orden que los niveles del factor, y asignarlo a los niveles del objeto `paises`. Si solo queremos cambiar una parte de los niveles, entonces tendremos que crear un vector con los nombres que queramos cambiar, por orden, y seleccionar con los corchetes la posición de los niveles después de la función `levels()`.

```
paises <- factor(c("BRA", "ARG", "VEN", "URU", "PAR"))
paises
levels(paises)
levels(paises)  <-  c("Argentina",  "Brasil",  "Paraguay",
"Uruguay", "Venezuela")
levels(paises)[4] <- c("URUGUAY")
paises

factor(paises, labels = c("Argentina", "Brasil", "Paraguay",
"Uruguay", "Venezuela"))
```

La segunda manera de cambiar los nombres de los factores[10] es dentro mismo de la función `factor()`, donde indicaremos los nombres nuevos, por orden, en el argumento `labels`.

10. Cuando tengamos algún problema asociado con un factor, es muy probable que lo podamos resolver con el paquete *forcats*.

3.1.3. Lógicos

Las variables categóricas nominales también pueden tomar la forma de un vector lógico, en el que la variable solo puede tener dos valores discretos y normalmente no ordenables: verdadero (`TRUE`) y falso (`FALSE`). En este caso se tratará de una variable binaria (también llamada *dicotómica*, *booleana* o *dummy*), que normalmente sirve para indicar la presencia o ausencia de un determinado concepto. Cuando el concepto está presente, la variable tomará el valor lógico `TRUE` mientras que cuando el concepto está ausente tomará el valor lógico `FALSE`.

La vida puede ser binaria

Toda variable se puede convertir en un vector lógico. Por ejemplo, podemos establecer una variable lógica, *democracia*, que tome el valor `TRUE` si el país observado es una democracia y el valor `FALSE` si no lo es. Lo mismo podemos hacer con la variable *mujer* (`TRUE` si es mujer, `FALSE` si no lo es), *rojo* (`TRUE` si el color es rojo, `FALSE` si es cualquier otro color), *año_2007* (`TRUE` si los datos son de 2007, `FALSE` si no lo son) o *paises_ricos* (`TRUE` si un país tiene un PIB per cápita superior a un umbral determinado y `FALSE` si es inferior).

Las variables lógicas no son muy frecuentes en los marcos de datos, pero son muy útiles como variable operativa cuando tengamos que seleccionar la presencia o ausencia de determinadas condiciones en una variable. A continuación, hemos creado el marco de datos `pov`, que contiene las variables *country* y *poverty*. La primera variable es un vector de carácter que representa los nombres de los países, mientras que la segunda variable es un vector numérico que representa la pobreza de cada país, medida

como el porcentaje de personas que viven por debajo de 1,90 dólares al día.[11]

```
country = c("Armenia", "Austria", "Benin", "Bolivia", "Brasil",
            "Colombia", "El Salvador", "Etiopia", "Honduras",
            "Indonesia")
poverty = c(1.90, 0.7, 49.6, 6.4, 3.4, 4.5, 1.9, 26.7, 16.2, 7.2)
pov <- data.frame(country, poverty)


> pov$country == "Austria"
FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE


> pov$poverty < 15
TRUE TRUE FALSE TRUE TRUE TRUE TRUE FALSE FALSE TRUE
```

Con el código `pov$country == "Austria"` pedimos que nos devuelva un vector lógico cuyos valores sean verdad si el valor correspondiente a la variable *country* es igual en Austria y falso si no lo es. Como vemos, solo nos devuelve como `TRUE` el segundo valor, y los otros nos los devuelve como `FALSE`. Con el código `pov$poverty < 15` pedimos un vector lógico que tenga los valores `TRUE` si el valor correspondiente a la variable *poverty* es inferior a 15 y `FALSE` si es igual o superior. Como vemos, el nuevo vector lógico es una variable dicotómica que nos dice para cada obser-

11. Fijaos en que hemos creado el marco de datos de una manera diferente a la de los anteriores ejemplos: primero hemos creado los vectores y después los hemos unido con la función `data.frame()`. R interpreta los nombres de los vectores como el nombre que tomarán las columnas del marco de datos.

vación si el porcentaje de personas que viven por debajo de 1,90 dólares al día es inferior a 15.

Operadores booleanos

Los operadores booleanos (OR, AND y NOT) son una manera común de trabajar cuando agrupamos condiciones lógicas. Cuando hacemos búsquedas avanzadas por internet usamos operadores booleanos, puesto que pedimos que la búsqueda incluya una palabra, pero no otra o queremos que incluya una palabra u otra. En nuestro caso, supongamos que queremos saber qué observaciones del marco de datos `pov` son o bien Austria o bien tienen una pobreza inferior al 15 %. Esta función nos la permite hacer el operador OR (símbolo | en R), que nos devuelve `TRUE` si al menos uno de los requisitos que hemos indicado se cumple. En el primer caso, Armenia no es Austria, pero tiene una pobreza baja, por lo cual nos devuelve un `TRUE` porque cumple uno de los requisitos. En el tercer valor, Benin no cumple ninguno de los dos requisitos y, por lo tanto, nos devuelve `FALSE`.

```
> pov$country == "Austria" | pov$poverty < 15
TRUE TRUE FALSE TRUE TRUE TRUE TRUE FALSE TRUE TRUE
```

El operador AND (símbolo & en R) es más restrictivo y solo nos devolverá `TRUE` en el supuesto de que se cumplan todos los requisitos indicados. En el ejemplo siguiente, solo el segundo valor nos lo devuelve como `TRUE` porque es el único caso que es Austria y tiene una pobreza inferior a 15.

```
> pov$country == "Austria" & pov$poverty < 15
FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Finalmente, tenemos el operador NOT (símbolo `!` en R), que nos indica que no tiene que cumplir el requisito que indicamos. Las dos maneras de utilizarlo que indicamos a continuación devuelven el mismo vector. Pedimos dos requisitos: no puede ser Austria y (AND) tiene que tener una pobreza inferior al 15 %. Como vemos, el símbolo de negación se puede situar, en el primer caso, al principio de la fórmula o, en el segundo caso, cambiando el igual por el símbolo `!=`.

```
> !pov$country == "Austria" & pov$poverty < 15
> pov$country != "Austria" & pov$poverty < 15
TRUE FALSE FALSE TRUE TRUE TRUE TRUE FALSE FALSE TRUE
```

3.2. Variables categóricas ordinales

Las *variables ordinales* son variables categóricas que se pueden ordenar de manera lógica.

Las variables ordinales se diferencian de las variables categóricas nominales porque son ordenables y de las numéricas discretas porque sus valores representan categorías en lugar de números. Esto significa que las operaciones que podemos hacer con los valores nos permiten saber si son iguales, diferentes, mayores o menores.[12]

12. Las operaciones que se pueden realizar con variables categóricas ordinales son las siguientes: igual: ==; no igual: `!=`; mayor: >; mayor o igual: >=; menor: < y menor o igual: <=.

Una variable ordinal tiene que ser necesariamente tratada con R mediante un vector lógico o un factor. Podemos codificar la variable como vector lógico siempre que sea una variable binaria y la categoría superior sea codificada como `TRUE`. En este caso, podemos seguir las instrucciones del apartado anterior sobre vectores lógicos. Sin embargo, en la mayoría de los casos trataremos las variables ordinales como factores.

Ejemplo

Una encuesta puede preguntar «valore su grado de satisfacción con la democracia» y dar como posibles respuestas «bajo», «medio» o «alto». Es evidente que «alto» es mayor que «medio» y «medio» es mayor que «bajo». Por lo tanto, en la variable hay tres categorías posibles y estas se pueden ordenar de manera lógica.

Diferencia entre una variable ordinal y una variable numérica discreta

Si podemos asignar un número a cada categoría de una variable ordinal, ¿que diferencia una variable ordinal de una variable discreta? Podríamos asignar, por ejemplo, el número 1 a «bajo», el 2 a «medio» y el 3 a «alto», pero estos números no tendrían significado propio, a diferencia de las variables numéricas. Es decir, la asignación sería igual de válida si para indicar «bajo», «medio» o «alto» utilizáramos 2, 4 y 6; 10, 11 y 12; 1.050, 1.100 y 1.150... En otras palabras, en las variables ordinales sabemos el orden de los valores, pero desconocemos el valor de cada categoría y la distancia exacta entre categorías.

3.2.1. Factores

Los factores son uno de los instrumentos principales que tenemos para almacenar variables categóricas, pero su gran potencial es que pueden guardar variables ordinales, puesto que tienen la capacidad de ordenar las categorías siguiendo el orden que indicamos. Para ilustrarlo, hemos replicado una clasificación ordinal del Banco Mundial que categoriza los países según su nivel de ingresos, que puede ser *low*, *lower-middle*, *upper-middle* y *high*. Para recrear esta clasificación, hemos construido el marco de datos `wb`, donde la primera variable *country* muestra quince países caribeños y la segunda variable *income* muestra el nivel de ingresos correspondiente.[13] Fiajos en la estructura de las variables, en el tipo de vector de la variable *country*, en los atributos de la variable *income* y también en cómo se vería esta última variable como vector entero.

```
wb <- data.frame(country = c("Antigua and Barbuda", "Belize",
"Costa Rica", "Dominica", "Dominican Republic", "El Salvador",
"Guyana", "Guatemala", "Haiti", "Honduras", "Jamaica",
"Nicaragua", "Panama", "Surinam", "Trinidad and Tobago"),
income = factor(c("high", "upper-middle", "upper-middle",
"upper-middle", "upper-middle", "lower-middle", "upper-middle",
"upper-middle", "low", "lower-middle", "upper-middle", "lower-
middle", "high", "upper-middle", "high")),
stringsAsFactors = FALSE)
```

13. Indicando `stringsAsFactors = FALSE` y aplicando `factor()` a la segunda variable hemos conseguido que la variable *country* sea un vector de carácter y que la variable *income* sea un factor.

```
str(wb)
typeof(wb$country)
attributes(wb$income)
unclass(wb$income)
```

Si nos fijamos, podemos comprobar que los niveles del factor `wb$income` están ordenados alfabéticamente, empezando por *high*, que sería el primero por orden alfabético. Esta observación la podemos hacer también con `levels(wb$income)`. R todavía no sabe qué nivel es superior al otro y por eso nos ordena los niveles alfabéticamente. Para convertir esta variable en ordinal, tenemos que introducir la función `factor()` con el nombre de la variable, un segundo argumento `order = TRUE` y un tercer argumento con `levels,` y un vector que indique, de mayor a menor, cómo se ordenarán los niveles. Intenta hacer la operación siguiente:

```
wb$income <- factor(wb$income, order = TRUE,
levels = c("low", "lower-middle", "upper-middle", "high"))


> class(wb$income)
"ordered" "factor"


> wb$income
Levels: low < lower-middle < upper-middle < high
```

Si después de transformar el vector consultamos la clase del objeto, veremos que nos indica que es un `"ordered" "factor"`. Si imprimimos el vector `wb$income` podemos comprobar que nos aparecerá el orden de los niveles, que no nos aparecía anteriormente. Ahora R tiene claro que `"low"` es inferior a `"lower-middle"`, que es inferior a `"upper-middle"`, que es inferior a `"high"`.

Del mismo modo que podemos convertir un vector de carácter en factor con `factor()` o `as.factor()`, podemos hacer el procedimiento a la inversa mediante `as.character()`. Si transformamos la variable en carácter (`wb$country <- as.character(wb$country)`), debemos tener en cuenta que R eliminará tanto la ordinalidad de los factores como los números enteros que tenían asociados y transformará los niveles en caracteres. Del mismo modo, `as.logical()` transformará un vector cualquiera en vector lógico y `as.numeric()` lo transformará en numérico.

3.3. Variables numéricas

Las *variables numéricas*, denominadas también *cardinales* o *cuantitativas*, son variables que representan números en sus valores.

Los valores de las variables numéricas se pueden representar con números que tienen significado por sí mismos. Que tengan significado numérico quiere decir, en primer lugar, que siempre podremos ordenar sus valores, puesto que, por ejemplo, un número negativo es inferior a un número positivo y el número 500 es superior al 450.

Que tengan significado numérico también significa que, además de ser ordenables, la distancia entre los valores de una variable numérica es conocida. En una variable ordinal no podemos conocer la distancia entre un valor `"Medio"` y un valor `"Alto"`. En cambio, en una variable numérica podemos conocer y calcu-

lar la distancia entre valores. Esto nos permitirá hacer operaciones aritméticas como sumar, restar, multiplicar, dividir, sacar la raíz cuadrada o buscar el logaritmo neperiano.[14]

Ejemplos de variables numéricas

La temperatura, la latitud, el porcentaje de tierra cultivable, la cantidad de ayuda oficial al desarrollo, la migración neta, la tasa de crecimiento natural de la población o las emisiones de dióxido de carbono a la atmósfera.

Para almacenar datos numéricos, R utiliza dos tipos de vectores: el vector entero y el vector numérico o doble. Con el vector entero, almacenaremos normalmente las variables numéricas discretas, que no pueden tener decimales, mientras que con el vector numérico almacenaremos las variables continuas, que sí pueden contener decimales. Aunque es importante conocer la distinción entre discretas y continuas, en la práctica, con R, esta distinción nos será poco útil, puesto que podremos hacer las mismas operaciones tanto con un tipo de variable como con el otro. Por lo tanto, la distinción tiene más sentido teórico que práctico. Además, para almacenar un vector entero lo tendremos que especificar con una `L` al final de cada valor, lo que no facilita la tarea de crear vectores enteros.[15] A continuación mostramos dos maneras de crear un vector entero y ver el tipo de vector que ha creado.

14. Las operaciones que se pueden realizar con variables numéricas son las siguientes: igual: `==`; no igual: `!=`; mayor: `>`; mayor o igual: `>=`; menor: `<`; menor o igual: `<=`; sumar: `+`; restar: `–`; multiplicar: `*`; dividir: `/`; exponencial: `^`; logaritmo: `log()`.

15. La letra `L` no aparecerá en el vector cuando lo visualicemos, pero es la manera de que R sepa que lo tiene que almacenar como vector entero.

```
typeof(c(4L, 8L, 15L, 16L, 23L, 42L))
typeof(as.integer(c(4, 8, 15, 16, 23, 42)))
```

Entonces, como norma general y para ser prácticos, trataremos con variables numéricas con el vector doble o numérico, aunque tenemos que saber que también podemos utilizar un vector entero para tratar con las variables numéricas discretas. En un plano teórico, las variables numéricas discretas contienen valores numéricos específicos que pueden ser contados o enumerados. Por lo tanto, estos valores son limitados y no aceptan decimales, como por ejemplo el número de hijos, la población de un país, el número de mujeres en los parlamentos nacionales, el número de habitantes de un país o el número de homicidios.

Discretizar una variable numérica (1): El método *if_else()*

A veces podemos querer transformar una variable numérica en una variable discreta, normalmente categórica. A este proceso lo denominaremos *discretizar*. Uno de los métodos existentes es mediante la función `if_else()`, disponible en el paquete dplyr. En esta función tenemos que dar tres argumentos:

- En primer lugar, estableceremos una condición lógica: que pueda ser verdadero o falso.
- En segundo lugar, asignaremos el valor que tendrá la nueva variable si la condición es verdadero.
- Y, en tercer lugar, asignaremos el valor de la nueva variable si la condición es falso. En el ejemplo siguiente, indicamos que si el PIB per cápita es superior o igual a 30.000, el valor del nuevo vector será 1. Si, en cambio, el PIB per cápita es inferior a 30.000, el valor del nuevo vector será 0.

```
if_else(md_agr$PIB_cap >= 30000, 1, 0)
```

El vector que creamos no hace falta que sea numérico. Si indicamos `"Rico"` y `"Pobre"` en el segundo y el tercer argumento nos devolverá un vector de carácter, mientras que si indicáramos `TRUE` y `FALSE` nos devolvería un vector lógico. En el primer argumento, con la función `between()` podremos indicar un intervalo de datos. Por ejemplo, en el código siguiente, asignaremos el valor `"Medium"` a todos los valores que estén entre 20.000 y 30.000 (incluidos) y el valor `"Other"` a los que estén fuera de este intervalo.

```
if_else(between(md_agr$PIB_cap, 20000, 30000), "Medium",
"Other")
```

Esta operación se puede hacer también con la función del paquete base de R `ifelse()`. Otra función muy similar que tiene dplyr es `recode()`.

Las variables numéricas continuas, por su parte, nos permiten adoptar un número infinito de valores. La infinidad se tiene que entender como el espacio que separa dos valores. Pensad en el porcentaje de superficie forestal de un país. La cifra podría ser, supongamos, un 15,69 %. ¿Es un valor numérico continuo, puesto que puede aceptar decimales, pero que lo hace infinito en comparación con un valor numérico discreto? Son variables numéricas continuas el PIB per cápita, la esperanza de vida, el tiempo medio de uso de internet, la ayuda oficial recibida para desarrollo o el porcentaje de la población que tiene acceso a agua potable.

Discretizar una variable numérica (2): el método *case_when()*

Una segunda manera de discretizar una variable es con la función del paquete dplyr `case_when()`, especialmente útil cuando queremos crear una nueva variable con más de dos categorías. Tomando de ejemplo el marco de datos `pov`, utilizado anteriormente en este módulo, queremos crear una nueva variable *f_poverty* con tres categorías a partir de la variable numérica *poverty*. A los números inferiores a 2, los denominaremos `"Low"`, a los números entre 2 y 10 los denominaremos `"Medium"` y al resto de valores los denominaremos `"High"`.

```
pov$f_poverty <- case_when(pov$poverty < 2 ~ "Low",
between(pov$poverty, 2, 10) ~ "Medium", TRUE ~ "High")
```

Cada argumento de la función `case_when()` es una secuencia de dos fórmulas. En la primera indicamos a qué valores aplica la transformación y, separado por el símbolo ~, en la segunda indicamos el nuevo valor. Los nuevos valores no tienen que ser necesariamente de carácter. Pueden ser lógicos o numéricos. Como vemos en el código, hemos utilizado la función `between()` para indicar un intervalo de valores. Si en el último argumento de la serie indicamos `TRUE`, nos agrupará el resto de casos que no hayamos especificado en ninguna condición.

La operación nos ha convertido la nueva variable *f_poverty* en un vector de carácter. Para hacerla ordinal, la hemos tenido que convertir en factor y asignar los niveles y el orden correspondiente.

```
pov$f_poverty <- factor(pov$f_poverty, order = TRUE,
levels = c("Low", "Medium", "High"))
```

Es importante saber que podemos hacer dos tipos de operaciones con las variables numéricas según la longitud de los vectores:

1) En primer lugar, si hacemos una operación de un solo valor con un vector de una determinada longitud, R aplicará la operación del valor a cada uno de los valores del vector. En el ejemplo siguiente, R multiplicará por 3 cada uno de los valores del vector.

2) En segundo lugar, podemos hacer operaciones entre vectores de igual longitud. En este caso, R devolverá un vector de la misma longitud y aplicará la operación entre vectores. En el ejemplo, R multiplicará el primer valor del primer vector por el primer valor del segundo vector y lo colocará en el primer valor del vector resultante. En el segundo valor del vector resultante, R habrá multiplicado el segundo valor del primer vector por el segundo valor del segundo vector, y así sucesivamente.

```
3 * c(1, 3, 6)
c(1, 3, 6) * c(1, 3, 6)
```

En este código, en el primer caso nos devolverá el vector `c(3, 9, 18)` y, en el segundo caso, el vector `c(1, 9, 36)`.

Ejercicios de autoevaluación

Para un mejor aprendizaje, intentad hacer mentalmente el máximo de ejercicios posible, sin utilizar R.

1) ¿Cuál es el valor del objeto nuevo?

```
suma <- 2 + 2
nuevo <- suma * 5
```

2) ¿Qué nos devuelve la operación final de esta secuencia?

```
cinco <- 5
cinco <- 4
cinco * cinco
```

3) ¿Qué nos devuelve la operación final de esta secuencia?

```
Cinco <- 5
cinco <- 4
Cinco + cinco
```

4) ¿Qué nos devuelve esta operación?

```
4:8
```

5) ¿Qué número devuelve esta operación?

```
objeto <- c(1,7:9)
sqrt(sum(objeto))
```

6) ¿Cuántos decimales devuelve esta operación?

```
round(2.718)
```

7) ¿Qué tipo de vector nos generará este valor?

```
"TRUE"
```

8) ¿Qué tipo de vector nos generará este valor?

```
"324"
```

9) ¿Qué nos devuelve la operación final de esta secuencia?

```
objeto <- c(1,7:9)
objeto[c(1:2, 4)]
```

10) ¿Cuál es la clase de vector que nos aparecerá?

```
class(c("3", FALSE, 42))
```

11) ¿Qué vector nos devuelve esta operación?

```
2 * c(1,2)
```

12) ¿Qué vector nos devuelve esta operación?

```
walt <- c(1, 2, 3)
walt + c(4, 4, 4)
```

13) Indicad el resultado de las siguientes operaciones:

```
md42 <- data.frame(p = c("Francia", "Andorra"), pibxc =
c(35000, 40000), pob = c(50, 0.1))
a) md42[2,2:3]
b) round(md42[2,3])
c) sample(md42[,3], 1)
d) sum(md42[,2] / 2)
```

14) Indicad el resultado del siguiente código:

```
str_to_lower(c("ABQ", "LAX", "JFK", "BCN"))
```

15) ¿Qué vector obtendremos en el siguiente código?

```
as.integer(factor(c("A", "B", "A", "B", "C", "A")))
```

16) Indicad el resultado lógico de la siguiente operación:

```
TRUE > FALSE
```

17) Indicad el resultado lógico de la siguiente operación:

```
TRUE == FALSE
```

18) Indicad el resultado lógico de la siguiente operación:

```
"Spain" == "SPAIN"
```

19) Indicad el resultado lógico de la siguiente operación:

```
TRUE & FALSE
```

20) Indicad el resultado lógico de la siguiente operación:

```
TRUE | FALSE
```

Solucionario

1) `20`
2) `16`
3) `9`
4) `4 5 6 7 8`
5) `5`
6) `Cero. Con round(2.718) nos devolverá el valor 3.`
7) `carácter (porque va entre comillas)`
8) `carácter (porque va entre comillas)`
9) `1 7 9`
10) `"character"`
11) `2 4`
12) `5 6 7`
13)

```
a)  pibxc pob 40000 0.1
b)  0
c)  50 o 0.1 según el azar
d)  37500
```

14) `abq, lax, jfk, bcn`
15) `1 2 1 2 3 1`
16) `TRUE`
17) `FALSE`
18) `FALSE`
19) `FALSE`
20) `TRUE`

Solución del ejercicio 1

```
numérico carácter carácter carácter
```

Solución del ejercicio 2

```
round(17/3)
sum(1:10)
max(vector_numerico)
pi
round(pi)
```

Solución del ejercicio 4

```
seq(from = 7, to = 32, by = 5)
rep(x = q, length.out = 12, each = 2)
sort(c(3, 9, 5, 6), decreasing = TRUE)
```

Glosario

<- Crea un objeto.

: Selecciona una cadena de valores según su posición.

:: Especifica la función de una librería determinada.

? Abre el cuadro de ayuda de una función u objeto.

?? Hace una búsqueda por palabra clave.

[ ] Devuelve los valores seleccionados de un vector.

[,] Devuelve filas y columnas de un marco de datos.

$ Establece la separación entre el marco de datos y la variable.

| Booleano OR, devuelve TRUE si se cumple uno de los requisitos.

& Booleano AND, devuelve TRUE si se cumplen todos los requisitos.

! Booleano NOT, niega la condición lógica.

== Igual, usado en operaciones lógicas.

!= No es igual que.

>= Mayor o igual que.

< Menor que.

<= Menor o igual que.

> Mayor que.

+ Suma.

– Resta.

***** Multiplica.

/ Divide.

^ Eleva exponencialmente un valor.

`args()` Muestra los argumentos de una función

`as.character()` Convierte una variable en carácter

`as.data.frame()` Transforma un objeto en un marco de datos.

`as_data_frame()` Transforma un objeto en un marco de datos (dplyr).

`as.factor()` Convierte una variable en factor

`as.logical()` Convierte una variable en lógica

`as.numeric()` Convierte una variable en numérica.

`as_tibble()` Crea un *tibble*, un tipo de marco de datos (dplyr).

`attributes()` Muestra los atributos de uno objeto.

`c()` Crea un vector.

`case_when()` Discretización múltiple de una variable numérica (dplyr).

`class()` Muestra la clase de objeto.

`clean_names()` Limpia los nombres de un vector de carácter (janitor).

`data.frame()` Crea un marco de datos.

`factor()` Crea un factor.

`help()` Abre el cuadro de ayuda de una función u objeto.

`if_else()` Discretización binaria de una variable numérica (dplyr).

`length()` Devuelve la longitud de un objeto.

`levels()` Muestra los niveles de un factor.

`ls()` Muestra los objetos creados.

`max()` Devuelve el valor máximo de un vector.

`recode()` Discretiza una variable numérica (dplyr).

`rep()` Repite una cadena de valores.

`round()` Redondea un valor numérico.

`sample()` Devuelve la muestra aleatoria de un objeto determinado.

`seq()` Devuelve una secuencia de valores.

`sort()` Ordena los valores de un vector.

`sum()` Sumatorio de los valores de un objeto.

`str_detect()` Devuelve un vector lógico que indica si ha encontrado el conjunto de caracteres que hemos especificado (stringr).

`str_replace()` Cambia, en un vector, un conjunto de caracteres por otro (stringr).

`str_remove()` Elimina el conjunto de caracteres que indicamos (stringr).

`str_to_lower()` Convierte todos los caracteres en minúsculas (stringr).

`str_to_upper()` Convierte todos los caracteres en mayúsculas (stringr).

`tbl_df()` Crea un *tibble*, un tipo de marco de datos (dplyr).

`tibble()` Crea un *tibble*, un tipo de marco de datos (dplyr).

`tribble()` Crea un *tibble*, un tipo de marco de datos (dplyr).

`typeof()` Devuelve el tipo de objeto.

`unclass()` Elimina la clase de un objeto.

`View()` Muestra el objeto en el visor.

Bibliografía

Grolemund, G.; Wickham, H. (2016). *R for Data Science* [documento en línea]. Canadá: O'Reilly. [Fecha de consulta: 9 de julio de 2020]. <https://es.r4ds.hadley.nz/>

Halperin, S.; Heath, O. (2016). *Political Research: Methods and Practical Skills*. Oxford: Oxford University Press.

R Development Core Team (2000). *Introducción a R. Notas sobre R: Un entorno de programación para análisis de datos y gráficos* [documento en línea]. [Fecha de consulta: 9 de julio de 2020]. <https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf>

Capítulo II
Explorar, transformar y visualizar

1. Introducción

El objetivo de este capítulo es familiarizarnos con el entorno R de una manera ágil y sin demasiada carga teórica para poder aplicar rápidamente los conocimientos aprendidos al usar el programa. En las páginas siguientes, aprenderemos en pocos pasos a transformar un marco de datos de dimensiones considerables en información útil y visualmente atractiva. Para este propósito, tendremos que aprender las funciones básicas de dos de los paquetes esenciales de R: dplyr y ggplot2. La librería dplyr incluye principalmente funciones orientadas a manipular marcos de datos. Cuando decimos *manipular*, no nos referimos a ello en mal sentido, sino en el sentido de adaptar y transformar los datos en información útil que nos ayude a responder a preguntas concretas que nos queramos formular. Manipular significa, por ejemplo, cambiar el orden de las filas de un marco de datos sobre la base de un criterio determinado, seleccionar una parte de las filas o de las columnas, o bien crear columnas con información nueva a partir de datos ya existentes. Una vez hayamos transformado los datos a nuestro gusto con dplyr, la librería ggplot2 nos permitirá crear visualizaciones gráficas para poder comunicar de manera atractiva nuestros resultados.

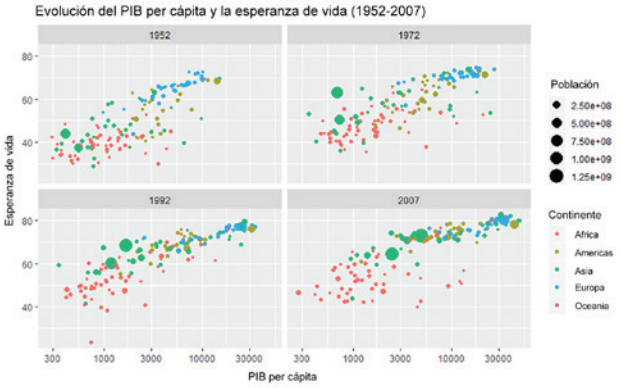
Para este proceso, tendremos que tener instalados y cargados dplyr y ggplot2 en R y también el paquete gapminder, que utilizaremos para hacer los ejercicios en este capítulo. Este paquete inclu-

ye un marco de datos con indicadores socioeconómicos como el PIB per cápita o la esperanza de vida en 142 países diferentes. Una vez tengáis los paquetes instalados y cargados, tecleando la función `search()` podéis comprobar que efectivamente dplyr, ggplot2 y gapminder están cargados en el Global Environment de R. Instalaremos los paquetes con la función `install.packages()` y los cargaremos, por separado, con `library()`.

En las páginas siguientes, utilizaremos el lenguaje de R de manera aplicada, dominaremos los instrumentos básicos para transformar y visualizar datos, y seremos capaces de comunicarnos con el programa de manera fluida. Sin ir más lejos, al final del capítulo habremos aprendido a construir visualizaciones como la figura 1, donde seremos capaces de agrupar en un solo gráfico hasta cinco variables.

¿Sois capaces de identificar cuáles son cada una y cómo están representadas?

Figura 1. Visualización de un marco de datos tratado



Las cinco variables del gráfico. El PIB per cápita de cada país es la variable que está representada en el eje horizontal. En el eje vertical, encontramos la esperanza de vida. La tercera variable es el continente, donde cada categoría tiene asignado un color diferente. La población de cada país está representada por el tamaño de cada uno de los puntos, de forma que países con poca población tendrán un punto pequeño y países grandes tendrán un punto mayor. La quinta variable es el año. Hemos construido una parrilla de cuatro gráficos que representan cuatro años diferentes. Fuente: elaboración propia.

Interpretar números como e+08

Es bastante habitual en R encontrar números con notación científica que acostumbran a tener el formato e+. Interpretarlos es más fácil de lo que parece, puesto que simplemente tenemos que mover los decimales tantas veces a la derecha como nos indique el último número, si es positivo, o tantas veces a la izquierda si es negativo. Por ejemplo, 2.50+e08 se traducirá como 250.000.000.

Fijaos en cuántas preguntas podemos llegar a responder con esta figura:

1) ¿Hay una relación positiva entre el PIB per cápita y la esperanza de vida?
2) ¿En qué continente son más bajos la esperanza de vida y el PIB per cápita?
3) ¿En qué continente están los países con más población?
4) ¿A qué continente pertenecen la mayoría de los países con un PIB per cápita y una esperanza de vida altos?

Si os fijáis, estas preguntas las podemos contestar con una visualización rápida. También podemos apreciar otros detalles, como que en 1992 tenemos un caso extremo: un país africano con una esperanza de vida extremadamente baja. También vemos algunos casos, especialmente en África, pero también en Asia en 1952 y en 1972, de países relativamente ricos, pero con una esperanza de vida muy baja. En cambio, no observamos que haya casos en el sentido contrario: países muy pobres, pero con una esperanza de vida alta.

En este capítulo, aprenderemos a hacer este proceso: observar y explorar un marco de datos, formularnos preguntas sobre los

datos y examinar cómo se relacionan entre ellos, tratarlos para poder responder a estas preguntas, elegir la mejor visualización para poder responderlas y, finalmente, comunicarlas. La visualización nos puede ayudar a formular nuevas preguntas, que nos pueden llevar a investigar nuevos aspectos de los datos.

2. Explorar

La exploración inicial de los datos[1] es el primer paso que tenemos que hacer cuando tenemos delante un marco de datos (Babbie, 2013, pág. 90; King y otros, 1994). El objetivo principal consiste en entender, a grandes rasgos, cuáles son la composición y la estructura de nuestros datos. Nos interesa saber las dimensiones del marco de datos, el número de variables que hay, la tipología de estas variables, y también nos puede interesar observar una pequeña muestra de los datos. Una vez hayamos hecho esta exploración general, el segundo paso consistirá en ver las características más sustantivas de cada una de las variables de interés, de manera más específica.

1. En la exploración inicial, siempre es conveniente dar un vistazo a las instrucciones, al libro de códigos o al material asociado a las bases de datos. En el caso de los marcos de datos que provienen de paquetes de R, podemos consultar la ayuda por medio, indistintamente, de `?gapminder` o bien de `help(gapminder)`. En este pequeño manual hay una descripción de las variables que nos permite, por ejemplo, ver una descripción de qué es y cuánto mide cada variable.

2.1. Exploración general

Como ya hemos incorporado a R el paquete gapminder, lo primero que haremos es una exploración inicial de los datos. Directamente, podemos imprimir el marco de datos `gapminder` para ver en la consola una idea inicial.

```
> gapminder
# A tibble: 1,704 x 6
     country       continent   year    lifeExp   pop        gdpPercap
     <fct>         <fct>       <int>   <dbl>     <int>      <dbl>
1    Afghanistan   Asia        1952    28.8      8425333    779.
2    Afghanistan   Asia        1957    30.3      9240934    821.
3    Afghanistan   Asia        1962    32.0      10267083   853.
4    Afghanistan   Asia        1967    34.0      11537966   836.
5    Afghanistan   Asia        1972    36.1      13079460   740.
6    Afghanistan   Asia        1977    38.4      14880372   786.
7    Afghanistan   Asia        1982    39.9      12881816   978.
8    Afghanistan   Asia        1987    40.8      13867957   852.
9    Afghanistan   Asia        1992    41.7      16317921   649.
10   Afghanistan   Asia        1997    41.8      22227415   635.
# ... with 1,694 more rows
```

¿Qué observamos en esta tabla? En primer lugar, vemos que es un marco de datos en formato *tibble* que contiene 1.704 observaciones y seis variables. Las variables, situadas en las columnas, son el país, el continente, el año, la esperanza de vida, la población y el PIB per cápita. Estas variables están formadas por tipos de vectores diferentes. Podemos comprobar que las variables *país* y *continente* están registradas como factores <*fctr*>, las variables *año* y *población* están registradas como vector entero <*int*>, mientras que las dos restantes son un vector numérico o doble <*dbl*>.

En la dimensión vertical están las observaciones. En lo que nos tenemos que fijar en esta dimensión es en la unidad de análisis del marco de datos. Es decir, tenemos que comprobar qué mide cada una de las filas. Fijémonos en que, en este marco de datos, cada fila representa una combinación de país y año, de forma que el resto de variables nos muestran datos en relación con un país determinado en un año concreto.

Ver los nombres de cada columna

La función `names()` nos permite ver los nombres que toman las variables de un marco de datos. Solo hace falta que introduzcamos el nombre del marco de datos dentro del paréntesis. En marcos de datos muy anchos, con muchas columnas, podemos utilizar los corchetes para seleccionar una parte de las columnas que queramos visualizar. Por ejemplo, `names(gapminder)[2:4]` nos devolverá los nombres de las columnas entre la 2 y la 4.

También podemos explorar el marco de datos de una manera más ajustada con las funciones `head()` o `tail()`. La primera función muestra por defecto las seis primeras filas del marco de datos, mientras que la última nos muestra las seis últimas. Si en lugar de seis queremos ver una cantidad diferente de observaciones, podemos indicarlo añadiendo dentro de la función un nuevo argumento con el número de observaciones que queremos visualizar.

Ejemplo

Si queremos explorar las primeras quince observaciones, introduciremos el código `head(gapminder, 15)`. Si queremos explorar las diez últimas, escribiremos `tail(gapminder, 10)`.

Otras funciones que podemos utilizar para explorar de manera general los datos son `dim()`, para ver el número de filas y de columnas, o `str()` y `glimpse()`, que hacen tareas exploratorias parecidas.[2] Estas dos funciones son muy útiles para explorar marcos de datos con muchas variables, puesto que nos muestran las columnas verticalmente. En lugar de ver, como es habitual, las variables en el eje horizontal y las observaciones en el eje vertical, con estas funciones encontraremos las variables distribuidas en vertical y encabezadas por el símbolo del dólar (`$`). La razón por la cual se nos muestra así es porque, en la exploración inicial, nos interesará más bien observar todas las variables, pero solo algunas observaciones. Esta operación nos será más fácil si nos desplazamos en vertical por la consola, que no tiene límites para mostrarnos la información en formato vertical, pero sí que nos tendría que cortar los datos si nos los muestra en horizontal.

¡Los datos en la cabeza, no en la pantalla!

La manera de explorar marcos de datos con R funciona de modo diferente en comparación con otros programas como Excel o SPSS. En lugar de visualizar siempre el marco de datos en pantalla y desplazarnos de manera horizontal y vertical, R asume que, cuando trabajamos con grandes bases de datos, no necesitamos visualizar la tabla entera, sino tener una idea general de su contenido. Es por eso por lo que nos tendremos que acostumbrar a tener la estructura de los datos ¡en la cabeza, no en la pantalla!

2. La función `glimpse()` pertenece al paquete dplyr, mientras que `str()` es una función de base de R. Podríamos considerar `glimpse()` como una versión avanzada de `str()`, puesto que nos devuelve una tabla más limpia y que se ajusta automáticamente a la anchura de nuestra consola. Intentad imprimir `gapminder` con las dos funciones para ver las diferencias visuales.

A continuación, hemos llamado con la función `str()` a la estructura de gapminder. Si el marco de datos tuviera veinte variables más, sería más cómodo explorar los datos de este modo que en horizontal. Podemos observar, por ejemplo, cuántas categorías hay en las variables definidas como factores. La variable *country* tiene 142 países diferentes (nos indica que es un factor con 142 niveles), mientras que la variable *continent* tiene cinco continentes diferentes (cinco niveles).

```
> str(gapminder)
Clases 'tbl_df', 'tbl' and 'data.frame': 1704 obs. of 6 variables:
$ country   : Factor w/ 142 levels "Afghanistan",..: 1 1 1 ...
$ continent : Factor w/ 5 levels "Africa","Americas",..: 3 ...
$ year      : int 1952 1957 1962 1967 1972 1977 1982 1987 ...
$ lifeExp   : num 28.8 30.3 32 34 36.1 ...
$ pop       : int 8425333 9240934 10267083 11537966 ...
$ gdpPercap : num 779 821 853 836 740 ...
```

Ahora ya tenemos una idea general de qué contiene el marco de datos `gapminder`. Durante la exploración general, habrá variables que nos habrán llamado más la atención que otras y que querremos trabajar. O bien pensemos que puede haber observaciones (países y años concretos) que nos interese más analizar.

2.2. Exploración específica

La exploración específica permite obtener más información del contenido de las variables del marco de datos que más nos han llamado la atención. Como ya sabéis, para denominar a una

variable que se encuentra dentro de un marco de datos, tenemos que escribir el nombre del marco de datos, seguido del símbolo `$` y a continuación el nombre del vector correspondiente. Si lo hacemos así, R nos devolverá todo el vector. Cuando trabajamos con grandes bases de datos, los vectores acostumbran a ser larguísimos, de forma que si observamos todo el vector no podremos sacar información relevante. Probad, por ejemplo, el siguiente código:

```
gapminder$country
```

En lugar de nombrar a la variable directamente, lo que haremos es pedir información más concreta mediante algunas funciones. En la exploración general, hemos visto que la variable *country* tenía 142 países. Podríamos, por ejemplo, preguntarnos cuáles son, o bien cuáles son los diferentes años en que los datos han sido capturados. La función `unique()` nos ayuda a obtener los valores únicos de una variable, de forma que nos será muy útil para variables categóricas e incluso para numéricas discretas. Así, `unique(gapminder$country)` nos devolverá un vector con los países únicos y `unique(gapminder$year),` todos los años de los que tenemos datos:

```
> unique(gapminder$year)
 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

R nos devuelve un vector cuyo primer año del que observamos datos es 1952 y el último, 2007. Entre estos dos periodos, tenemos datos cada cinco años. Podríamos combinar esta función con otras que ya hemos visto anteriormente, como por ejemplo, `head(unique(gapminder$year), 5)`

para ver solo los primeros cinco años que aparecen en la base de datos. En el supuesto de que queramos saber solo los años más actuales, una opción sería añadir la función `sort()` y pedir que nos ordene los años en orden descendente y, a continuación, que nos enseñe los cinco valores más altos: `head(sort(unique(gapminder$year), decreasing = TRUE), 5)`. Si eliminamos el argumento `decreasing` o bien lo pasamos a `FALSE`, veremos los cinco valores más bajos de la variable.

Contar casos

Si nos da pereza contar cuántos años únicos tenemos, podemos pedirle a R que los cuente por nosotros con la función `length()`. Si introducimos `length(unique(gapminder$year))` vemos que tenemos doce años diferentes.

Una función muy utilizada y que nos resultará muy útil para la exploración específica es `summary()`. Esta orden nos mostrará un resumen del objeto de interés. Esto quiere decir que podemos pedir tanto el sumario entero del marco de datos `gapminder` (nos devolverá un sumario de cada una de sus variables) como el de una variable específica. R nos ofrece un sumario diferente según el tipo de vector. En el código siguiente hemos pedido un sumario de *continent*, *lifeExp* y *gdpPercap*.

```
> summary(gapminder$continent)
Africa     Americas   Asia       Europe     Oceania
624        300        396        360        24


> summary(gapminder$lifeExp)
```

```
Min.      1st Qu.   Median    Mean      3rd Qu.   Max.
23.60     48.20     60.71     59.47     70.85     82.60


> summary(gapminder$gdpPercap)
Min.      1st Qu.   Median    Mean      3rd Qu.   Max.
241.2     1202.1    3531.8    7215.3    9325.5    113523.1
```

El `summary()` permite hacernos una idea de cómo están distribuidos los valores de la variable según el tipo de vector con que lo hayamos codificado. Fijaos en que, de este modo, podemos responder a preguntas más específicas, como por ejemplo:

«¿En qué continente tenemos más datos registrados?»
«¿Cuál es la esperanza de vida máxima de un país?»
«¿Y el PIB per cápita mínimo?»

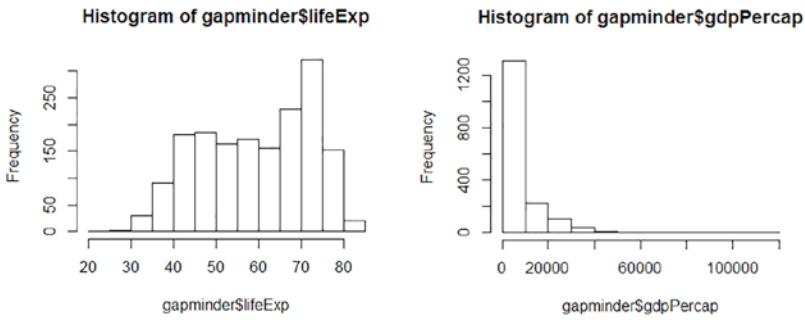
También podemos hacer algunas apreciaciones interesantes con los datos que tenemos: prácticamente no hay diferencia entre la mediana y la media en la esperanza de vida, mientras que sí que la hay en el PIB per cápita.

En último lugar, en la exploración específica también podemos pedir la visualización gráfica de los datos mediante algunas funciones de los paquetes de base de R. Según el tipo de variable que queramos visualizar, utilizaremos una función diferente. Para una variable numérica, emplearemos la función `hist()` para visualizar un histograma. Para las variables categóricas, usaremos `plot()`, función que también nos irá bien para visualizar la interacción entre dos datos numéricos. Para visualizar la interacción entre una variable categórica y una numérica usaremos `boxplot()`. Probad de reproducir los códigos siguientes:

```
hist(gapminder$lifeExp)
hist(gapminder$gdpPercap)
plot(gapminder$continent)
plot(gapminder$gdpPercap, gapminder$lifeExp)
boxplot(gapminder$continent, gapminder$lifeExp)
```

En los dos primeros códigos, hemos pedido un histograma porque es la manera de visualizar cómo están distribuidos los valores de variables numéricas, como la esperanza de vida y el PIB per cápita. Fijémonos en la columna más alta de la figura 2, que indica el intervalo que tiene más observaciones. Esta columna está situada entre 70 y 75 años en la esperanza de vida y por debajo los 10.000 dólares en el PIB per cápita (hay que tener en cuenta que las observaciones empiezan a partir de 1952 y, por lo tanto, encontraremos muchos países con un PIB per cápita muy bajo).

Figura 2. Visualización simple de un histograma



Fuente: elaboración propia.

También observamos que la distribución de los valores en un histograma y otro es completamente diferente. Mientras que los casos están distribuidos de manera suficientemente uniforme a

lo largo del histograma de esperanza de vida, la mayoría de casos del PIB per cápita están situados en un extremo. Esto explica el motivo por el cual la media y la mediana son tan diferentes en el PIB per cápita y tan similares en la esperanza de vida. Veremos más detalles sobre estos estadísticos descriptivos más adelante, lo cual nos ayudará a entender mejor la diferencia entre la media y la mediana. Podéis probar a hacer una exploración específica de los otros gráficos que tenéis en el código anterior.

3. Transformar

A estas alturas, gracias a la exploración inicial, ya tenemos una imagen mental bastante clara de la estructura de nuestros datos. Ahora que sabemos qué variables tenemos, de qué tipo son y cómo están distribuidos los valores en cada una de ellas, es el momento de hacernos preguntas todavía más concretas sobre estos datos. Y por eso tendremos que aprender a transformar y manipular nuestros datos. Una pregunta podría ser, por ejemplo, cómo ha evolucionado la esperanza de vida de España a lo largo de los años. También nos podríamos preguntar qué diferencias hay entre la evolución del PIB per cápita entre España y Francia. O incluso podríamos crear datos nuevos a partir de los ya existentes.

En este apartado, aprenderemos a transformar los datos con las funciones del paquete dplyr a partir de una sintaxis basada principalmente en seis verbos: *filtrar, ordenar, seleccionar, mutar, resumir* y *agrupar*, que resumimos en la tabla 8.

Tabla 8. Seis funciones básicas del paquete dplyr

Objetivo	Función	Acción
Manipular filas	`filter()`	Elimina filas.
	`arrange()`	Reordena filas.
Manipular columnas	`select()`	Elimina o reordena las columnas.
	`mutate()`	Crea nuevas columnas o transforma las existentes con valores construidos a partir de datos.
Manipular grupos de observacions	`group by()`	Agrupa observaciones.
	`summarize()`	Resume varias observaciones en unas sola mediante una operación.

Fuente: elaboración propia.

Obtener los datos de PIB total

Si nos fijamos bien en los datos que tenemos, veremos que podemos obtener el PIB total de cada país, puesto que disponemos de información del PIB per cápita y la población. Como sabéis, el PIB de un país es el conjunto de bienes y servicios producidos por todos los habitantes que viven en un país durante un año. Como tenemos los datos del número de habitantes que vive en un país en un año determinado (`pop`) y lo que producen de media (`gdpPercap`), multiplicando estos dos datos podremos crear una nueva columna con el PIB total.

En las próximas páginas explicamos, una por una, dichas funciones. Pero también tenemos que saber que sacaremos mayor partido de ello si las aplicamos de manera simultánea. Combinar funciones es una tarea visualmente poco agradable, puesto que tenemos que poner funciones dentro de otras funciones, y es complicado leer un código entre tantos paréntesis.

Si, siguiendo la tabla anterior, queremos filtrar unos datos, después crear una nueva columna y finalmente ordenar los datos, podemos, o bien crear un código diferente para cada verbo, o bien intentar añadir capas y capas de paréntesis en un solo código.

Por suerte, hay otra opción para ahorrarnos todos estos procedimientos. Dentro del paquete dplyr, tenemos la *pipe* (símbolo `%>%`), herramienta que nos permite aplicar sobre el mismo objeto varias funciones de una manera mucho más intuitiva y ordenada. Lo que hace la *pipe* es cambiar la gramática de las funciones, de forma que nos transforma una función de varios argumentos `f(x, y)` en la estructura `x %>% f(y)`. A primera vista, cuesta comprender su utilidad, pero a través de diferentes ejemplos iremos comprobando la funcionalidad. Para ilustrar una primera idea, en la tabla 9 observamos cómo las funciones `arrange()` y `filter()` se agrupan a la manera clásica y con la *pipe*. A la manera clásica, empezamos por la función y vamos poniendo los argumentos dentro del paréntesis. Con la *pipe*, ponemos el primer argumento antes de indicar la función. Esto nos permite encadenar funciones siempre que tengan el primer argumento en común. De este modo, todas las funciones de dentro de la *pipe* empezarán por el segundo argumento.

Tabla 9. La función `pipe(%>%)`

Clásica y *pipe*	`f(x, y) = x %>% f(y)`
Agrupación clásica	`arrange(filter(gapminder, continent == "Europe"))`
Agrupación con *pipe*	`gapminder %>% filter(continent == "Europe") %>% arrange(lifeExp)`

Fuente: elaboración propia.

Crear objetos para guardar las transformaciones

Es importante saber que, cuando transformamos un objeto con una función, esta no guarda los cambios en cuestión a menos que lo indiquemos expresamente. En la consola veremos cómo quedarían estos cambios, pero R en ningún caso nos modifica el objeto. Si queremos guardar las operaciones, tendremos que sobrescribir el objeto o crear uno nuevo con el símbolo `<-`. Por ejemplo, si queremos solo los datos de Alemania, podemos introducir `gapminder_germany <- filter(gapminder, country == "Germany")`. Por ahora, recomendamsos no sobrescribir los cambios sobre el mismo marco de datos.

Podemos añadir tantas funciones como queramos a la *pipe*. Si en el ejemplo quisiéramos añadir una función nueva, colocaríamos otra *pipe* después de la función *arrange* y abriríamos otra línea con la función nueva correspondiente. Durante este capítulo nos iremos dando cuenta poco a poco de la utilidad de la *pipe*. Por ahora, solo intentad retener que es una nueva manera de estructurar gramaticalmente un código en R.

3.1. Filtrar

La función `filter()` nos permite filtrar un subconjunto de las filas de un marco de datos sobre la base de una o varias condiciones lógicas. Supongamos que solo queremos ver las observaciones de un solo país, en este caso de Alemania. Escogemos `"Germany"`, filtramos por este país y pedimos observar solo las primeras seis filas con `head()`.

```
> gapminder %>%
   filter(country == "Germany") %>%
   head()
#  A tibble: 6 × 6
   country   continent   year   lifeExp   pop        gdpPercap
   <fctr>    <fctr>      <int>  <dbl>     <int>      <dbl>
1  Germany   Europe      1952   67.5      69145952   7144.114
2  Germany   Europe      1957   69.1      71019069   10187.827
3  Germany   Europe      1962   70.3      73739117   12902.463
4  Germany   Europe      1967   70.8      76368453   14745.626
5  Germany   Europe      1972   71.0      78717088   18016.180
6  Germany   Europe      1977   72.5      78160773   20512.921
```

En este ejemplo hemos filtrado solo por una condición lógica, pero la función `filter()` permite hacer filtros con varias condiciones. En la mayoría de los casos solo hará falta que pongamos una coma e introduzcamos un nuevo argumento con otra condición lógica. La coma hace de condición lógica AND, en que pedimos que nos devuelva solo los casos que cumplen las condiciones que hemos señalado. En la primera función del código siguiente, pedimos a R que nos devuelva los casos que cumplan a la vez dos requisitos: que sean del año 1987 y que el PIB per cápita sea superior a los 25.000 dólares (conseguiremos el mismo resultado si en lugar de una coma ponemos el símbolo `&`).

Igual sencillo (=) e igual doble (==)

Como veis, estamos trabajando con dos tipos de igualdad. Para diferenciarlos, tenemos que pensar que el igual doble (==) solo lo utilizaremos cuando estemos comprobando las condiciones de un objeto, de

forma que nos creará, implícita o explícitamente, un vector lógico. En cambio, el igual sencillo (=) nos hace asignaciones de valores o resultados, como veremos más adelante.

También podemos filtrar los datos de otras maneras, como con la condición lógica OR (símbolo |). Como vemos, en la segunda función del código estamos filtrando por los datos que cumplan cualquiera de los dos requisitos: solo con que los datos sean iguales o superiores a 1982, o bien que el PIB per cápita sea superior a 25.000 dólares, ya se incluirá en la selección. Este filtro devolverá más casos que el anterior. En el tercer código, hemos separado los requisitos por el símbolo `&`, pero esta vez pedimos que nos enseñe todos los años excepto 1982 (con el símbolo de negación `!=`) para aquellos países con un PIB per cápita inferior o igual a 25.000 dólares. En la última función hemos incluido el símbolo `%in%` para indicar que queremos filtrar por un determinado número de categorías, que especificamos dentro de un vector.

Filtrar los datos perdidos

R nos filtra automáticamente los datos perdidos que hay en el vector que indicamos dentro de la función `filter()`. En el supuesto de que queramos conservar los datos perdidos, lo podemos indicar adentro de la misma función con `is.na()`, por ejemplo `filter(gdpPercap > 30000 |is.na(gdpPercap)`.

```
gapminder %>%
   filter(year == 1982, gdpPercap > 25000)
```

```
gapminder %>%
    filter(year >= 1982 | gdpPercap > 25000)


gapminder %>%
    filter(year != 1982 & gdpPercap <= 25000)


gapminder %>%
    filter(year %in% c(1982, 1987, 1992))
```

A la hora de filtrar, tenemos que tener en cuenta que los factores requieren una atención especial. Esto se debe al hecho de que, cuando filtramos los datos por medio de una variable que es un factor, R elimina los valores que hemos indicado, aunque no elimina ningún nivel de la variable. Es decir, si en `gapminder` filtramos por el continente «África», nos quedará un marco de datos con solo una sola categoría en la variable *continente*, pero, en cambio, si pedimos los niveles de la variable, veremos que continúa teniendo cinco.

```
gap_afr <- gapminder %>%
          filter(continent == "Africa")
> unique(gap_afr$continent)
Africa
Levels: Africa Americas Asia Europe Oceania
```

Para eliminar los niveles vacíos, tendremos que crear otra *pipe* después de la función `filter()` y añadir la función `droplevels()`. Hay que remarcar, finalmente, que podemos añadir tantas condiciones como queramos siempre y cuando no filtremos

más de la cuenta, puesto que en este caso R nos devolverá un marco de datos sin ninguna observación.

3.2. Ordenar

La función `arrange()` simplemente ordena las filas de manera ascendente o descendente a partir de los valores de una columna determinada. Por lo tanto, probablemente es la función que tiene menos misterio de todos los verbos que estudiaremos de la librería dplyr. Antes que nada, ¿sabríais intuir cómo está ordenado el marco de datos `gapminder`?

Ejercicio 1. ***Gapminder***

Imprimid el marco de datos `gapminder` e intentad deducir por qué columna está ordenado en primer lugar. ¿Los valores están ordenados en orden ascendente o descendente? ¿Y cuál es la columna que está ordenada en segundo lugar?

Si hemos hecho el intento, habremos podido comprobar que `gapminder` está ordenado primero por países por orden alfabético descendente, puesto que empieza por todas las observaciones que tiene de Afganistán, sigue por Albania, etc. En segundo lugar, el marco de datos está ordenado por año de manera ascendente, puesto que empieza con datos de 1952 y va subiendo hasta 2007. Si queremos cambiar cómo está ordenado `gapminder` y, por ejemplo, nos gustaría ver en primer lugar los países con la esperanza de vida más baja, teclearemos el siguiente código:

```
gapminder %>%
    arrange(lifeExp)
```

Vemos que la mayoría de casos registrados con la esperanza de vida más baja son del inicio de la recogida de datos en 1952 y 1957. Sin embargo, los datos de esperanza de vida más bajos que encontramos en toda la base de datos son los de Ruanda en 1992, en la época de la guerra civil y el genocidio.

¿Cómo sería si miráramos los datos de la esperanza de vida a partir del país con el índice más alto? Pues tenemos que especificar que queremos que nos devuelva la tabla en orden descendente con la función `desc()`:

```
gapminder %>%
    arrange(desc(lifeExp))
```

El país que ha registrado una esperanza de vida más alta es Japón, con 82,6 años en 2007, seguido de Hong Kong. Como vemos, el tercer país de la tabla vuelve a ser Japón, de forma que el hecho de que tengamos mezcladas esperanzas de vida de años diferentes hace que se nos dupliquen países en la tabla.

En este punto, llega la parte más creativa de dplyr y donde podremos empezar a explotar realmente las virtudes de la *pipe*, puesto que gracias al símbolo `%>%` podremos trabajar con varias funciones a la vez. Supongamos que solo nos interesa ver los datos más recientes de países que tienen la esperanza de vida alta:

1) Filtramos las observaciones que cumplan los dos requisitos siguientes: que sean del año 2007 y que tengan una esperanza de vida superior a los 78 años.

2) Creamos una nueva *pipe* y pedimos que nos ordene los resultados por esperanza de vida en orden descendente.

3) Como podemos establecer un orden para tantas columnas como queramos, pediremos que, en caso de tener dos países con la misma esperanza de vida, primero nos muestre el que tiene un PIB per cápita superior.

```
gapminder %>%
    filter(year == 2007, lifeExp > 78) %>%
    arrange(desc(lifeExp), desc(gdpPercap))
```

Como veis, mediante la *pipe* hemos encadenado dos funciones:

1) Primero, hemos indicado el argumento `x`, en este caso el objeto *gapminder*, seguido de la primera *pipe*.

2) En segundo lugar, hemos introducido la primera función con los argumentos correspondientes seguido de otra *pipe* y la segunda función con sus argumentos correspondientes.

3.3. Seleccionar

Hasta ahora, hemos visto dos funciones del paquete dplyr que transforman las filas (las filtran o las ordenan). La próxima función es `select()`, una función parecida a `filter()`, pero que, en lugar de reducir el número de filas, lo que hace es reducir el número de columnas. Esta función es especialmente útil cuando tenemos marcos de datos con un número enorme de variables. Para poder trabajar mejor el marco de datos, nos interesará reducir las columnas a una cantidad más manejable.

El marco de datos `gapminder` no tiene un número excesivo de columnas, pero su número es suficiente para poder practicar los diferentes usos de la función `select()`.

1) El primer código que vemos a continuación nos devuelve un marco de datos con solo tres columnas: *country*, *year* y *lifeExp*.

2) En el segundo código, el símbolo negativo nos sirve para excluir una variable y, por lo tanto, nos devolverá todo el marco de datos excepto la columna *pop*.

3) En el tercer caso, seleccionamos desde la variable *country* hasta *year* y, además, la variable *gdpPercap*.

4) En el último caso, hemos dado una utilidad un poco distinta a `select()`, puesto que lo único que hemos hecho es indicar un orden de columnas diferente al orden original.[3]

```
gapminder %>%
   select(country, year, lifeExp)


gapminder %>%
   select(-pop)


gapminder %>%
   select(country:year, gdpPercap)
```

3. También podemos hacer la selección por números en lugar de por nombres de variables. Por ejemplo, `select(1:3)` nos seleccionará de la primera columna a la tercera.

```
gapminder %>%
    select(country, year, continent, gdpPercap, lifeExp, pop)
```

Dentro de `select()` podemos incluir funciones, algunas resumidas en la tabla 10, que nos ayudarán en la selección de las variables que queremos retener. Imaginemos que queremos cambiar de orden algunas columnas y situarlas al principio, pero mantener el resto; entonces, podemos utilizar `everything()` (por ejemplo, `select(gapminder, continent, year, everything())`).

Tabla 10. Funciones adicionales con `select()`

`everything()`	Muestra el resto de columnas.
`starts_with()`	Empieza con unas columnas determinadas.
`ends_with()`	Acaba con unas columnas determinadas.
`contains()`	Contiene unas columnas determinadas.

Fuente: elaboración propia.

3.4. Mutar

El paquete dplyr también incorpora la función `mutate()`, que permite transformar los valores de una variable o crear nuevas variables a partir de datos ya existentes. Cuando mutamos una columna, podemos escoger dos opciones como primer argumento:

1) Si lo primero que especificamos dentro de la función es el nombre de la columna que queremos transformar, R nos sobrescribirá los valores en la misma columna;

2) Si, en cambio, indicamos el nombre de una columna nueva que aún no existe en el marco de datos, R nos mantendrá la ori-

ginal y nos creará una nueva con la transformación que hemos indicado.

Veamos un ejemplo de cada una y empecemos por la primera opción, que implica transformar los datos de una columna. En la variable *pop* de `gapminder`, los datos contienen muchos números. Supongamos que nos gustaría visualizar los datos de una manera más simple y tener las cifras de la variable en millones. En otras palabras, que si un país tiene 7.500.000 de habitantes pasáramos a ver este valor en 7,5 (en R el separador será un punto, no una coma). Lo que tendremos que hacer es lo siguiente:

```
gapminder %>%
    mutate(pop = pop / 1000000)
```

Como en este primer caso indicamos en la función que *pop* = *pop*, R entiende que tiene que sustituir los datos originales de la columna *pop* por los nuevos datos que hemos creado con la operación. Con la tabla que genera el código anterior, veréis cómo R ha dividido entre un millón todos los valores de la columna *pop*, de forma que ahora vemos los datos en millones. En este proceso, por lo tanto, R no ha creado ninguna variable.

Si lo que queremos es mantener la columna original y crear una nueva a partir de la transformación mediante `mutate()`, indicaremos como primer argumento un nombre diferente en cualquiera de las columnas que conforman el marco de datos. En este ejemplo, crearemos la variable *gdp*, que indica el PIB total de cada país. Como en el marco de datos tenemos los datos de población y los de PIB per cápita, para saber el PIB solo hará falta que multipliquemos las variables *gdpPercap* y *pop*. Como ninguna otra columna del marco de datos tiene el nombre de *gdp*,

R interpretará que tiene que incorporar una nueva variable en el marco de datos que hasta ahora no existía.

```
> gapminder %>%
   mutate(gdp = gdpPercap * pop) %>%
   head()
# A tibble: 6 × 6
   country      continent  year   lifeExp  pop       gdpPercap  gdp
   <chr>        <fct>      <int>  <dbl>    <int>     <dbl>      <dbl>
1  Afghanistan  Asia       1952   28.8     8425333   779.       6567086330.
2  Afghanistan  Asia       1957   30.3     9240934   821.       7585448670.
3  Afghanistan  Asia       1962   32.0     10267083  853.       8758855797.
4  Afghanistan  Asia       1967   34.0     11537966  836.       9648014150.
5  Afghanistan  Asia       1972   36.1     13079460  740.       9678553274.
6  Afghanistan  Asia       1977   38.4     14880372  786.       11697659231.
```

Exprimimos algo más las funciones de dplyr y las *pipe* con todo lo que hemos aprendido hasta ahora. En el código siguiente, hemos filtrado por el año 1952 y el continente asiático. Como solo nos quedará una sola categoría en las variables *year* y *continent*, hemos eliminado estas columnas con la función `select()`. Seguidamente, hemos mutado los datos, primero creando la variable *gdp* y después hemos calculado el porcentaje sobre el PIB total de las filas filtradas con `gdp / sum(gdp)`. Finalmente, ordenaremos los resultados por la columna *gdp*, acabada de crear, en orden descendente, de forma que los valores más altos nos queden en parte superior de la tabla. En este código hemos creado el objeto `gap_asia`. Una vez lo hayamos creado, imprimiremos el objeto para ver el resultado.

Calcular porcentajes

Anotad bien la fórmula `x / sum(x)`, puesto que más adelante nos puede ser de mucha utilidad para calcular proporciones con nuestros datos.

```
gap_asia <- gapminder %>%
   filter(year == 1952, continent == "Asia") %>%
   select(-continent, -year) %>%
   mutate(gdp = gdpPercap * pop,
          perc_gdp = gdp / sum(gdp)) %>%
   arrange(desc(gdp))
```

Tened en cuenta que el orden en que establezcamos las diferentes funciones puede determinar la tabla resultante. Si primero hubiéramos mutado y después filtrado los datos, la nueva columna de porcentajes nos habría calculado los porcentajes sobre el total de la base de datos, no sobre el total de las columnas filtradas. En cambio, como hemos mutado después de filtrar, la mutación nos la ha hecho solo para los países del continente asiático en 1952. También podéis comprobar cómo podemos llamar a objetos acabados de crear. Por ejemplo, podemos ordenar por el PIB porque, justo antes, hemos creado esta columna con `mutate()`.

3.5. Resumir

La función `summarize()` nos resume en un solo valor los datos de una columna a partir de una determinada operación matemática. Su funcionalidad es parecida a `mutate()`, pero, en

lugar de trabajar en horizontal creando nuevas columnas, trabaja en vertical, transformando varios datos de una misma columna en una sola. En el objeto `gap_asia` tenemos una tabla con la esperanza de vida, la población, el PIB per cápita, el PIB y el porcentaje sobre el PIB del continente para los países del continente asiático en 1952. Ahora intentaremos pedir resúmenes de este marco de datos. Calcularemos:

1) el total de población en el continente,
2) la media del PIB per cápita,
3) la proporción de observaciones con un PIB per cápita superior a la media,
4) el número de países con una esperanza de vida superior a los 60 años, y
5) el número de observaciones que tenemos en la muestra.

Dentro de la función `summarize()`, separados por comas, hemos creado cinco sumarios:

1) *tpop* nos suma la población de todos los países filtrados,
2) *mean_gdpcap* nos calcula la media,
3) *cerca* nos calcula la proporción de países que se encuentran por encima de la media,
4) *esp* nos devuelve el número de países con una esperanza de vida superior a los 60 años, y
5) *count* recuenta las observaciones filtradas.

Calcular proporciones y casos

Las funciones `mean()` y `sum()`, aparte de sacar la media y sumar, también son muy útiles para otras cosas. Si pedimos un vector lógico

dentro de `mean()`, la función nos calculará la media de los valores que son verdad y los que son falsos, devolviéndonos, al fin y al cabo, una proporción de qué porcentaje de observaciones son verdad. Si pedimos un vector lógico dentro de `sum()`, sumaremos los casos en que la operación lógica sea verdad.

```
> gap_asia %>%
    summarize(tpop = sum(pop),
              mean_gdpcap = mean(gdpPercap, na.rm = TRUE),
              prop = mean(gdpPercap > mean_gdpcap),
              esp = sum(lifeExp > 60),
              count = n())
# A tibble: 1 x 5
    tpop         mean_gdpcap   prop        esp         count
    <int>        <dbl>         <dbl>       <int>       <int>
1   1395357351   5195.         0.0909      4           33
```

R nos ha devuelto un sumario para cada operación que hemos pedido. La población total era de unos 1.400 millones de habitantes en 1952 en Asia. La riqueza mediana por habitante era de unos 5.200 dólares y sabemos que solo un 9 % de los países del continente estaban por encima de esta media. La función `n()`, sin nada dentro del paréntesis, ha hecho el recuento de 33 observaciones (en este caso, países asiáticos en 1952), de los cuales solo cuatro tenían una esperanza de vida superior a los 60 años.

Sacar datos perdidos del sumario

Si nos fijamos en el código de la media, veremos que hemos introducido el argumento `na.rm = TRUE` (significa *NA remove*, eliminar valores

perdidos). Es importante conocer este argumento y utilizarlo cuando pedimos estadísticos descriptivos, puesto que, en cualquier operación, excluirá los datos perdidos cuando queramos obtener un sumario.

Al contrario, por defecto, este argumento es falso (`na.rm = FALSE`) de forma que, en el caso de haber un solo dato perdido en la columna, R nos hubiera devuelto en la tabla el valor *NaN (Not a Number)*.

3.6. Agrupar

Los datos que nos ofrece `summarize()` son muy genéricos y para nosotros solo tienen un valor bastante significativo si podemos compararlos con otros datos. Supongamos que queremos resumir los datos del código anterior para cada continente. Una opción, lenta y farragosa, sería aplicar la función `filter()` para cada continente y generar un sumario diferente cada vez. Primero, filtraríamos por Asia y haríamos un sumario, después por África, luego por Europa, etc. Así, tras un buen rato, acabaríamos teniendo los resúmenes por cada categoría. Una opción más rápida es mediante `group_by()`, que agrupa los resúmenes de `summarize()` según los valores de una variable categórica o una numérica discreta. Si agrupamos, por ejemplo, por la variable *year*, R nos devolverá un sumario para cada valor de la variable *year*.

La función `group_by()` por ella misma no tiene ningún efecto sobre la tabla, sino que se tiene que aplicar en combinación con otra función. Normalmente, la combinaremos con `summarize()`, pero también se puede combinar con las otras funciones de dplyr .

En el siguiente código, aprovecharemos partes de los que hemos pedido anteriormente, pero en lugar de filtrar por un con-

tinente determinado, pediremos a R que nos agrupe los sumarios por continente. Lo que haremos será poner la función `group_by()` con el nombre de la variable que queremos que nos agrupe los sumarios. R nos devuelve un sumario para cada continente.

Ejercicio 2. Agrupar por varias variables

Podemos agrupar con la función `group_by()` por varias variables. Intentad sacar el filtro del año y añadidlo al agrupamiento con `group_by(continent, year)` para que también agrupe la tabla a la vez por continente y por año. Así podréis ver una evolución temporal para cada continente. Probad también a poner los datos de forma inversa para ver cómo cambia la distribución de la tabla: `group_by(year, continent)`.

```
> gapminder %>%
    filter(year == 1952) %>%
    group_by(continent) %>%
    summarize(tpop = sum(pop),
              mean_gdpcap = mean(gdpPercap, na.rm = TRUE),
              prop = mean(gdpPercap > mean_gdpcap),
              esp = sum(lifeExp > 60),
              count = n())
#  A tibble: 5 x 6
   continent   tpop        mean_gdpcap  prop     esp    count
   <fct>       <int>       <dbl>        <dbl>    <int>  <int>
1  Africa      237640501   1253.        0.327    0      52
2  Americas    345152446   4079.        0.24     6      25
3  Asia        1395357351  5195.        0.0909   4      33
4  Europe      418120846   5661.        0.433    23     30
5  Oceania     10686006    10298.       0.5      2      2
```

La función `group_by()` nos hace una agrupación previa de los datos antes de que `summarize()` aplique los cálculos pertinentes. R nos ha devuelto un marco de datos en que cada observación, tal como hemos indicado, es un continente. Vemos que Asia era el continente con más población en 1952, mientras que el PIB per cápita en Oceanía era el más alto (aunque solo hay dos países en el recuento). Europa era el continente con más países con una esperanza de vida superior a los 60 años.

Tenemos que tener en cuenta que la función `group_by()` se aplica solo en variables discretas que contengan un número suficiente de observaciones en cada grupo (como país o continente, que son categóricas, pero asimismo año, que también es discreta y tiene suficientes observaciones para cada valor de la variable). Si aplicamos la función a variables numéricas continuas, lo más probable es que nos acabe devolviendo el mismo marco de datos.

3.7. Recapitulando dplyr

Para cerrar esta sección, os ofrecemos un ejemplo de cómo las seis funciones diferentes de dplyr que hemos estudiado se pueden combinar en un solo código. En primer lugar, informamos a R de cuál es nuestra *x* y abrimos una *pipe*. A continuación, filtramos por los datos del 2007 y mutamos los datos para crear la nueva columna *gdp*. Seguidamente, agrupamos los datos por continente y pedimos un sumario del país con más población, del país con menos población, la diferencia entre el país con más población y el país con menos población y un recuento. Para acabar, ordenamos los datos en orden descendente en la nueva columna *diff* que nos aparecerá con el sumario y seleccionamos un orden diferente de las columnas, de forma

que nos aparezca primero el continente, después la variable *diff* y, a continuación, el resto de variables.

```
gapminder %>%
    filter(year == 2007) %>%
    mutate(gdp = gdpPercap * pop) %>%
    group_by(continent) %>%
    summarize(max_pop = max(pop),
              min_pop = min(pop),
              diff = max_pop - min_pop,
              count = n()) %>%
    arrange(desc(diff)) %>%
    select(continent, diff, everything())
```

Evidentemente, un código con más operaciones no es necesariamente un código mejor. Un buen código es simplemente aquel que nos ayuda a transformar los datos de manera que los resultados respondan a nuestras preguntas. Y, para ello, normalmente no nos hará falta utilizar cada vez las seis funciones de dplyr.

4. Visualizar

La visualización es una parte importantísima en el trabajo de un analista de datos, puesto que todo el trabajo llevado a cabo en el proceso de análisis también se tiene que poder comunicar por medio de gráficos atractivos y fáciles de entender. Esto quiere decir que parte de la información que hemos conseguido generar mediante varias funciones del paquete dplyr se puede quedar en nada si no somos capaces de trans-

mitirla al gran público. Demasiada información en un gráfico, o un gráfico que no explica lo bastante bien los datos con que estamos trabajando, puede dificultar la legibilidad y la comprensión de todo nuestro trabajo. Es por eso por lo que tenemos que ser especialmente cuidadosos a la hora de mostrar el producto final.

Seguramente, ya conocemos algunas funciones de base de R como `plot()` o `boxplot()`, que nos permiten visualizar los datos de una manera rápida y exploratoria. En esta sección aprenderemos a crear visualizaciones más atractivas de los datos, que nos sirvan no solamente para explorar, sino también para comunicar y presentar nuestros hallazgos. El paquete que utilizaremos es ggplot2, un paquete de gráficos muy potente pensado para poder representar estéticamente los datos a partir de una gramática implícita formada por varias capas de sintaxis.

La estructura basada en diferentes capas del paquete ggplot2 conlleva una gran ventaja para trabajar con código, puesto que nos permitirá aplicar siempre la misma estructura independientemente del gráfico que queramos visualizar. A continuación, describiremos siete capas, aunque hay algunas más. Para crear un gráfico mediante ggplot2 necesitamos siempre, como mínimo, las tres primeras capas. Lo primero que tendremos que hacer es indicar la función `ggplot()`. Dentro de esta función, indicaremos las dos primeras capas.

1) En la primera capa, indicaremos el marco de datos.

2) En la segunda capa, los estéticos (cómo se dispondrán las variables en el gráfico).

3) Seguido del signo + indicaremos la tercera capa, que es la geometría (qué figura geométrica representará los datos).

La estructura básica de cualquier visualización con ggplot2 la encontramos resumida en la tabla 11. Las primeras tres capas (base de datos, estéticos y geometría) serán siempre necesarias para producir una visualización, mientras que las otras cuatro que explicaremos en este capítulo (*facet*, escalas, coordenadas y temas) son opcionales. Tenemos que tener en cuenta que, a medida que queramos crear gráficos más sofisticados, nos será más necesario dominar todas las capas de ggplot2.

Para saber más sobre ggplot2

Podéis ver un ejemplo de las posibilidades de ggplot2 en este enlace (http://www.ggplot2exts.org/gallery/) o bien en la página web oficial de Tidyverse (https://ggplot2.tidyverse.org/reference/). También podéis consultar los libros de Chang (2012) y Grolemund y Wickham (2016). Para trabajar con el paquete, nos será muy útil este resumen (https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf) preparado por los creadores de RStudio.

Tabla 11. Estructura de capas de ggplot2

`ggplot(marco de datos, aes(x, y, otros estéticos)) +`
`geometría() +`
`facet() +`
`escalas() +`
`coordenadas() +`
`temas()`

Fuente: elaboración propia.

Antes de empezar a repasar la estructura de capas, es muy importante remarcar cuatro aspectos fundamentales de ggplot2:

1) Fijaos en que el paquete se llama ggplot2, pero la función para crear gráficos es `ggplot()` sin el '2' final.

2) La función `aes()` abre siempre la segunda capa y los estéticos se sitúan dentro del paréntesis.

3) A partir de la segunda capa, ggplot2 une las otras capas mediante el símbolo +. Olvidarnos de él es bastante frecuente y nos llevará a mensajes de error en la consola.

4) Finalmente, tened también en cuenta que, en muchas ocasiones, llamaremos a una geometría determinada, pero no tendremos que poner nada dentro del paréntesis, como `geom_point()` o `geom_abline()`, puesto que estaremos utilizando implícitamente los argumentos por defecto asociados a la geometría.

Errores comunes con ggplot2

Cuando trabajamos con ggplot2, también es muy frecuente el error de describir la *x* y la *y* del gráfico sin ponerlas al final de la función de estéticos (por ejemplo, `aes(x = var1, y = var2)`), como también el de no cerrar el segundo paréntesis después de estéticos. Recordad el hecho de empezar con `aes()`, el argumento de los estéticos, y que siempre habrá dos paréntesis que cerrar al final de esta línea: el de `ggplot()` y el de `aes()`.

4.1. Las primeras capas

4.1.1. Marco de datos

En la primera capa, simplemente introducimos el nombre del marco de datos que queremos utilizar. Ningún misterio. Solo hay que tener en cuenta dos consideraciones:

1) Antes de visualizar datos con ggplot2, normalmente trataremos antes el marco de datos con paquetes como dplyr.

2) ggplot2 se puede enlazar con la lógica de *pipes*. Por lo tanto, tal como mostramos al inicio de la sección anterior, podemos poner el marco de datos al comienzo del código, hacer las transformaciones pertinentes con dplyr y a continuación enlazar una *pipe* con la función `ggplot()`. Como vemos en el ejemplo siguiente, dado que ya hemos puesto el marco de datos al inicio del código, situaremos los estéticos como primer argumento. Recordad que a partir de aquí las capas irán conectadas mediante el símbolo +.

```
gapminder %>%

   filter(country == "Germany") %>%

   ggplot(aes(x = year, y = lifeExp)) +

   geom_line()
```

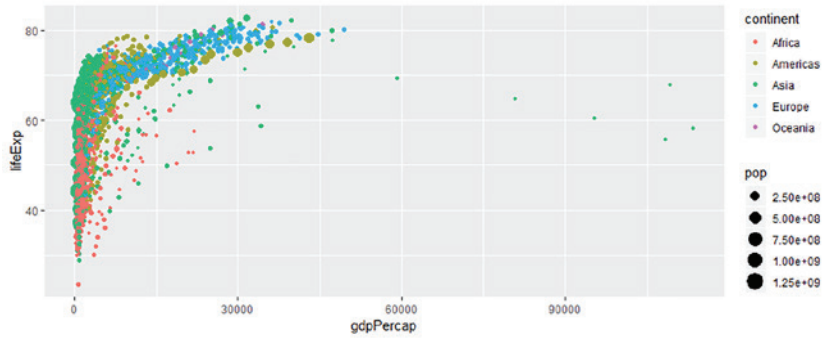
4.1.2. Estéticos

La segunda capa de sintaxis son los estéticos, representados por la función `aes()`, que indica la manera en que ggplot2 dispone estéticamente cada una de las variables que queremos representar. La variable que en este apartado definimos como *x* estará representada en el eje horizontal y la que definimos como *y* en el eje vertical. También tenemos la opción de incluir variables adicionales con otros tipos de estéticos, como el color, la forma o el tamaño. Por ejemplo, fijaos en el siguiente código:

```
gapminder %>%
    ggplot(aes(x = gdpPercap, y = lifeExp, color = continent,
    size = pop)) +
    geom_point()
```

Hemos representado este código en la figura 3. Como veis, hemos representado el PIB per cápita en el eje horizontal de las *x* y la esperanza de vida en el eje vertical de las *y*. Además de esta relación entre *x* e *y*, también especificamos que queremos visualizar dos dimensiones más: el color nos representará la variable *continente* (cada continente será un color), mientras que el tamaño nos representará la variable de *población* (el tamaño de la geometría variará según el valor de la población en cada caso). Cerraremos con un doble paréntesis e introduciremos el símbolo + para dar paso a la geometría, donde especificaremos a R que utilice un diagrama de dispersión para representar estos datos con la función `geom_point()`.

Figura 3. Visualización con cuatro estéticos



Fuente: elaboración propia.

Variables dependiente e independiente

Normalmente, pondremos la variable que consideramos independiente o explicativa en el eje horizontal y la variable que consideramos dependiente o explicada en el eje vertical.

En esta figura hemos utilizado un tipo de gráfico llamado *diagrama de dispersión*, que es una buena manera de visualizar una relación cuando tenemos variables numéricas en los ejes horizontal y vertical. Si bien los estéticos *x* e *y* serán siempre los que nos marcarán el tipo de gráfico que utilizaremos para representar los datos, tenemos que saber que la utilización de otros estéticos como el color o la forma estarán muy determinados por el tipo de variable que queramos representar. El color, por ejemplo, funciona muy bien con variables categóricas, como es el caso de la variable *continent*. En el gráfico podemos ver que el estético *color* asigna automáticamente un color diferente a cada continente y dispone una leyenda en la parte derecha del gráfico. El tamaño

(*size*) es más útil para representar variables continuas, de forma que un punto más grande nos representará un valor más alto de la variable. En la tabla 12 tenéis un resumen de los diferentes estéticos que podemos utilizar según el tipo de variable.

Tabla 12. Estéticos en función del tipo de variable

Estético	Código	Descripción
X	`x`	Eje horizontal.
Y	`y`	Eje vertical.
Color	`col / color`	Si la variable es categórica, nos dará un color diferente para cada categoría. Si es numérica, nos hará una gradación de color según el valor de la variable.
Tamaño	`size`	En variables numéricas, nos hará la geometría más grande o más pequeña (diámetros de los puntos, tamaño del texto o grosor de las líneas) según cada valor asociado.
Forma	`shape`	Restringido a variables categóricas. La representación por defecto es un punto, pero se puede representar con otras figuras como un triángulo, un cuadrado, una redonda, etc. Se tiene que introducir un valor del 1 al 25 (véase https://ggplot2.tidyverse.org/reference/scale_shape.html). A los valores del 1 al 20 solo se les puede cambiar el color. Del 21 al 25 se les puede cambiar el color y llenarlos.
Llenar	`fill`	Llena las geometrías de forma 21 a 25, polígonos u otras figuras geométricas.
Transparencia	`alpha`	Modifica la transparencia de la geometría en un valor que varía de 0 a 1.
Tipo de línea	`linetype / lty`	Modifica el tipo de línea. La que corresponde al valor 1 es continua y hasta 12 son tipos de líneas discontinuas.
Etiquetas	`labels`	Restringido a variables categóricas. Nos permite poner etiquetas en el gráfico.

Fuente: elaboración propia.

Para utilizar los estéticos, pondremos el código del estético seguido de un símbolo = y el nombre de la variable. Añadid, para hacer la prueba, `shape = continent` dentro de los estéticos del código anterior.

4.1.3. Geometría

La geometría es la tercera capa de la función `ggplot()`, separada de las dos primeras capas con el símbolo +. Aquí indicaremos el elemento visual que queremos usar para representar los datos: un histograma, un diagrama de cajas o, como en el ejemplo anterior donde tenemos dos variables numéricas, un diagrama de dispersión. La geometría está normalmente representada por *geom* seguido de una barra baja y del objeto geométrico en cuestión: `geom_point()`, por el diagrama de dispersión; `geom_histogram()`, por el histograma; `geom_bar()`, por el diagrama de barras, etc.

Geometrías diferentes en ggplot2

El paquete ggplot2 tiene hasta 37 geometrías diferentes. En este capítulo, para introducir ggplot2, solo veremos más a fondo la geometría del diagrama de dispersión. En el siguiente capítulo de estadística descriptiva, podremos aprender otras representaciones gráficas como el diagrama de línea, el diagrama de barras o el histograma.

Todas las geometrías están asociadas con unas características por defecto. Por ejemplo, si queremos representar un punto en el gráfico, por defecto será de color negro y tendrá un tamaño determinado. Es por eso por lo que muchas veces no habrá que especificar nada dentro del paréntesis de una geometría, puesto

que las características por defecto asociadas a la geometría ya nos serán adecuadas. Si, en cambio, queremos cambiar las características por defecto de un determinado gráfico, como el color de una línea o el grosor de un punto, lo tendremos que indicar expresamente dentro del paréntesis. Si queremos, por ejemplo, un diagrama de dispersión con cuadrados azules, más grandes que su tamaño por defecto, y con un toque transparente, tendremos que cambiar la forma (*shape*), el color, el tamaño (*size*) y la transparencia (*alpha*).

Ejercicio 3. Probar combinaciones

Reproducid el código siguiente probando varias combinaciones de color (`"red"`, `"light blue"`, `"green"`, etc.), tamaño (`0.1`, `1`, `4` o `10`), forma (`0`, `8`, `15` o `24`) y transparencia (`0.1`, `0.3` o `0.8`).

```
gapminder %>%
   ggplot(aes(x = gdpPercap, y = lifeExp, fill = continent)) +
   geom_point(color = "blue", size = 1, shape = 22, alpha = 0.6)
```

Tenemos que saber diferenciar entre los estéticos que representan una variable, que normalmente indicaremos en la segunda capa dentro de la función `aes()`, y los estéticos que representan un atributo del gráfico, que indicaremos en la tercera capa dentro de la geometría. En el ejemplo anterior, *fill* es un estético que nos ofrece información sobre la variable *continent*. Podemos hacer que *fill* pase a ser un atributo de la geometría si lo ponemos dentro de la geometría e indicamos, por ejemplo, con `fill = "yellow"`. En este caso, *fill* ya no nos da información de ninguna variable, sino que pasa a ser una manera de representar la geometría.

También es importante retener que los estéticos pueden ir tanto en la función de `ggplot()` como dentro de la geometría. Poner los estéticos dentro de la geometría nos será útil cuando tengamos varias geometrías que tienen todos los estéticos coincidentes. Para intentar aclarar un poco esta distinción entre la capa de estéticos y la de geometría, y entre estéticos y atributos, fijaos en el siguiente código:

Nombres de colores

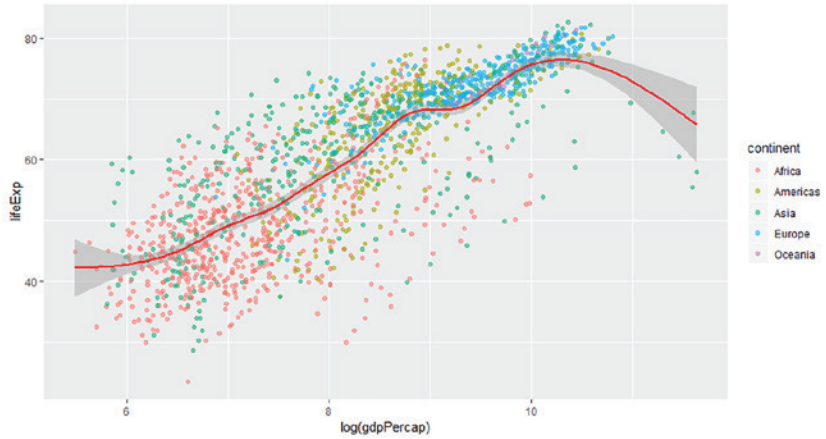
Podemos denominar los colores de múltiples maneras diferentes. En esta web (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) tenemos centenares de nombres de colores que podemos asignar a *col o fill*.

```
gapminder %>%
    ggplot(aes(x = log(gdpPercap), y = lifeExp)) +
    geom_point(aes(col = continent), alpha = 0.5) +
    geom_smooth(col = "red")
```

Dentro de `ggplot()` hemos asignado los estéticos generales del gráfico: todas las geometrías representarán en el eje de las *x* el logaritmo del PIB per cápita y, en el eje de las *y,* la esperanza de vida. Queremos visualizar este gráfico con dos geometrías: un diagrama de dispersión `geom_point()` y una línea de regresión local `geom_smooth()`. Hay, sin embargo, un estético, el color, que solo estará representado en la primera geometría y no afectará a la segunda. Esto quiere decir que veremos puntos de color diferente según el continente que representen, pero no veremos cinco líneas de color diferente según cada continente. El estético del color, pues, lo tendremos que poner dentro de

la función `geom_point()` con la función `aes()`. Fuera de los estéticos pediremos un atributo, que no representa ninguna variable, sino que es una característica de la geometría: todos los puntos tendrán un 50 % de transparencia. Seguidamente, con el símbolo +, añadimos `geom_smooth()`, que cogerá los estéticos generales *x* e *y*, pero no el específico de color para cada continente. En cambio, sí que pediremos ver una línea de color rojo en lugar de la azul que nos mostraría por defecto. En la figura 4 vemos el resultado.

Figura 4. Diagrama de dispersión y línea de regresión local



Fuente: elaboración propia.

Hasta ahora, solo hemos utilizado las tres primeras capas de ggplot2, suficientes para crear gráficos de todo tipo. Hay, sin embargo, otras capas que nos pueden ayudar a ampliar el número de variables que queremos representar, definir los títulos que daremos a cada una de las variables o cambiar otros aspectos visuales, como el tipo de letra o el color de fondo.

4.2. Otras capas

4.2.1. *Facet*

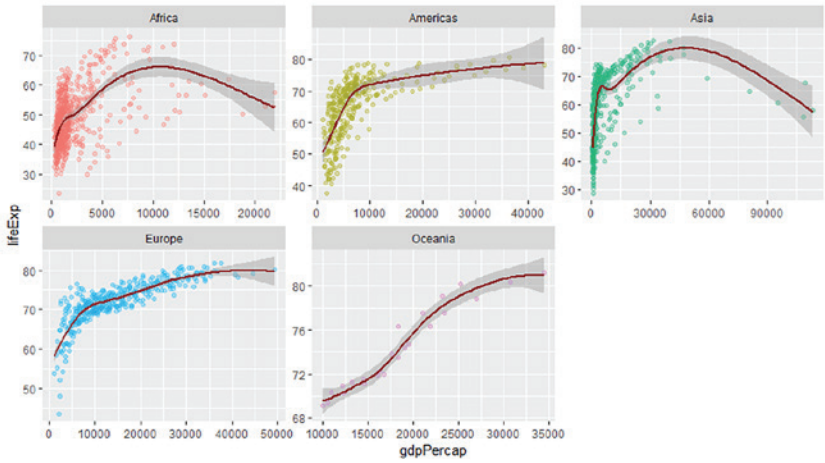
El nombre de la cuarta capa, *facet*, se podría traducir como 'aspecto' o 'dimensión' de un objeto. Es precisamente lo que hace esta función: la capa *facet* permite construir múltiples gráficos a partir de las diversas dimensiones de una variable categórica (Tufte, 1983). Por ejemplo, podemos ver la relación entre el PIB per cápita y la esperanza de vida según el continente.

```
gapminder %>%
    ggplot(aes(x = gdpPercap, y = lifeExp)) +
    geom_point(aes(col = continent), alpha = 0.3,
               show.legend = FALSE) +
    geom_smooth(col = "dark red") +
    facet_wrap(. ~ continent, scales = "free")
```

Dentro de las funciones *facet,* siempre utilizaremos la virgulilla, el símbolo ~, donde indicaremos si dividimos los gráficos por columnas o por filas. Si, como en el ejemplo, la variable categórica está situada a la derecha de la virgulilla, construiremos los *facets* por columnas. Si está a la izquierda, construiremos los *facets* por filas. En el lado contrario, pondremos un punto. Fijaos en que hemos mantenido el estético de color para el diagrama de dispersión, de forma que los puntos de cada *facet* salen pintados de un color diferente. Hemos decidido esconder la leyenda de los estéticos del diagrama con `show.legend = FALSE` porque era redundante. Por defecto, los ejes horizontal y vertical con las funciones *facet* nos aparecen con la misma escala, pero en nuestro caso hemos indicado que las escalas

sean libres con `scales = "free"`, de forma que cada gráfico tiene una escala diferente según sus valores (por ejemplo, en Oceanía el PIB per cápita llega a 35.000, mientras que en Asia sobrepasa los 90.000).

Figura 5. PIB per cápita y esperanza de vida en *facets* por continentes



Fuente: elaboración propia.

Otra variante de *facet* es `facet_grid()`. A diferencia de `facet_wrap()`, en este caso tenemos la opción de construir una parrilla de *facets* que compartan etiquetas y ejes de coordenadas. Esta función es muy útil cuando queremos añadir una variable categórica al *facet* de las filas y otra al *facet* de columnas. Como veis si reproducís el código siguiente, hemos creado un `facet_grid()` y hemos construido una parrilla con los años en las filas y el continente en las columnas.

```
gapminder %>%
    filter(year %in% c(2002, 2007)) %>%
    ggplot(aes(x = gdpPercap, y = lifeExp)) +
```

```
    geom_point(aes(col = continent), alpha = 0.3,
               show.legend = FALSE) +
    geom_smooth(col = "dark red") +
    facet_grid(year ~ continent)
```

Si comparáis las dos *facets*, veréis que `facet_wrap()` utiliza etiquetas y coordenadas en cada uno de los gráficos, mientras que `facet_grid()` construye una parrilla con un número muy reducido de etiquetas y coordenadas.

Estilos de *facet*

En este enlace (https://ggplot2.tidyverse.org/reference/facet_grid.html) podéis ver otros ejemplos de su funcionamiento.

4.2.2. Escalas

Por *escala* (*scale*) nos referimos a la manera de representar, modular y mostrar información de los diferentes estéticos del gráfico, en particular de los ejes horizontal y vertical. Siempre introduciremos las escalas en el mismo orden: primero, pondremos el nombre *scale*, seguido de guion bajo, el estético, otro guion bajo y el nombre de la escala. Por ejemplo, en el siguiente código modulamos y mostramos información nueva en los ejes *x* e *y*. En primer lugar, indicamos el código con el eje y el tipo de variable (*continuous* si es numérica y *discrete* si es categórica). Dentro de la función, tenemos varios argumentos para introducir información o modular el gráfico.

```
gapminder %>%
    ggplot(aes(x = gdpPercap, y = lifeExp)) +
    geom_point(aes(col = continent), alpha = 0.3) +
    geom_smooth(col = "dark red", se = FALSE) +
    scale_x_continuous(name = "PIB per cápita",
                       expand = c(0,0)) +
    scale_y_continuous(name = "Esperanza de vida",
                       limits = c(20, 80),
                       breaks = c(20, 40, 60, 80),
                       label = c("Jóvenes", "Adultos",
                                 "Mayores", "Viejos")) +
    ggtitle("PIB per cápita y esperanza de vida")
```

En este código hemos modificado primero las escalas del eje de las *x* indicando que es una variable numérica con `scale_x_continuous()`. Dentro de la función, hemos especificado los límites inferior y superior del gráfico, así como «PIB per cápita» como título de eje con *name*. También con *name*, hemos denominado el eje vertical «Esperanza de vida» y hemos establecido los límites superior e inferior en 20 y 80 años con el argumento límites. A continuación, hemos cortado el eje en cuatro partes con *breaks* y, con *label,* hemos asignado nombre a las etiquetas de cada corte. Finalmente, hemos dado un título general al gráfico con la función `ggtitle()`.

Para saber más

Encontraréis más información para variables numéricas en https://ggplot2.tidyverse.org/reference/scale_continuous.html y para variables categóricas en https://ggplot2.tidyverse.

org/reference/scale_discrete.html. También podemos definir las escalas del resto de estéticos, como el color, el tamaño o la forma.

4.2.3. Coordenadas

Esta capa controla las dimensiones del gráfico. Hasta ahora, todos los gráficos que hemos visualizado tienen dos dimensiones y se denominan *coordenadas cartesianas*. Una dimensión es horizontal, representada por el eje de las *x*, y la otra, vertical, representada por el eje de las *y*. Sin embargo, hay otros tipos de coordenadas como, por ejemplo, las coordenadas polares, que nos posibilitan visualizar diagramas circulares como el del código siguiente:

```
gapminder %>%
   filter(year == 2007) %>%
   group_by(continent) %>%
   summarize(population = sum(pop, rm.na = TRUE)) %>%
   ggplot(aes(x = 1, y = population, fill = continent)) +
   geom_col() + coord_polar(theta = "y")
```

Girar las coordenadas

Esta capa la usaremos muy poco. Quizás la función más útil es `coord_flip()`, que intercambia las dimensiones cartesianas del gráfico de forma que *x* pasa a estar representada en el eje vertical e *y,* en el eje horizontal.

4.2.4. Tema

Toda la parte visual que no tiene que ver con los datos se introduce en la capa de tema. Aquí es donde podemos cambiar el color de fondo, el tipo de letra de cada elemento, el tamaño de las letras o cualquier otro elemento del gráfico.[4] En este manual os sugerimos posibles temas que R tiene prediseñados. Por defecto, R siempre os mostrará `theme_gray()`, pero podéis cambiarlo si añadís un nueva capa a la función `ggplot()`, por ejemplo, `theme_classic()`, `theme_bw()` o `theme_dark()`. Podéis bajaros nuevos temas como el que utilizan *The Economist* o *The Wall Street Journal,* mediante el paquete *ggthemes* cargando el `library(ggthemes)`.

4.3. Combinando dplyr y ggplot2

Con todo, ahora ya estamos preparados para crear el código que nos permite construir el gráfico que hemos visto en la figura 1 de este capítulo.

```
gapminder %>%
    filter(country != "Kuwait",
           year %in% c(1952, 1972, 1992, 2007)) %>%
    ggplot(aes(x = gdpPercap, y = lifeExp, col = continent,
           size = pop)) +
```

4. Encontraréis más información de cómo cambiar cada uno de los elementos en la página de la función `theme` (https://www.rdocumentation.org/packages/ggplot2/versions/3.1.0/topics/theme).

```
    geom_point() +
    scale_x_log10() +
    facet_wrap(~ year) +
    labs(x = "PIB per cápita", y = "Esperanza de vida",
           col = "Continente", size = "Población") +
    ggtitle("Evolución del PIB per cápita y la esperanza de vida
    (1952-2007)")
```

Fijaos en que aquí tenemos hasta cinco dimensiones de los datos:

1) el PIB per cápita,
2) la esperanza de vida,
3) el continente,
4) la población, y
5) el año.

Fijaos en que hemos eliminado Kuwait porque se trataba de un caso extremo que nos distorsionaba la visualización de los datos. Con el símbolo `%in%` hemos filtrado cuatro valores de la variable *year*, que nos servirán para hacer un *facet* con ggplot2. Para visualizar mejor los datos, hemos escalado la *x* a logaritmo. Una buena manera de practicar es crear gráficos nuevos a partir de modificaciones de código que podéis hacer a partir de los ejemplos de este capítulo.

Ejercicios de autoevaluación

Para consolidar el aprendizaje, intentad hacer mentalmente el máximo de ejercicios posible, sin utilizar R.

1) Teclead el código para imprimir el siguiente marco de datos:

```
worldbankdata
```

2) Queremos ver las últimas siete filas y las primeras siete columnas del siguiente marco de datos:

```
imfdata
```

3) ¿Cuál es la función de dplyr que devuelve un resultado parecido a esta función?

```
str()
```

4) Teclead el código para visualizar la variable siguiente del marco de datos `imfdata`.

```
regions
```

5) Queremos ver los nombres únicos, ordenados de la Z a la A, de la siguiente variable del marco de datos `imfdata`.

```
regions
```

6) Visualizad un histograma de la siguiente variable del marco de datos `eurostat`.

```
social_expenditure
```

7) Transformad la función siguiente en una *pipe*.

```
length(unique(vector_categoric))
```

8) Detectad el error de la siguiente función:

```
filter(wbdata, country = "Japan")
```

9) Filtrad la siguiente variable para los años 1980, 1985 y 1990.

```
wbdata$year
```

10) Filtrad por países que tengan bombas nucleares (vector lógico) o un gasto militar superior al 4 % del PIB (vector numérico, escala de 0 a 1).

```
militarydata$nuclear / militarydata$gasto_pib
```

11) Ordenad el siguiente vector en orden descendente:

```
militarydata$gasto_pib
```

12) Seleccionad las primeras tres columnas del siguiente marco de datos y todas las columnas que acaben con la palabra *europe*.

```
tradedatabase
```

13) La columna siguiente está en escala de 0 a 1. Mutadla para ver los datos en porcentajes.

```
militarydata$gasto_pib
```

14) ¿Qué funciones hacen falta para resumir el siguiente marco de datos según los valores de una variable categórica?

```
eurostat
```

15) Agrupad los datos con la primera variable y resumid la media de los valores de la segunda.

```
militarydata$nuclear / militarydata$gasto_pib
```

16) Pedid un diagrama de dispersión con las variables siguientes como *x* e *y*.

```
militarydata$num_guerras / militarydata$gasto_pib
```

17) Cambiad el color de la geometría siguiente a rojo.

```
geom_smooth()
```

18) Poned transparencia del 50 % en la siguiente geometría:

```
geom_point()
```

19) Indicad cuál sería el argumento para llenar la siguiente geometría de color amarillo.

```
geom_bar()
```

20) Construid un `facet_wrap` por filas con la siguiente variable categórica:

```
militarydata$nuclear
```

Solucionario

1) `worldbankdata`

2) `tail(imfdata, 7)[1:7]`

3) `glimpse()`

4) `imfdata$regions`

5) `sort(unique(imfdata$regions), decreasing = TRUE)`

6) `hist(eurostat$social_expenditure)`

7) `vector_categoric %>% unique() %>% length()`

8) El igual (=) tendría que ser doble (==)

9) `wbdata %>% filter(year %in% c(1980, 1985, 1990))`

10) `militarydata %>% filter(nuclear == TRUE | gasto_pib > 0.04)`

11) `militarydata %>% arrange(desc(gasto_pib))`

12) `tradedatabase %>% select(1:3, ends_with("europe"))`

13) `militarydata %>% mutate(gasto_pib = gasto_pib * 100)`

14) `group_by()` y `summarize()`

15) `militarydata %>% group_by(nuclear) %>% summarize(mean(gasto_pib))`

16) `militarydata %>% ggplot(aes(x = num_guerras, y = gasto_pib) + geom_point()`

17) `geom_smooth(col = "red")`

18) `geom_point(alpha = 0.5)`

19) `geom_bar(fill = "yellow")`

20) `facet_wrap(nuclear ~ .)`

Glosario

`%>%` Símbolo *pipe* que replica el primer argumento de una función como argumento de las funciones siguientes (dplyr).

`%in%` Condición lógica que indica una selección de categorías.

`arrange()` Reordena las filas de un marco de datos (dplyr).

`boxplot()` En un diagrama de cajas, visualiza la relación entre una variable categórica y una variable numérica.

`coord_polar()` Cambia las coordenadas cartesianas por coordenadas polares (ggplot2).

`dim()` Muestra las dimensiones del marco de datos, primero las filas y después las columnas.

`droplevels()` Elimina los niveles vacíos de un factor.

`facet_grid()` Crea una parrilla de *facets* (ggplot2).

`facet_wrap()` Crea una parrilla de *facets* reduciendo los ejes y las coordenadas (ggplot2).

`filter()` Elimina las filas de un marco de datos (dplyr).

`geom_****()` Capa *de* ggplot2 que crea una geometría (ggplot2).

`ggplot()` Crea una visualización gráfica (ggplot2).

`ggtitle()` Muestra el título principal del gráfico (ggplot2).

`glimpse()` Muestra una estructura de los datos más limpia que `str()` y la adapta a las dimensiones de la consola (dplyr).

`group_by()` Agrupa observaciones de un marco de datos (dplyr).

`head()` Devuelve las seis primeras filas del marco de datos.

`hist()` Permite visualizar la distribución de una variable numérica.

`install.packages()` Instala librerías en R.

`labs()` Muestra las etiquetas del gráfico (ggplot2).

`library()` Muestra las librerías o carga una librería determinada.

`mutate()` Crea nuevas columnas o transforma las existentes con valores construidos a partir de datos (dplyr).

`names()` Muestra el nombre de las columnas de un marco de datos.

`plot()` Permite visualizar las frecuencias de una variable categórica o bien visualizar la relación entre dos variables numéricas.

`scale_*_****()` Prepara las escalas de un eje (ggplot2).

`search()` Muestra los paquetes cargados en R.

`select()` Elimina o reordena las columnas de un marco de datos (dplyr).

`str()` Muestra la estructura de los datos, con las variables en las filas, el tipo de variable y las primeras observaciones de cada variable.

`summarize()` Resume varias observaciones en una sola mediante una operación (dplyr).

`summary()` Resume una variable y devuelve un sumario específico según cada tipo de vector.

`tail()` Devuelve las seis últimas filas del marco de datos.

`unique()` Devuelve valores únicos de un vector.

Bibliografía

Babbie, E. R. (2013). *The practice of social research*. Wadsworth: Cengage Learning.

Chang, W. (2012). *R Graphics Cookbook*. Canadá: O'Reilly. <http://www.cookbook-r.com/graphs/>

Grolemund, G.; Wickham, H. (2016). *R for Data Science* [documento en línea]. Canadá: O'Reilly. [Fecha de consulta: 9 de julio de 2020]. <https://es.r4ds.hadley.nz/>

King, G.; Keohane, R. O.; Verba, S. (1994). *Designing Social Inquiry: Scientific Inference in Qualitative Research*. Princeton: Princeton University Press.

Tufte, E. (1983). *Visualization of Quantitative Information*. Michigan: Graphics Press.

Wilkinson, L. (1999). *The Grammar of Graphics*. Nueva York: Springer.

Capítulo III
Análisis univariante

1. Introducción

Tal como su nombre indica, el *análisis univariante* significa el análisis de una sola variable. La naturaleza de la variable que queremos estudiar determinará en buena parte los instrumentos que utilizaremos para el análisis univariante: si es una variable categórica, haremos un tipo de tratamiento, mientras que si es una variable numérica, realizaremos otro. Por este motivo, será muy importante que tengamos identificada la variable en R con el vector más apropiado.

En muchas ocasiones, el análisis univariante requiere utilizar más de una variable. Por ejemplo, quizás queremos observar una variable teniendo en cuenta los valores que toman otras variables del marco de datos. Esto puede llevar a confusión con el trabajo del análisis bivariante, que se ocupa de estudiar la relación entre dos variables. Por eso distinguiremos dos tareas cruciales del análisis de datos: la descripción y la explicación (King y otros, 1994). Este módulo se ocupa de la descripción. Por lo tanto, la utilización de dos variables se orienta a describir una variable y a comparar sus valores entre los diversos subgrupos de otra variable (Babbie, 2013). En ningún momento este proceso se orienta a sugerir si las dos variables en cuestión están o no asociadas entre ellas, que es trabajo del análisis bivariante.

En la primera parte del capítulo veremos diferentes representaciones gráficas que podemos utilizar en el análisis univariante.

El principal objetivo de una buena visualización es representar de manera clara cómo se distribuyen los valores en una variable. En gran parte, cómo representamos estos valores estará determinado por si la variable es de tipo categórico o numérico. En la segunda parte del capítulo pasaremos de la visualización a la cuantificación. Es decir, buscaremos cómo resumir la distribución de los datos en una variable de manera numérica, mediante uno o pocos números. Miraremos medidas de frecuencia, de centralidad, de dispersión y de localización.

2. Representaciones gráficas

En esta sección combinaremos funciones de los paquetes dplyr y ggplot2 para representar gráficamente la distribución de una variable del marco de datos `gapminder`. Para poder hacer estas representaciones, tendremos que estar suficientemente familiarizados con la gramática de ggplot2, puesto que en las páginas siguientes aplicaremos diversas de las geometrías que ofrece el paquete. Según el tipo de variable que queramos representar y la manera como la queramos representar, utilizaremos como principales formas de representación gráfica:

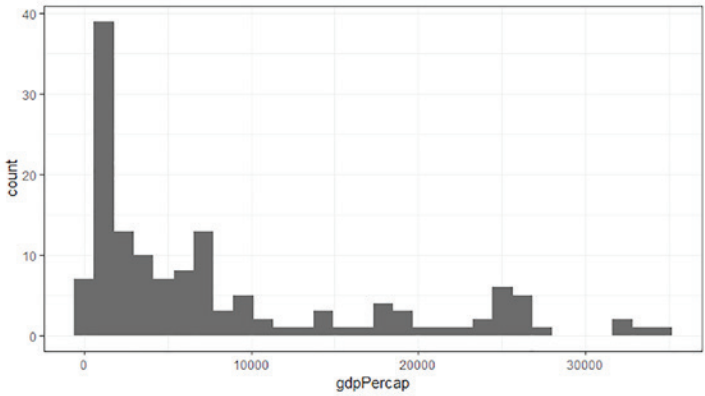
- el histograma,
- el diagrama de líneas,
- el diagrama de barras,
- el diagrama de cajas, y
- el diagrama de dispersión.

2.1. Histograma

El histograma nos permite visualizar con varias barras verticales la distribución de los valores de una variable numérica. Cada una de las barras representa un intervalo de valores de la variable y la altura de la barra corresponde al número de casos de cada intervalo. Hay que tener en cuenta, pues, que las barras del histograma no representan los valores numéricos originales de la variable, sino una función estadística que separa los datos en diferentes intervalos y las apila por columnas. Con R, representaremos esta figura con la función `geom_histogram()`.

En la figura 6 hemos creado un histograma para observar cómo están distribuidos los valores de la variable *gdpPercap* en 1992. Cada barra horizontal representa un intervalo de valores de la variable que indicaremos en el eje de las *x* dentro de los estéticos de *ggplot*. No hace falta que indiquemos el eje vertical de las *y*, puesto que siempre veremos el recuento de casos de cada columna (denominaremos *frecuencias* a la cantidad de casos). Por defecto, el histograma corta la variable en treinta intervalos de la misma anchura y representa la cantidad de valores que hay en cada intervalo con la estatura de las columnas. En nuestro caso, cada barra representa un intervalo de unos 1.300 dólares. Así, en el primer intervalo, encontraremos el número de países situados entre el intervalo de 0 y 1.300 dólares; en el segundo, aquellos entre 1.300 y 2.600 dólares; en el tercero, los de entre 2.600 y 3.900, y así sucesivamente.

Figura 6. Histograma del PIB per cápita



Fuente: elaboración propia.

```
gapminder %>%
    filter(year == 1992) %>%
    ggplot(aes(x = gdpPercap)) +
    geom_histogram() +
    theme_bw()
```

En el histograma que hemos creado vemos que el intervalo con más frecuencias es el segundo, probablemente situado entre unos 1.300 y 2.600 dólares per cápita al mes (llamaremos *moda* a la barra más alta, como veremos más adelante). Si miramos en el eje vertical el recuento de frecuencias, vemos que en este intervalo se sitúan cerca de cuarenta países. Alrededor de los 30.000 dólares, en cambio, no hay ningún caso.[1]

1. Porcentajes en el eje de las *y*: si en lugar de visualizar el número de frecuencias en el eje vertical preferís visualizar los porcentajes (también denominado *densidad*), podéis introducir el siguiente estético dentro de la geometría: `geom_histogram(aes(y= ..density..))`.

Cuando creáis un histograma, la consola os indica que ha creado treinta intervalos de datos como medida por defecto.[2] Podemos cambiar la anchura de los intervalos que representa cada columna de dos maneras:

1) La primera es indicando la cantidad de intervalos. Si, en lugar de los treinta por defecto queremos observar solo veinte, en la función añadiremos `geom_histogram(bins = 20)`. Podéis hacer varias pruebas introduciendo números diferentes para ver cómo cambia la visualización.

2) La segunda manera de cambiar los intervalos es indicando la anchura. Si queremos, por ejemplo, que cada uno de los intervalos nos represente 3.000 dólares per cápita, usaremos el argumento `binwidth` de la manera siguiente: `geom_histogram(binwidth = 3000)`. También podéis hacer diferentes pruebas para ver cómo cambia la visualización, y también cambiar el color de las barras indicando dentro de la geometría el color con el atributo *fill*, por ejemplo, `geom_histogram(fill = "dark blue")`.

Cómo se calcula el intervalo de cada columna

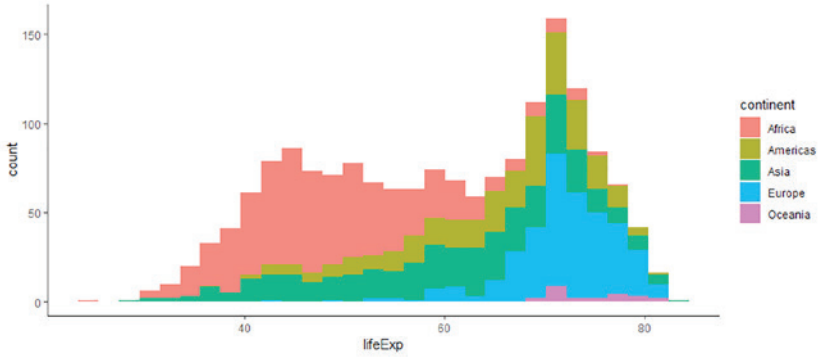
Para calcular la anchura de las columnas de un histograma R, mirad el rango de los datos. Es decir, el valor máximo (en este caso unos 40.000 dólares) y el valor mínimo (que se sitúa cerca de 0) y divide el rango en treinta intervalos de la misma longitud. Por ejemplo, el intervalo de todos los valores de la variable PIB per cápita es: `diff(range(gapminder$gdpPercap)) / 30`. Encontraréis más información

2. Sería lo mismo que introducir el argumento `bins = 30` dentro de la función del histograma. En la consola os tendría que especificar el retorno: *`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.*

sobre este proceso si miráis la ficha de los histogramas. Solo tenéis que entrar `?geom_histogram` en la consola y mirar la descripción de los argumentos.

El color también nos puede servir para introducir una nueva variable en el gráfico. Aunque el histograma se utilice para ver la distribución de una variable numérica, también podemos ver a la vez la distribución de estos valores en varias categorías. Observad la figura 7, donde visualizamos la distribución de la esperanza de vida en los diferentes continentes. En el código, hemos llenado las barras del histograma utilizando el argumento `fill = continent` como estético para indicar la variable categórica.

Figura 7. Histograma con variable categórica



Fuente: elaboración propia.

```
gapminder %>%
    ggplot(aes(x = lifeExp, fill = continent)) +
    geom_histogram(position = "stack") + theme_classic()
```

En el código, podéis observar cómo hemos añadido un argumento nuevo dentro de la geometría: `position = "stack"`.

Si sacamos este argumento, veréis que no cambia nada en la visualización. Esto se debe a que, por defecto, la posición de las columnas es *stack* (apilada). Esto significa que, en cada columna, apilará las categorías una sobre la otra. Alternativamente, podría poner las categorías una junto a la otra o superponerlas.

En general, `"stack"` será la mejor posición que tendremos para visualizar una variable categórica dentro de un histograma. Por otro lado, podemos probar a visualizar las columnas en otras posiciones: intentad, por ejemplo, cambiar la posición del anterior histograma a `"fill"`. Esta opción llena la barra hasta la parte superior del gráfico y es útil para ver los porcentajes de cada categoría en cada intervalo. Con la posición `"dodge"` veremos de lado las barras separadas por categorías,[3] mientras que con `"identity"` las columnas no se apilan, sino que conservan su identidad real. Esta última, sin embargo, se puede observar mejor con un diagrama de densidad.

Diagrama de densidad: la alternativa al histograma

Otra opción de análisis univariante para variables numéricas es el diagrama de densidad. Este diagrama resume los datos dibujando un área que nos permite tener una visión general de la forma que tiene la distribución. El diagrama de densidad es útil cuando queremos añadir información adicional al gráfico (porque nos lo deja muy limpio) y también para superponer varias áreas. Probad los siguientes códigos:

3. En los histogramas, la posición `"dodge"` es solo recomendable cuando tenemos pocos intervalos. Probad, por ejemplo, a combinar *dodge* con un número reducido de intervalos con la función `geom_histogram(position = "dodge", bins = 10)`.

```
gapminder %>%
   ggplot(aes(x = lifeExp)) +
      geom_density(fill = "orange", bw = 1)
gapminder %>%
    ggplot(aes(x = lifeExp, fill = continent)) +
    geom_density(alpha = 0.4)
gapminder %>%
   filter(year %in% c(1962, 1977, 1992, 2007)) %>%
   ggplot(aes(x = lifeExp, fill = factor(year))) +
      geom_density(alpha = 0.4, bw = 5)
gapminder %>%
    filter(year %in% c(1962, 1977, 1992, 2007)) %>%
    ggplot(aes(x = lifeExp, fill = factor(year))) +
       geom_density(bw = 5) +
       facet_wrap(~ year)
```

1. En el primer gráfico, hemos indicado la anchura (*bw*, que es la abreviatura de *binwidth*).
2. En el segundo, hemos representado varios continentes y hemos dado una transparencia de 0,4 a los diagramas de densidad. Con los mismos datos, podéis intentar hacer un histograma especificando dentro de la geometría `position = "identity"`.
3. En el tercero, hemos visualizado el PIB per cápita en cuatro años diferentes. Por eso hemos tenido que indicar a R que trate como categórica la variable *year* y la convierta en factor.
4. El último gráfico es parecido al tercero, pero en este caso hemos reducido la anchura de los intervalos y hemos separado los datos por *facets*. Probad a hacer este mismo gráfico con un histograma.

Fijaos en que en el eje vertical siempre vemos la densidad (porcentaje) y que el área que cubre cada distribución está por defecto normalizada, de forma que todas las áreas cubren exactamente la misma superficie.

Hemos visto que hay varias maneras de representar un histograma; por ejemplo, modificando la cantidad de intervalos del eje horizontal o añadiendo una variable categórica y cambiando la posición de las barras. No hay una regla concreta para decidir cuál es la mejor visualización, sino que la decisión estará determinada por lo que queramos comunicar, la cantidad de datos disponibles y su distribución. Lo mejor que podemos hacer es realizar varias pruebas y valernos del sentido común para decidir cuál es la mejor visualización.

2.2. Diagrama de líneas

El diagrama de líneas se utiliza normalmente para visualizar la tendencia de una variable numérica en el tiempo. En el eje horizontal de las *x*, situaríamos la variable temporal (en el caso de `gapminder` es el año, pero podrían ser meses, días...), mientras que, en el eje vertical de las *y*, situaríamos la variable numérica que queremos explicar. Con R, representaremos esta figura geométrica con la función `geom_line()`. En el código siguiente hemos hecho un sumario de la media de la esperanza de vida mundial agrupada por año. A continuación, hemos pedido un diagrama de líneas y hemos especificado varios atributos a la geometría:

- el color,
- el tamaño, y
- el tipo de línea.

Los tipos de vectores `date`

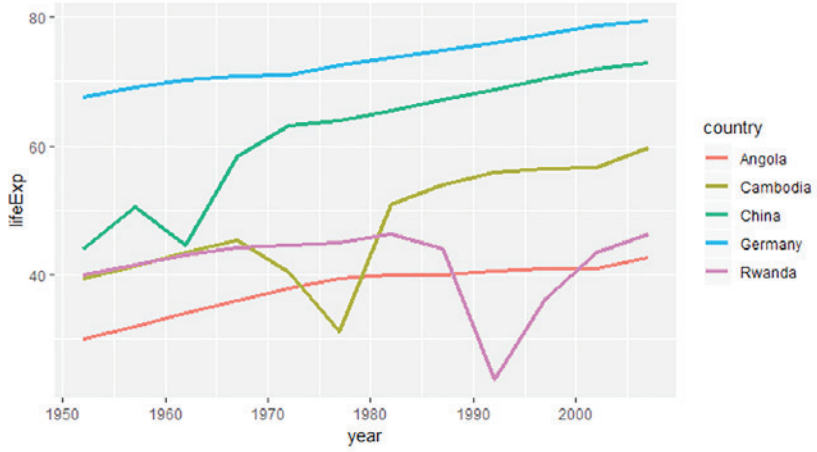
Los años se almacenan como vectores enteros, pero, si necesitamos operar con franjas más cortas de tiempo, nos puede interesar convertir el vector `date`, un tipo de vector especial que almacena valores numéricos, pero que los visualiza como si fueran una fecha.

Si imprimimos el código, observaremos que ha habido un aumento importante de la media de la esperanza de vida por país a lo largo del tiempo, puesto que ha pasado de estar por debajo de los 50 años en 1952 a superar los 65 años poco antes del año 2000.

```
gapminder %>%
    group_by(year) %>%
    summarize(mean_lifeExp = mean(lifeExp)) %>%
    ggplot(aes(x = year, y = mean_lifeExp)) +
    geom_line(col = "dark green", size = 1.2, lty = 5)
```

Hay centenares de combinaciones que podemos realizar con los atributos del diagrama línea. Probad a cambiarlos modificando el color, el tamaño o el tipo de línea dentro de la función de la geometría. También podemos hacer que cada línea represente una variable categórica, mediante el estético de color. En la figura 8, hemos filtrado cinco países de `gapminder` y hemos pedido un diagrama de líneas para ver la evolución temporal de su esperanza de vida. Hemos situado el año en el eje horizontal y la esperanza de vida en el eje vertical, y hemos incluido un estético adicional, el color, que representa la variable *country*. Cada color representará la evolución de la esperanza de vida en un país diferente.

Figura 8. Diagrama de línea con variable categórica



Fuente: elaboración propia.

```
gapminder %>%
    filter(country %in% c("Cambodia", "Germany", "China",
    "Angola", "Rwanda")) %>%
    group_by(year, country) %>%
    summarize(mean_lifeExp = mean(lifeExp)) %>%
    ggplot(aes(x = year, y = mean_lifeExp, col = country)) +
    geom_line(size = 1.2)
```

En esta figura podemos observar más detalles que con la que hemos generado con el código anterior. Fijémonos en que, en el caso de China, Camboya y Ruanda, se producen reducciones repentinas de la esperanza de vida, que marcan periodos históricos relevantes en estos países. El mejor estético para diferenciar diferentes líneas es el color, aunque también podemos utilizar el tipo de línea (*lty*) o una combinación de color y tipo de línea.

El estético *group* para agrupar sin estéticos

También podemos utilizar *group* en los estéticos para construir líneas que representen diferentes valores de una variable categórica. En este caso, sin embargo, no nos mostrará ninguna diferencia visual entre las líneas.

2.3. Diagrama de barras

El diagrama de barras no es un histograma.[4] Para diferenciarlos, diremos que el diagrama de barras toma como punto de partida una variable categórica, mientras que el histograma siempre nos mostrará una variable numérica. Cada una de las columnas del diagrama de barras representa los diferentes valores que toma la variable categórica y la altura de las barras representa unos determinados valores que queremos comparar entre categorías. Hay dos maneras de representar un diagrama de barras:

1) Que la altura de las barras represente el número de frecuencias de cada categoría. En este caso, utilizaremos la función geométrica `geom_bar()` para representar el diagrama. Fijaos en el código siguiente. En este caso, solo necesitamos indicar la

4. Hay dos diferencias muy importantes entre el histograma y el diagrama de barras. En primer lugar, las etiquetas del eje horizontal están situadas dentro de las barras del diagrama de barras porque representan la categoría; en cambio, en el histograma están situadas entre las barras porque representan la división entre intervalos. En segundo lugar, en el histograma no hay separación entre barras porque representan la continuidad de la variable. En cambio, como el diagrama de barras representa variables categóricas (no continuas), hay separación entre las barras.

variable categórica que representaremos en el eje de las *x*. En el eje vertical nos generará automáticamente un recuento de frecuencias en cada categoría.

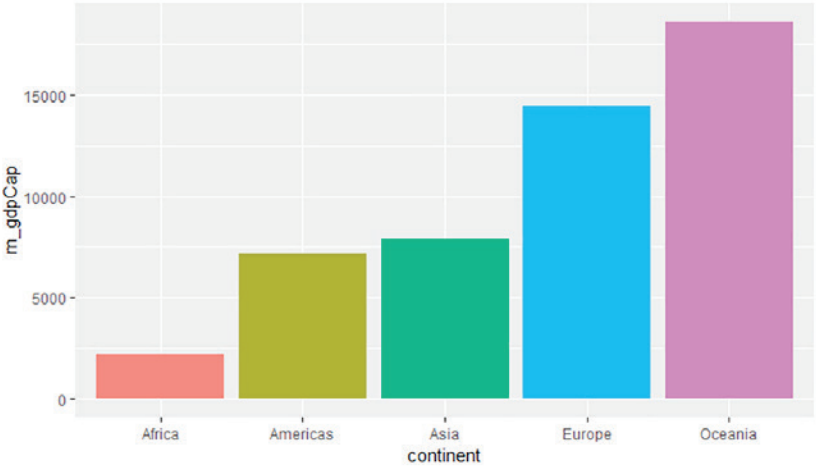
```
ggplot(gapminder, aes(x = continent)) + geom_bar()
```

2) Situando una variable categórica en el eje de las *x,* pero indicando que la altura de las barras nos muestre el sumario de otra variable, normalmente numérica, que indicaremos en el eje de las *y*. En este caso, utilizaremos la función geométrica `geom_col()` para representar el diagrama. Para poder conseguir esta visualización las funciones de dplyr `group_by()` y `summarize()` nos serán de gran ayuda, como vemos en el código asociado a la figura 9. En primer lugar, hemos agrupado los datos por continente y hemos pedido un sumario de la media del PIB per cápita en cada continente. De este modo, en el eje horizontal situaremos la variable categórica continente y en el eje vertical el sumario de la variable numérica agrupada por continente. Para visualizar con más claridad las diferencias entre columnas también hemos introducido el estético *fill* para que nos muestre un color por cada continente. Para evitar redundancias, hemos eliminado la leyenda.

Atributos que diferencian `geom_bar` y `geom_col`

La única diferencia entre estas dos maneras de representar diagramas de barras es la función estadística que tienen asociada por defecto. La función `geom_bar()` hace un recuento de frecuencias de cada categoría de *x,* puesto que tiene por defecto el atributo `stat = "count"`. La función `geom_col()` nos muestra la identidad de *y,* puesto que tiene por defecto el atributo `stat = "identity"`.

Figura 9. Diagrama de barras

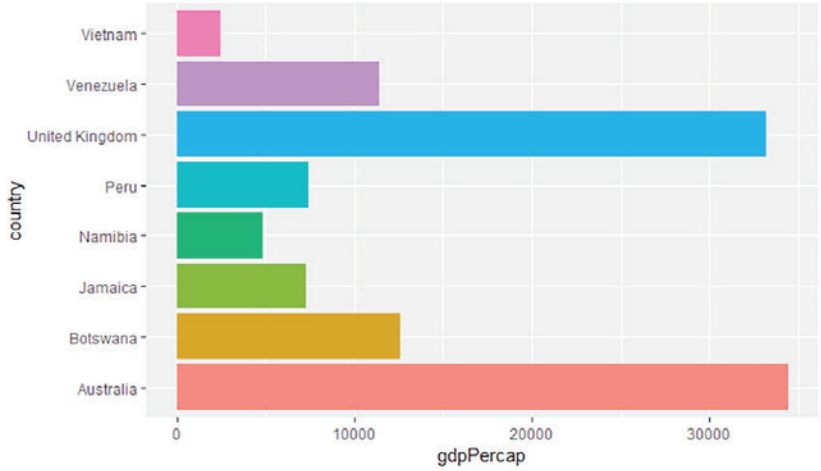


Fuente: elaboración propia.

```
gapminder %>% group_by(continent) %>%
    summarize(m_gdpCap = mean(gdpPercap)) %>%
   ggplot(aes(x = continent, y = m_gdpCap, fill = continent)) +
    geom_col(show.legend = FALSE)
```

Los diagramas de barras no tienen que ser necesariamente verticales. Nos podemos encontrar con que queramos representar un número relativamente elevado de categorías y no podamos distinguir bien sus nombres en el eje horizontal. También nos podemos encontrar con que tengamos pocas categorías, pero los nombres de cada categoría sean muy largos. En estos casos, podemos rotar los ejes del gráfico con la opción `coord_flip()`, que intercambia de posición *x* e *y*. En la figura 10 hemos rotado los ejes del diagrama de barras porque teníamos ocho categorías para mostrar.

Figura 10. Diagrama de barras horizontales



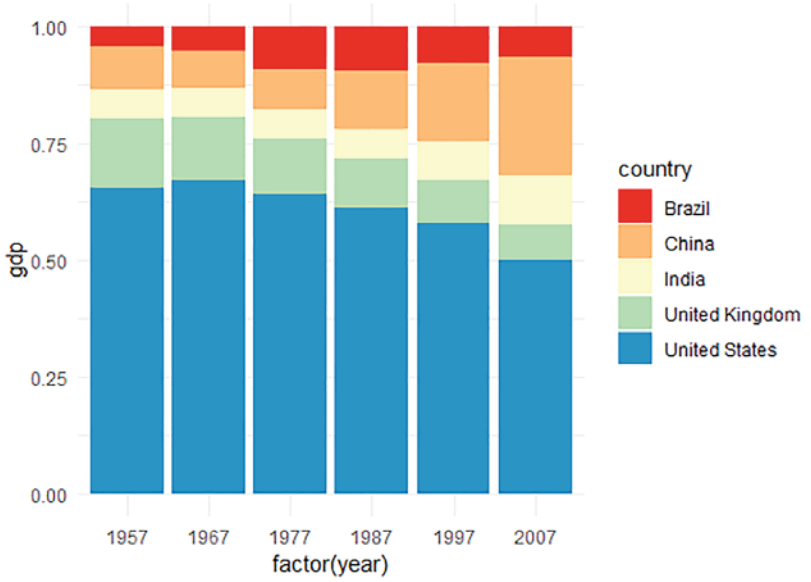
Fuente: elaboración propia.

```
gapminder %>%
    filter(year == 2007, country %in% c("Australia", "Botswana",
    "Namibia", "Jamaica", "Peru", "United Kingdom", "Venezuela",
    "Vietnam")) %>%
    arrange(desc(country)) %>%
    ggplot(aes(x = country, y = gdpPercap, fill = country)) +
    geom_col(show.legend = FALSE) +
    coord_flip()
```

En los ejemplos que hemos visto hasta ahora, en la altura de las barras del diagrama se representaban números absolutos. Es decir, las barras nos indicaban un determinado valor de una variable. Otra herramienta que tenemos permite que las columnas representen proporciones en lugar de valores. Así, podremos ver la dimensión relativa de cada valor dentro de una categoría concreta. En la figura 11 hemos querido observar la evolución

del PIB, en términos relativos, entre grandes potencias mundiales: Brasil, China, India, Reino Unido y Estados Unidos. En este caso, no nos interesa tanto cómo ha crecido el PIB en números absolutos, sino cómo ha variado la relación entre ellos. Esta visualización nos la permite el argumento `position = "fill"` dentro de la geometría.

Figura 11. Diagrama de barras con proporciones



Fuente: elaboración propia.

```
gapminder %>%
    filter(country %in% c("Brazil", "China", "India", "United
    Kingdom", "United States"), year %in% c(1957, 1967, 1977,
    1987, 1997, 2007)) %>%
    group_by(year, country) %>%
    summarize(gdp = gdpPercap * pop) %>%
```

```
    ggplot(aes(x = factor(year), y = gdp, fill = country)) +
    geom_col(position = "fill") +
    scale_fill_brewer(palette = 9, type = "div") + theme_mini-
    mal()
```

Fijaos en que esta opción nos permite representar tres variables. En el eje horizontal, hemos pasado la variable *año* a factor para que nos la considere categórica; el eje vertical nos muestra el PIB per cápita en términos relativos en vez de en términos absolutos, mientras que las barras están fraccionadas por países. Podemos observar que el tamaño relativo del PIB ha disminuido claramente en el caso de Estados Unidos y el Reino Unido, y ha aumentado en el caso de China y la India.

Evolución de los diagramas de barras

A la hora de representar el resumen de una variable numérica siguiendo los valores de una variable categórica, los diagramas de barras se sustituyen progresivamente por otras representaciones más sofisticadas que pueden representar mejor esta relación. Pensémoslo bien: la barra representa un solo dato, el sumario de una serie de valores de *y*, pero con la barra no podemos visualizar otra información útil como la distribución de los valores, dado que no sabemos el mínimo ni el máximo ni tampoco la cantidad de valores. Para hacernos una idea de la evolución de los diagramas de barras, imprimid el código siguiente, como muestra de un gráfico que representa exactamente los mismos valores (y mucha más información) que la figura 9.

```
gapminder %>%
   ggplot(aes(x = continent, y = gdpPercap, col = continent)) +
```

```
    geom_point(alpha = 0.2, position = position_jitter(width =
    0.2), show.legend = FALSE) +
    stat_summary(fun.y = mean, geom = "point", col = "black") +
    theme_light()
```

Cuando utilizamos el estético *fill* para representar varios colores dentro de las barras, podemos elegir la posición de las columnas como también hemos podido hacer con el histograma. Tanto `geom_bar()` como `geom_col()` tienen por defecto el argumento `position = "stack"`, que apila las barras una encima de la otra, pero podremos escoger tres maneras más de visualizar la posición de las columnas.

Alternativas para visualizar las columnas

Con `fill` nos llena cada columna hasta arriba de todo y nos enseña las proporciones de cada una. La posición `dodge` ubica las barras una junto a la otra (va bien cuando tenemos las variables *x* o *fill* con pocas categorías, por ejemplo, una variable binaria), mientras que `identity` nos muestra la estatura real de cada barra sin apilarlas. Esta última opción se visualiza mejor con transparencias (atributo `alpha`).

2.4. Diagrama de cajas

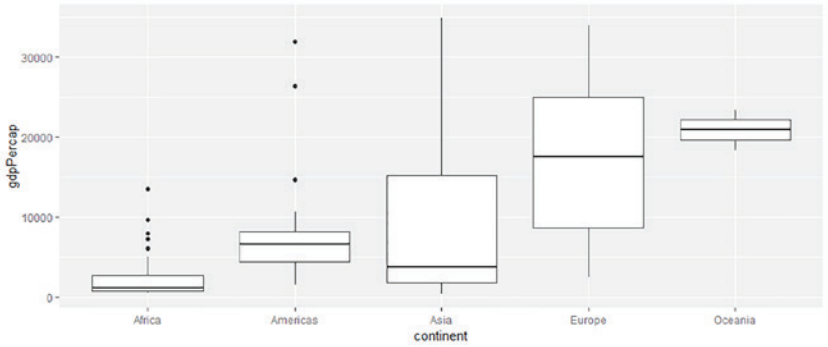
El diagrama de cajas nos ayuda a observar y a comparar cómo una variable numérica está distribuida según los valores de una variable categórica. A diferencia del diagrama de barras, que solo nos permite visualizar un solo valor por medio de la altura de

cada barra, la utilidad del diagrama de cajas radica principalmente en que nos permite visualizar algunos estadísticos descriptivos, como la mediana de cada distribución o los casos extremos. Con R, representaremos esta figura geométrica con la función `geom_boxplot()`. En el siguiente código, hemos pedido un diagrama de cajas para ver cómo estaban distribuidos en 1992 los valores de la variable *gdpPercap* en varios continentes.

```
gapminder %>% filter(year == 1992) %>%
    ggplot(aes(x = continent, y = gdpPercap)) + geom_boxplot()
```

Este código imprime un gráfico que da varios estadísticos descriptivos sobre cómo está distribuida la variable numérica según los valores de la variable categórica que hemos representado en el eje de las *x*. Es por eso por lo que el diagrama de cajas es un diagrama bastante sofisticado, con muchos elementos, y poder interpretarlo adecuadamente requiere práctica. En la figura 12, representamos el diagrama de cajas que genera el código anterior.

Figura 12. Diagrama de cajas



Fuente: elaboración propia.

Observemos el diagrama paso a paso: en primer lugar, la línea más gruesa que hay dentro de las cajas es la mediana de la distribución (el número que separa la distribución ordenada en dos partes iguales). Vemos que el único continente con una mediana superior a los 20.000 dólares es Oceanía. Eso significa que si ordenamos por PIB per cápita todos los países de Oceanía, de mayor a menor, el que está en mitad de la distribución supera ligeramente los 20.000 dólares. Europa se sitúa aproximadamente en los 17.000, mientras que el resto de continentes no superan los 10.000 dólares de mediana. Las partes superior e inferior de cada caja representan el 75 % y el 25 % de la distribución. Es decir, si tuviéramos cien países en cada continente, la parte superior de la caja nos indicaría el 25.º país más rico y la parte inferior el 75.º. Las cajas de África, América y Oceanía son muy pequeñas, lo cual significa que hay poca diferencia de riqueza entre el país que hace el 25 % y el que hace el 75 % de la distribución ordenada: en África casi todos son muy pobres, en América casi todos son más bien pobres, mientras que en Oceanía todos son bastante ricos. En cambio, las cajas de Asia y Europa son mucho más largas, lo que significa que en Asia y en Europa la distribución es más dispersa y hay tanto países ricos como países pobres.

Fuera de las cajas, encontramos dos figuras geométricas: líneas y puntos. Las líneas que salen de las cajas hacia arriba y hacia abajo nos indican el intervalo donde se encuentran los países por debajo del 25 % y por encima del 75 %. Cuando hay casos extremos, que son en este caso países muy alejados del resto de la distribución, se representan con puntos.

Fijaos en que en este diagrama de cajas hemos representado exactamente los mismos datos que en el histograma de la figura 6 y parecidos a los datos de la figura 9. Todos tienen sus

ventajas e inconvenientes, y para seleccionar el diagrama más adecuado tenemos que pensar cuál es nuestro propósito y cómo se visualizarán mejor los datos. El histograma y el diagrama de densidad nos permiten dar un vistazo general a la forma que tiene la distribución de una variable. Si la distribución es bimodal —es decir, que tenemos dos intervalos con concentraciones altas de valores—, estos gráficos nos permitirán distinguir con claridad estas dos puntas. En cambio, esta observación no la podremos hacer con un diagrama de barras ni con el diagrama de cajas.

La ventaja de los diagramas de cajas y de barras es que podemos ver la distribución en orientación horizontal o vertical añadiendo la capa de coordenadas `coord_flip()`. El diagrama de cajas permite hacer más comparaciones, puesto que describe varias propiedades de los datos, como los valores que se encuentran en el 25, el 50 y el 75 % de la distribución, la amplitud de la distribución y también los casos extremos.

2.5. Diagrama de dispersión

El diagrama de dispersión es una figura de dos dimensiones que representa con puntos la relación entre dos variables numéricas. Con R, representaremos esta figura geométrica con la función `geom_point()`. Más allá de lo que ya hayamos podido ver hasta ahora sobre el diagrama de dispersión, en este apartado destacaremos cómo podemos solucionar gráficamente uno de los problemas más habituales de este tipo de gráfico: la sobrerrepresentación. Este fenómeno es habitual en variables ordinales y en variables numéricas discretas, cuando los puntos representan una cantidad de valores limitada y se mueven en un rango más

bien pequeño. Esto hace que los puntos se puedan solapar entre ellos. La sobrerrepresentación también es habitual en el caso de variables nominales cuando tenemos una gran cantidad de puntos para representar. Visualmente, los puntos se solapan y no podemos distinguirlos bien.

Por poner un ejemplo, pensemos en las variables *año* y *esperanza de vida*. Son dos variables numéricas que tendrían que poder representarse con un diagrama de dispersión. Sin embargo, observad qué pasa en la figura 13 cuando intentamos representar estas dos variables del marco de datos `gapminder`. Es evidente que tenemos un problema importante de sobrerrepresentación. La variable *año* admite pocos valores y está tipificada como vector entero, mientras que la variable *esperanza de vida* puede tener poca variación entre sus valores. Con todo, los puntos se solapan.

Figura 13. Diagrama de dispersión con sobrerrepresentación



Fuente: elaboración propia.

Si nos fijamos en los colores que representan los continentes, sospechamos que la esperanza de vida de los países africanos es menor y la de los europeos, mayor, pero nos gustaría ver los datos de manera más nítida para llegar a conclusiones más concretas. Una buena solución es sustituir `geom_point()`

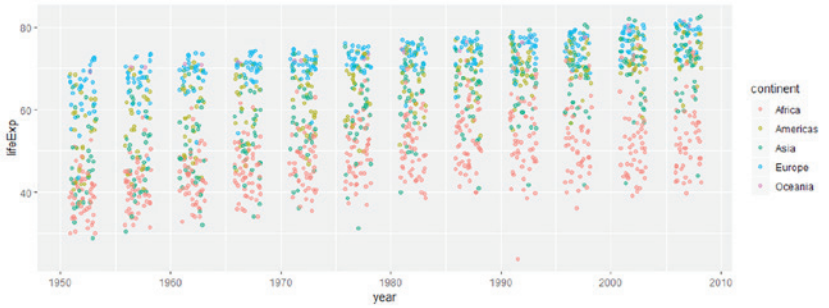
por `geom_jitter()`,[5] como vemos en el código siguiente, que añadirá un «ruido» o movimiento aleatorio a la posición de los puntos.

```
gapminder %>%
    ggplot(aes(x = year, y = lifeExp, col = continent)) +
       geom_jitter(width = 1.2, height = 0.1, alpha = 0.5)
```

El *jittering* separa los puntos entre sí mediante la introducción de un movimiento aleatorio horizontal y vertical que nos permite observar mejor el diagrama de dispersión. Si no añadimos nada dentro el paréntesis de `geom_jitter()` tendremos la misma aleatoriedad en vertical que en horizontal. Sin embargo, con el argumento `width` hemos querido especificar que el movimiento aleatorio en horizontal sea elevado, pero con el argumento `height` hemos ordenado que el movimiento aleatorio en vertical sea mínimo. En la figura 14 vemos el resultado:

5. Esta función hace lo mismo que `geom_point(position = "jitter")`. Por defecto, la posición de `geom_point()`es `"identity"`.

Figura 14. Diagrama de dispersión con *jittering*



Fuente: elaboración propia.

Otro método que hemos usado para reducir la sobrerrepresentación es el argumento `alpha`, que introduce transparencia en los puntos.

2.6. Retoques finales

Hay miles de retoques que se pueden hacer en un gráfico mediante ggplot2 para mejorar la visualización. Aquí os explicaremos dos retoques:

- poner títulos al gráfico y a los ejes, y
- eliminar casos extremos.

Para cambiar los títulos lo podemos hacer de tres maneras:

1) Especificarlo en las escalas de *x* y de *y*, como vemos en la primera línea del código siguiente. En el ejemplo, si trabajamos

con una variable numérica en el eje de las *x*, especificaremos el argumento `name` en la función `scale_x_continuous()`.[6]

2) Si no necesitamos modificar ninguna escala, nos será más fácil y rápido introducir argumentos dentro de la función `labs()` indicando el título general con `title`, el eje horizontal con *x* y el eje vertical con *y*. Si tenemos una leyenda que muestra los valores de un estético adicional, solo habrá que introducir el nombre del estético (*size*, *color*, *fill*...) seguido del título.

3) Los ejes horizontal y vertical, como también el título general del gráfico, se pueden introducir por medio de funciones separadas: `ggtitle()` permite introducir el título; `xlab()`, el título del eje horizontal, e `ylab()`, el título del eje vertical.[7]

```
scale_x_continuous(name = "Título eje x")

labs(title = "Título", x = "Título eje x", y = "Título eje y")

ggtitle("Título") + xlab("Título eje x")+ ylab("Título eje y")
```

El segundo retoque final para generar una mejor visualización del gráfico consiste en eliminar los casos extremos (*outliers*). Estos casos deforman los datos e impiden observar con claridad el centro de la distribución. Por ejemplo, si queremos mirar la relación entre el PIB per cápita y la esperanza de vida en 1952, hay un caso extremo que nos deforma completamente el gráfico.

```
gapminder %>%
```

6. Siempre pondremos *scale*, el nombre del estético y si la variable es *continuous* o *discrete*. Esto también implica estéticos como el color. Por ejemplo, en el caso de un estético categórico de color introduciremos `scale_color_discrete()`.

7. Si pedís ayuda de la función `theme()`, obtendréis mucha información sobre todo lo que podemos llegar a hacer con los títulos. También podéis consultar más información en este enlace: http://www.sthda.com/en- glish/wiki/ggplot2-title-main-axis-and-legend-titles.

```
    filter(year == 1952) %>%
    ggplot(aes(x = gdpPercap, y = lifeExp)) +
        geom_point()
```

Vemos que solo tenemos un único caso superior a los 20.000 dólares per cápita y que se aleja mucho del resto de la distribución, puesto que supera un PIB per cápita de 100.000 dólares. Para eliminar este caso extremo, podemos reducir los límites del eje de las *x* simplemente introduciendo la función `xlim(c(0, 20000))`. Otra opción es introducir un nuevo argumento dentro de la función `filter()` para indicar que filtre los casos donde `gdpPercap < 25000`. Estas dos opciones eliminarán el caso extremo y reproducirán el resto de datos.

Sin embargo, quizás queremos conservar este caso extremo y generar un nuevo marco de datos en el cual lo tengamos señalado. En este caso, podemos utilizar una combinación de las funciones `filter()` y `mutate()`, como vemos a continuación. Con la primera fórmula creamos el nuevo marco de datos `gap52` filtrado por `1952` que contiene la variable lógica *extrem*, que es `TRUE` para los casos con PIB per cápita superior a los 25.000 dólares y `FALSE` para el resto. A continuación, hacemos dos operaciones:

1) En la primera, filtramos por el caso extremo, que nos devolverá todos los valores que sean `TRUE` en la columna *extrem*. Aquí podemos comprobar que Kuwait es el caso extremo de nuestros datos.

2) En la segunda operación, filtramos excluyendo el caso extremo (con el símbolo `!`) y pedimos un ggplot2 de los datos filtrados.

```
gap52 <- gapminder %>% filter(year == 1952) %>%
   mutate(extrem = gdpPercap > 25000)
> gap52 %>% filter(extrem)
# A tibble: 1 x 6
    country   continent   year    lifeExp   pop      gdpPercap
1   Kuwait    Asia        1952    55.6      160000   108382
gap52 %>% filter(!extrem) %>%
   ggplot(aes(x = gdpPercap, y = lifeExp)) + geom_point()
```

Hay otras maneras de eliminar o filtrar casos extremos. La mayoría requerirán la función `filter()` o bien un poco de imaginación con las herramientas de los paquetes de R que ya conocemos.

3. Medidas de análisis univariante

Hasta ahora, hemos visto varias maneras de representar por medio de visualizaciones gráficas la información que contiene una variable. En este apartado buscaremos, en lugar de una visualización, un número o pocos que nos puedan sintetizar la misma información. En otras palabras, lo que queremos es cuantificar la manera en que están distribuidos los valores en la variable. El ejercicio de encontrar medidas para sintetizar distribuciones por medio de cifras concretas nos permitirá obtener información comparable, de forma que podremos contrastar diferentes variables entre ellas. Sin embargo, la manera en la que sintetizaremos la información variará según el tipo de variable. En el caso de las variables categóricas, no podemos hacer mucho

más que observar las frecuencias de cada valor. En el caso de las variables numéricas, podemos resumir la información mediante las diversas medidas que indiquen la centralidad y la dispersión de la distribución, y la localización de determinados valores. Buena parte de este trabajo la podemos hacer directamente con la función `summary()`, que devolverá las frecuencias de una variable categórica y algunas medidas de centralidad y localización de una variable numérica.

3.1. Tabla de frecuencias

La tabla de frecuencias es una de las modalidades más típicas para representar variables categóricas, como es el caso de la variable *continent* del marco de datos `gapminder`. Normalmente, en la tabla mostramos el número de veces que se repite cada valor categórico en la variable y su porcentaje sobre el total. Las frecuencias las podemos obtener con las funciones `table()` o `summary()`, y el porcentaje de frecuencias, con la función `prop.table()`.

```
table(gapminder$continent)
prop.table(table(gapminder$continent))
```

Si trabajamos con variables categóricas nominales, tendremos bastante con las dos funciones anteriores para visualizar los datos, pero, si se trata de variables ordinales, también nos puede interesar ver el acumulado para cada categoría. Es decir, querremos observar cómo se acumulan los valores y los porcentajes si los añadimos a los valores o porcentajes de las filas anteriores. En el código siguiente vemos cómo crear una tabla con frecuencias

y porcentajes acumulados a partir de la variable *income* del marco de datos `wb`.[8]

```
> data.frame(categories = rev(unique(wb$income)),
             freq = rev(summary(wb$income))) %>%
mutate(freq_cum = cumsum(freq),
       percent = round(prop.table(freq), 2),
       percent_cum = cumsum(percent))
     categories      freq      freq_cum   percent    percent_cum
1    low             3         3          0.20       0.20
2    lower-middle    8         11         0.53       0.73
3    upper-middle    3         14         0.20       0.93
4    high            1         15         0.07       1.00
```

La segunda columna de la tabla que hemos generado muestra las frecuencias que se repiten para cada categoría y la tercera columna muestra el acumulado de frecuencias, de forma que, por ejemplo, en la fila «lower-middle» vemos el acumulado de «lower-middle» y «low», en la fila «upper-middle» vemos el acumulado de «lower-middle», «low» y «upper-middle», y así sucesivamente. En las últimas dos columnas, hacemos el mismo procedimiento: primero, enseñamos el porcentaje de frecuencias sobre el total en cada categoría y, a continuación, el porcentaje acumulado.

Si queremos hacer una tabla de frecuencias con otra variable, solo habrá que sustituir las dos veces que aparece `wb$income` por la variable categórica que queramos reproducir.

8. Podéis encontrar el código del marco de datos `wb` en el anexo.

3.2. Medidas de centralidad

Para sintetizar una distribución en un solo valor, una manera es buscar dónde se encuentra ubicado el centro de la distribución. Fijaos en el siguiente vector: el objeto `ev` representa una distribución que nos indica la esperanza de vida, en números absolutos, recogida en once países de una región del mundo. Si queremos transformar estos datos en un solo dato numérico que nos resuma dónde se encuentra el centro de esta distribución, ¿cómo lo haríamos? ¿Qué número creéis que puede resumir mejor esta distribución?

```
ev <- c(65, 67, 68, 73, 74, 77, 80, 80, 80, 82, 83)
```

La respuesta es que dependerá de la pregunta concreta que formulemos, puesto que no hay una sola manera de sintetizar esta distribución para encontrar el centro. Principalmente, hay tres maneras de resumir estos datos:

1) Buscar el valor que se repite más veces. A esto se le llama *moda* y, en nuestro caso, sería el número 80, que se repite tres veces.

2) Buscar el valor que se encuentra en el centro de la distribución ordenada. A esto se le llama *mediana*, que en el vector `ev` es el número 77, puesto que es el valor que queda justo en medio de la distribución si lo ordenamos de mayor a menor.

3) Con la *media*, que se obtiene sumando todos los valores de la distribución y dividiéndolos por el número de casos.

3.2.1. Media

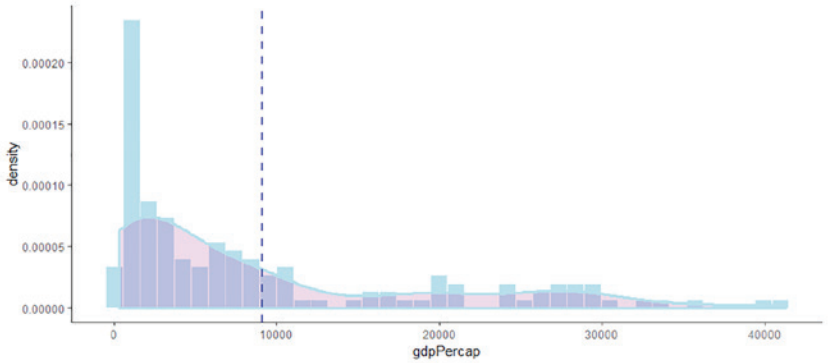
La media es una de las medidas de tendencia central más utilizadas para resumir distribuciones. Como vemos a continuación, para obtener la media tenemos que sumar todos los valores de un vector y dividir el resultado entre el número total de valores, que en este caso son once.

```
media <- sum(c(65, 67, 68, 73, 74, 77, 80, 80, 80, 82, 83)) / 11
```

Lo que hace la media es decirnos cuál sería el valor que obtendríamos si niveláramos el peso de cada valor de la distribución entre sus casos. Una manera muy ilustrativa para entender qué representa la media es con la distribución de la riqueza de un país. En cualquier país, la riqueza acostumbra a estar distribuida de manera desigual. Hay individuos que tienen más dinero y hay otros que tienen menos. Lo que nos dice la media es cuánto dinero le tocaría a cada habitante si cogiéramos el dinero de toda la población y lo repartiéramos a partes iguales. En todas las distribuciones, la media hace exactamente esto: suma todos los valores y los divide entre el número de frecuencias.

Fijémonos en la figura 15, donde hemos superpuesto un diagrama de densidad y un histograma que ilustran cómo estaba distribuido el PIB per cápita en los diferentes países del mundo en 1997. Con una línea azul hemos indicado dónde se encuentra la media.

Figura 15. Media del PIB per cápita mundial por países



Fuente: elaboración propia.

Si cogemos el dinero de los países más ricos y lo repartimos entre todos los países de forma que el dinero quede distribuido equitativamente, veremos que todos los países tendrían 9.090 dólares por habitante. Podemos pensar en la media como el punto que equilibra los datos. Este valor divide el diagrama de densidad en dos áreas aproximadamente iguales, de forma que el área de densidad de la izquierda es igual al área de densidad de la derecha. La media se calcula con la función `mean()`.[9] En el siguiente código hemos creado el objeto `gap97_mean` donde hemos especificado la media con una línea vertical `geom_vline()`. Imprimid el objeto después de crearlo.

```
gap97_mean <- gapminder %>%
    filter(year == 1997) %>%
    ggplot(aes(x = gdpPercap)) +
```

9. Alternativamente, también se puede calcular con `sum()` / `length()`, donde primero sumamos los valores de un vector y después dividimos el resultado por el recuento de observaciones.

```
        geom_histogram(aes(y=..density..), colour= "white",
                       fill= "lightblue", bins = 40) +
        geom_density(col = "lightblue", size = 1.2,
                     fill = "purple", alpha = 0.2) +
        geom_vline(aes(xintercept = mean(gdpPercap)),
                   colour = "blue",
                   linetype = "dashed", size = 1) +
        theme_classic()
gap97_mean
```

Las dos maneras más habituales de calcular la media es, o bien dentro de la función `summarize()`, o bien mediante un vector, como se indica en el código siguiente. En primer lugar, hemos creado un vector donde filtramos las observaciones por el año 1992 y, a continuación, hemos pedido la media del vector.

```
gdp_cap1992 <- filter(gapminder, year == 1992)
mean(gdp_cap1992$gdpPercap, trim = 0, na.rm = FALSE)
```

En este código tenemos que especificar dos argumentos adicionales cuya función por defecto es la media. Con el argumento `trim` podemos eliminar casos extremos de la distribución. Tendremos que especificar un valor de 0 a 0,5, que indicará el porcentaje de observaciones que eliminará de cada extremo de la distribución. El argumento `na.rm` es sumamente importante, puesto que elimina los NA de la distribución. El marco de datos `gapminder` no tiene datos perdidos, pero será frecuente encontrárnoslos en la mayoría de marcos de datos. Por eso, tendremos que cambiar el argumento a TRUE para que nos muestre la media.

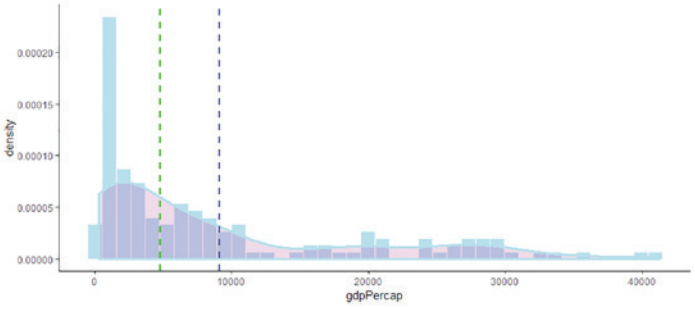
3.2.2. Mediana

La mediana nos indica qué valor tiene la observación que se encuentra justo en medio de la distribución. Dicho de otro modo: si tenemos una distribución con 144 países, la mediana nos ordena los valores de mayor a menor, y nos indica el valor del país que hace 72, que es el que se encuentra en medio de la distribución (lo obtenemos dividiendo 144 entre dos). Como vemos en el código siguiente, hemos utilizado el objeto `gap97_mean`, ya creado, que contiene toda la información sobre la geometría anterior, para añadirle una línea nueva con información sobre la mediana.

```
gap97_mean +
   geom_vline(aes(xintercept = median(gdpPercap)),
              color = "green", linetype = "dashed", size = 1)
```

En la figura 16 hemos ilustrado el resultado del código donde vemos una nueva línea vertical, de color verde, que representa la mediana.

Figura 16. Mediana y media del PIB per cápita mundial por países



Fuente: elaboración propia.

Es interesante ver cómo los dos centros divergen claramente. Si calculamos la mediana, veremos que se encuentra en los 4.782 dólares por habitante, mientras que la media se encontraba en los 9.090 dólares. Como la gran mayoría de observaciones ocurren en los valores más bajos del histograma, es lógico que la mediana se sitúe más sesgada a la izquierda de la distribución.

Otras medidas de localización

La mediana es una medida tanto de centralidad como de localización, puesto que nos indica el valor por debajo del cual tenemos ubicado el 50 % de observaciones de una distribución ordenada. Para referirnos a otras localizaciones de la distribución diferentes a la mediana, utilizaremos términos como:

- los percentiles (si contamos por unidades del 1 al 100),
- los deciles (si contamos de 10 en 10),
- los quintiles (de 20 en 20), y
- los cuartos (de 25 en 25).

Localizar el valor es fácil con la función `quantile()`, donde solo tenemos que indicar el nombre del objeto y la posición del valor que queremos localizar en la escala de 0 a 1. Por ejemplo, si queremos encontrar el percentil 40 (que también sería el cuarto decil o el segundo quintil) marcaremos `quantile(vector, 0.4)`. Podemos indicar tantas localizaciones como queramos si utilizamos el concatenado. Si queremos saber todos los quintiles, introduciremos `quantile(vector, c(0, 0.2, 0.4, 0.6, 0.8, 1)`. El cero nos devolverá el valor más bajo de la distribución y el 1, el más alto.

3.2.3. Moda

La observación más repetida en una distribución es la moda. Pueden ser ejemplos de moda la religión o el partido político dominante en un país. La moda es más fácil de encontrar en una variable categórica, puesto que será sencillamente la categoría con más frecuencias. Si imprimimos el sumario o pedimos una tabla de la variable continente de `gapminder`, encontraremos fácilmente la moda. Vemos claramente que la moda es África, la distribución con más frecuencias: 624.

```
> summary(gapminder$continent)
> table(gapminder$continent)
     Africa      Americas    Asia        Europe      Oceania
     624         300         396         360         24
```

Encontrar la moda en una variable numérica, en cambio, no es un ejercicio tan evidente. Tenemos que pensar que estas variables pueden adoptar un número infinito de valores y, por lo tanto, es poco probable encontrar algún número repetido y, si se da el caso, no nos aportará información más allá de la casualidad de que se repitan. La moda solo puede ser fácil de encontrar en el supuesto de que la variable en cuestión sea discreta, como la edad de una población. En este caso, si tenemos un número suficientemente grande de observaciones en relación con los valores que puede adoptar la variable, podremos obtener la moda sin problemas.

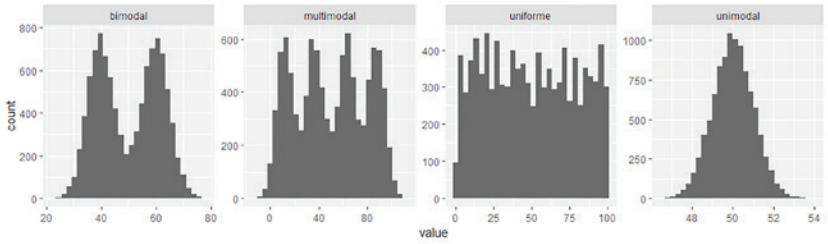
La discrecionalidad de la moda

Este procedimiento tiene un punto de discrecionalidad, puesto que podemos cambiar los intervalos del histograma o la sensibilidad del

diagrama de densidad. Esto puede hacer que la moda que establezcamos visualmente varíe en función de los parámetros que establecemos. Haced la prueba: coged el código que hemos utilizado para crear la figura 15 y cambiad los intervalos (*binwidth*) a 20. Veréis que la moda, por ejemplo, está más repartida entre dos intervalos.

En variables numéricas continuas, sin embargo, lo tenemos más complicado. Para obtener información sobre nuestro análisis, en lugar de encontrar un valor concreto será más útil saber dónde están concentrados los valores de la variable. Este ejercicio lo podemos hacer fácilmente con un histograma o un diagrama de densidad, en el cual podemos visualizar cuál es el punto más alto de la distribución. En la figura 17 podemos observar los diversos tipos de distribuciones que pueden dar lugar a diferentes tipos de modas.

Figura 17. Tipos de moda



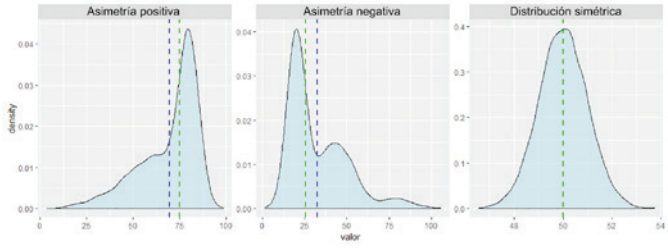
Fuente: elaboración propia.

Hablaremos de *distribución unimodal* cuando solo haya una punta, de *bimodal* cuando haya dos, de *multimodal* cuando haya más de dos puntas y de *uniforme* cuando no se aprecie ninguna punta concreta.

3.2.4. Tipos de transformaciones

A medida que nos vamos familiarizando con diferentes tipos de distribuciones, veremos que podemos intuir la forma que tienen sin visualizarlas. Tendremos bastante con conocer la moda, la media y la mediana. Si la media es inferior a la mediana, seguramente obtendremos un gráfico parecido al diagrama de densidad de la izquierda de la figura 18 y diremos que es una *distribución asimétrica positiva*.[10] Como vemos, la cola del gráfico está a la izquierda y los valores están sesgados a la derecha de la distribución. Si la media, en cambio, es superior a la mediana, posiblemente nos encontraremos con el gráfico del medio y diremos que es una *distribución asimétrica negativa*. La cola del gráfico, en este caso, se encuentra a la derecha de la distribución y los valores están sesgados a la izquierda de la distribución. En último lugar, si la moda, la media y la mediana coinciden más o menos, querrá decir que tenemos una distribución simétrica. En este caso, la cola a un lado y otro es parecida y no hay sesgo.

Figura 18. Distribuciones simétricas y asimétricas



Fuente: elaboración propia.

10. Con el diagrama de densidad se pierde información en relación con el histograma, puesto que no tenemos la información separada por intervalos. Aun así, es útil cuando queremos añadir información encima del gráfico, puesto que visualmente no quedará sobrecargado.

Las distribuciones de los ingresos en un país acostumbran a tener una asimetría negativa. Esto quiere decir que la mayoría de casos están situados a la izquierda de la distribución y la mediana es inferior a la media. En otras palabras, el individuo situado en mitad de la distribución (la mediana) no tiene tanto dinero como el que le tocaría a cada individuo si repartiéramos todo el dinero a partes iguales entre el conjunto de población (la media).

Sensibilidad de la media

La media es sensible a distribuciones sesgadas y valores extremos, de forma que, en estos casos, puede ser un método más adecuado para medir la centralidad. Una manera de evitar que los valores extremos influyan excesivamente en el cálculo de la media es con el argumento `trim`, que elimina la proporción que indicamos de cada extremo. Por ejemplo, `mean(variable1, trim = 0.01)` eliminará el último 1 % de cada extremo.

A menudo, para visualizar mejor estas distribuciones y hacer pruebas de significación estadística, se acostumbra a construir una versión transformada de la variable. En los casos de distribuciones con asimetría negativa, se aplica la escala logarítmica con las funciones `log()`, `log2()` o `log10()`, mientras que en distribuciones con asimetría positiva se puede aplicar el exponencial con la función `exp()`.

3.3. Medidas de dispersión

Las medidas de dispersión nos indican la separación o dispersión de los valores en una distribución numérica. En general, si

los valores están muy concentrados alrededor del centro indicarán un valor bajo, mientras que si están muy separados indicarán un valor alto. Hay tres maneras principales de mirar la dispersión:

- el rango,
- el rango intercuartílico, y
- la desviación típica.

3.3.1. Rango

El rango muestra la diferencia entre el valor máximo y el valor mínimo de una distribución. Es una medida de dispersión que no se utiliza mucho porque es muy sensible a los valores extremos, de forma que un solo valor puede tener un gran efecto en el valor que muestre el rango. En el código siguiente indicamos dos maneras de calcular el rango:

1) Con la función `range()`, para encontrar los valores máximo y mínimo de una distribución y aplicar `diff()` para ver la diferencia entre estos dos valores.

2) Restándole el valor mínimo `min()` de la distribución al valor máximo `max()`.

```
> diff(range(gapminder$gdpPercap))
> max(gapminder$gdpPercap)- min(gapminder$gdpPercap)
113282
```

Los valores mínimo y máximo de la distribución también se pueden obtener mediante la función `summary()`.

3.3.2. Rango intercuartílico

A diferencia del rango, el rango intercuartílico (*InterQuartile Range*, *IQR*) es menos sensible a los valores extremos, puesto que mide la diferencia entre el primer cuantil de la distribución y el tercero. En otras palabras, mide la diferencia entre los valores que ocupan los lugares 25 y 75 % del rango en una distribución ordenada. Esta medida de dispersión se puede obtener con la función `IQR()`. Alternativamente, como vemos en el siguiente código, también podemos calcularlo restando el cuantil 25 al cuantil 75, usando la función `quantile()` e indicando, en una escala de 0 a 1, la posición correspondiente del primer cuantil y el tercero.

El IQR es la medida de la caja

Fijaos en la figura 12. Lo que hace el IQR es medir de forma cuantitativa el tamaño de las cajas de un diagrama de cajas.

```
> IQR(gapminder$gdpPercap)
> quantile(gapminder$gdpPercap, 0.75) -
quantile(gapminder$gdpPercap, 0.25)
8123.402
```

El primer cuantil de la distribución y el tercero también se pueden obtener mediante la función `summary()`. Los encontraremos con el nombre de *1st Qu.* y *3rd Qu.*

3.3.3. Desviación típica

La desviación típica es una medida de dispersión bastante más compleja en comparación con las que hemos visto anteriormente, y es por eso por lo que le tendremos que dedicar más espacio y atención. Esta medida indica la dispersión de los valores respecto de la media. Para entender su significado numérico, tendremos que tener presentes tres cuestiones importantes:

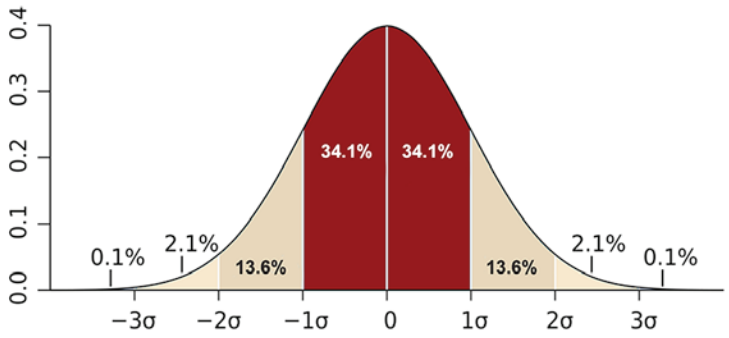
1) Un valor alto de desviación típica querrá decir que, de media, los valores están situados lejos de la media, mientras que un valor bajo de desviación típica significará que los valores están concentrados cerca de la media.

2) La desviación típica está medida con las mismas unidades que los valores originales. Esto quiere decir que, si medimos la esperanza de vida en años, donde los valores oscilan entre 50 y 80 años, tendremos desviaciones típicas más bajas que si medimos esperanzas de vida en meses, donde tendremos valores más elevados. Si medimos el PIB per cápita, obtendremos seguramente desviaciones típicas más altas, puesto que los valores de esta variable se mueven entre varios miles. Por lo tanto, la desviación típica nos será útil para comparar distribuciones, siempre que estén medidas en las mismas unidades.

3) La desviación típica nos indica con bastante precisión qué porcentaje de valores está situado alrededor de la media. En una distribución simétrica normal, como en la figura 19, una distribución típica nos indicará que el 68,2% de los valores está situado a la izquierda y a la derecha de la media. Dos desviaciones típicas nos indicarán que un 95,4% de los valores está situado ala izquierda y a la derecha de la media; y tres desviaciones típi-

cas nos indicarán que el 99,6 % de los valores está situado a la izquierda y a la derecha de la media.

Figura 19. Distribución normal con desviaciones típicas



Fuente: M. W. Toews, CC BY 2.5
(https://en.wikipedia.org/wiki/Standard_deviation#/media/ File:Standard_deviation_diagram.svg).

Para ver el concepto de la desviación típica de una manera más práctica, en el código siguiente hemos creado los vectores `dev_ex1` y `dev_ex2`, formados por nueve valores cada uno. Con la función `mean()` veremos que la media de los dos vectores es la misma: 5,11. Esto significa que si vamos sacando peso a los valores numéricos más altos y damos este peso a los valores numéricos más bajos, de forma que al final nos quede este peso repartido de manera equitativa entre todos los valores, la cifra que obtendremos será 5,11.

```
dev_ex1 <- c(4, 6, 5, 8, 5, 3, 5, 6, 4)

dev_ex2 <- c(3, 9, 4, 8, 6, 1, 5, 8, 2)
```

Sabemos que tanto en un vector como en el otro la media es la misma, pero ¿hasta qué punto es adecuado el número 5,11 para representar los valores de cada vector? Será más adecuado

si todos los valores son cercanos a 5. En cambio, 5,11 será una medida menos fiel de representar los valores del vector si estos son valores alejados de 5, como 1, 2 o 9.

La desviación típica nos ayuda a hacernos una idea de si los valores de un vector están cerca o lejos de la media. Para calcularla, hemos reproducido los pasos para el vector `dev_ex1` en el código siguiente: en primer lugar, calculamos la diferencia de cada uno de los vectores respecto de la media. Para los valores que sean inferiores a la media, esta diferencia nos saldrá negativa, mientras que, para los valores superiores a la media, la diferencia será positiva. Para hacer que todos los números sean positivos, elevaremos todos los valores al cuadrado y seguidamente los sumaremos. A continuación dividimos esta suma por el número de observaciones menos uno para evitar que el valor obtenido sea más grande a medida que tengamos más observaciones. El valor que obtenemos hasta aquí se denomina *varianza*. La desviación típica se obtiene de la raíz cuadrada de la varianza.[11]

```
(diff_mediana <- dev_ex1 - mean(dev_ex1))
(diff_cuadrado <- (diff_mediana)^2)
(suma_diff_cuadrado <- sum(diff_cuadrado))
(varianza <- suma_diff_cuadrado/(length(dev_ex1) - 1))
(desviacion_tipica <- sqrt(varianza))
```

La desviación típica del vector `dev_ex1` es de 1,45, mientras que si cambiamos los códigos y buscamos la desviación típica del vector `dev_ex2` veremos que es de 2,84. Por suerte, para saber

11. Hemos puesto las fórmulas del cuadro siguiente entre paréntesis para poder ver impreso en la consola cada uno de los pasos para obtener la desviación típica.

la desviación típica no habrá que hacer estas operaciones complicadas, sino que la función `sd()` nos la calculará directamente.

Y ahora que tenemos la desviación típica de estos vectores, ¿cómo podemos interpretarlos? Principalmente, lo que podemos decir es que en el primer vector encontraremos aproximadamente un 68,2% de los valores entre 3,66 y 6,56, mientras que en el segundo vector encontraremos el 68,2% de los valores entre 2,27 y 7,95.[12] Así pues, si miramos las desviaciones típicas de uno y vector otro, podemos interpretar que los valores del segundo están más dispersos en relación con la mediana que en el primero. Si añadimos o restamos dos desviaciones típicas a la media, podremos saber también dónde estarán situados el 95,4% y el 99,6% de los valores.

La desviación típica es mucho más sensible a los valores extremos que el IQR, pero lo es menos que el rango.

Ejercicio: ejemplo de sensibilidad a los valores extremos

Para probar la sensibilidad de las medidas sintéticas de dispersión ante casos extremos, fijaos en este vector: `c(1,2,3,4,5,6,7,8,9,10,20)`. Tenemos un caso extremo, el número 20. Probad a calcular el rango `diff(range())`, el rango intercuartílico `IQR()` y la desviación típica `sd()`. Después, sustituid el número 20 por el 11 y volved a hacer los mismos cálculos. ¿Cuál es la medida de dispersión más sensible a los casos extremos? ¿Y cuál es la menos sensible?

12. Estos números los obtenemos restando una desviación típica a la mediana (5,11 – 1,45 = 3,66, y 5,11 – 2,84 = 2,27) y sumando una desviación típica a la mediana (5,11 + 1,45 = 6,56, y 5,11 + 2,84 = 7,95). La estimación asume que se trata de una distribución simétrica normal con un número elevado de casos.

Ejercicios de autoevaluación

Para un mejor el aprendizaje, intentad hacer mentalmente el máximo de ejercicios posible, sin utilizar R.

1) Cambiad la función para que muestre las proporciones en cada intervalo.

```
geom_histogram()
```

2) Tenemos una distribución con valores de 10 a 100. ¿Qué argumento indicaremos para tener un histograma formado por quince intervalos?

```
geom_histogram()
```

3) Visualizad el siguiente diagrama de densidad con una transparencia del 50 %.

```
geom_density()
```

4) Generad un diagrama de línea con todas las variables del siguiente marco de datos.

```
md$exports, md$year, md$country
```

5) Cambiad los estéticos de la geometría de forma que el color de la línea y el tipo de línea muestre la variable *country*.

```
geom_line()
```

6) Indicad la geometría correcta para visualizar el recuento de observaciones.

```
geom_bar(), geom_col()
```

7) Eliminad la leyenda del gráfico con un argumento dentro de la geometría.

```
geom_col()
```

8) Pedid un diagrama de cajas con las siguientes variables:

```
md$categorica, md$numerica
```

9) Introducid ruido horizontal en el siguiente diagrama de dispersión:

```
ggplot(md, aes(x, y)) + geom_point()
```

10) Sin utilizar las funciones de dplyr, cread el vector lógico `extrem` a partir de la siguiente variable para detectar casos superiores al valor 5.

```
militarydata$gasto_pib
```

11) Pedid una tabla de frecuencias con porcentajes de la siguiente variable:

```
tradedata$countries
```

12) Generad una tabla de frecuencias de clase vector para la siguiente variable:

```
tradedata$countries
```

13) ¿Cuál es la moda del siguiente vector?

```
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10)
```

14) ¿Cuál es la mediana del siguiente vector?

```
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10)
```

15) Simplificad las funciones siguientes para calcular la media de este objeto:

```
sum(distribution) / length(distribution)
```

16) Calculad el cuantil 66 del siguiente vector:

```
militarydata$gasto_pib
```

17) Pedid el valor mínimo del siguiente vector:

```
militarydata$gasto_pib
```

18) Preguntad a R si la IQR del siguiente vector es inferior al rango.

```
md$vector
```

19) ¿Cuál es el resultado obvio del ejercicio 18?

```
TRUE o FALSE
```

20) Calculad la desviación típica del siguiente vector:

```
militarydata$vector
```

Solucionario

1) `geom_histogram(position = "fill")`

2) `geom_histogram(bins=15)o geom_histogram(binwidth = 6)`

3) `geom_density(alpha = 0.5)`

4) `ggplot(md, aes(x = year, y = exports, col = country)) + geom_line()`

5) `geom_line(aes(col = country, lty = country)`

6) `geom_bar()`

7) `geom_col(show.legend = FALSE)`

8) `ggplot(md, aes(x = categorica, y = numerica)) + geom_boxplot()`

9) `ggplot(md, aes(x, y)) + geom_jitter(width = 1.2, height = 0)`

10) `extrem <- militarydata$gasto_pib > 5`

11) `prop.table(table(tradedata$countries))`

12) `summary(tradedata$countries )`

13) `10`

14) `6`

15) `mean(distribution)`

16) `quantile(militarydata$gasto_pib, 0.66)`

17) `min(militarydata$gasto_pib)`

18) `IQR(md$vector) <- diff(range(md$vector))`

19) `TRUE`

20) `sd(militarydata$vector)`

Glosario

`geom_bar()` Introduce la geometría de un diagrama de barras.

`geom_boxplot()` Introduce la geometría de un diagrama de cajas.

`geom_col()` Introduce la geometría de un diagrama de barras.

`geom_density()` Introduce la geometría de un diagrama de densidad.

`geom_histogram()` Introduce la geometría de un histograma.

`geom_hline()` Introduce la geometría de una línea horizontal.

`geom_jitter()` Introduce ruido en un diagrama de dispersión.

`geom_line()` Introduce la geometría de un diagrama de línea.

`geom_point()` Introduce la geometría de un diagrama de dispersión.

`geom_vline()` Introduce la geometría de una línea vertical.

`ggtitle()` Permite poner un título en el gráfico.

`IRQ()` Muestra el rango intercuartílico de una distribución.

`labs()` Permite poner etiquetas en el gráfico.

`max()` Muestra el valor más alto de una distribución numérica.

`mean()` Devuelve la suma de todos los valores de una distribución, dividida por el número de valores.

`median()` Devuelve el valor central de una distribución ordenada.

`min()` Muestra el valor más bajo de una distribución numérica.

`prop.table()` Permite obtener los porcentajes de una tabla de frecuencias.

`quantile()` Muestra el valor del cuantil que especificamos de una distribución.

`range()` Devuelve el valor mínimo y máximo de una distribución.

`sd()` Muestra la desviación típica de una distribución.

`table()` Muestra la tabla de frecuencias de una variable categórica.

`xlab()` Permite introducir el título de eje de las *x*.

`ylab()` Permite introducir el título de eje de las *y*.

Bibliografía

Babbie, E. R. (2013). *The practice of social research*. Wadsworth: Cengage Learning.

Brancati, D. (2018). *Social Scientific Research*. Londres: Sage Publications Ltd.

Halperin, S.; Heath, O. (2016). *Political Research: Methods and Practical Skills*. Oxford: Oxford University Press.

King, G.; Keohane, R. O.; Verba, S. (1994). *Designing Social Inquiry: Scientific Inference in Qualitative Research*. Princeton: Princeton University Press.

Anexo

Código del apartado 3.1

```
wb <- data.frame(country = c("Antigua and Barbuda", "Belice",
"Costa Rica", "Dominica", "Dominican Republic", "El Salvador",
"Guyana", "Guatemala", "Haiti", "Honduras", "Jamaica",
"Nicaragua", "Panama", "Surinam", "Trinidad and Tobago"),
income = factor(c("high", "upper-middle", "upper-middle",
"upper-middle", "upper-middle", "lower-middle", "upper-middle",
"upper-middle", "low", "lower-middle", "upper-middle", "lower-
middle", "high", "upper-middle", "high")), stringsAsFactors =
FALSE)
```

Código de la figura 17

```
moda <- data.frame(
unimodal = rnorm(10000, 50), #unimodal
bimodal = c(rnorm(5000, 60, 5), rnorm(5000, 40, 5)), #bimodal
multimodal = c(rnorm(2500, 12, 7), rnorm(2500, 37, 7),
rnorm(2500, 63, 7), rnorm(2500, 88, 7)),#multimodal uniforme =
rep(sample(1:100, 5000, replace = T), 2))
moda_ty <- gather(moda, tipo)
ggplot(moda_ty, aes(x = value)) + geom_histogram() +
facet_grid(. ~ tipo, scale = "free")
```

Código de la figura 18

```
simetria <- data.frame(simetrica = rnorm(10000, 50),
dre = c(rnorm(5500, 80, 5), rnorm(2500, 65, 9),
rnorm(1500, 50, 9), rnorm(500, 30, 9)),
esq = c(rnorm(5500, 20, 5), rnorm(2500, 40, 9),
rnorm(1500, 50, 9), rnorm(500, 80, 9)))
distr_ty <- gather(simetria, tipo, valor)
mm <- distr_ty %>%
group_by(tipo) %>%
summarize(grp.mean = mean(valor),
grp.median = median(valor))
ggplot(distr_ty, aes(x = valor)) +
geom_density(fill = "lightblue", alpha = 0.5) +
facet_wrap(. ~ tipo, scale = "free",
labeller = as_labeller(c(dre = "Asimetría positiva",
esq = "Asimetría negativa",
simetrica = "Distribución simétrica"))) +
geom_vline(data = mm, aes(xintercept=grp.mean),
color = "blue", linetype = "dashed",
size=1) +
geom_vline(data = mm, aes(xintercept=grp.median), color =
"green", linetype = "dashed", size=1)
```

Capítulo IV
Análisis bivariante

1. Introducción

El análisis bivariante se ocupa del análisis estadístico con dos variables. A veces las funciones del análisis bivariante se confunden con las del análisis univariante, puesto que a menudo este último también admite el análisis con dos variables. La gran distinción, sin embargo, es que el análisis univariante está orientado a hacer comparaciones entre subgrupos y describir las similitudes y las diferencias de esta variable entre subgrupos, mientras que el análisis bivariante busca principalmente explicar cómo dos variables se relacionan entre sí (Babbie, 2013). Por ejemplo, en análisis univariante podemos observar si los países que tienen acceso al mar son más ricos que los países que no lo tienen. En cambio, en análisis bivariante, buscaremos la asociación entre estas dos variables y diremos que tener acceso al mar tiene un efecto sobre la riqueza. Cuando hablemos, pues, de asociaciones o efectos entre variables, solo lo haremos mediante las técnicas del análisis bivariante, que nos permitirá sugerir desde un punto de vista estadístico si el comportamiento de una variable está parcialmente determinado por la otra (King y otros, 1994).

Establecer asociaciones entre dos variables es una práctica frecuente en el día a día. Acostumbramos a decir que la contaminación tiene un efecto sobre el calentamiento global, que los países democráticos suelen ser más ricos o que tener una ideología de derechas influye a la hora de votar un partido

determinado. En otras palabras, creemos que si conocemos los valores de la primera variable (niveles de contaminación, democracia o ideología) podremos saber algo sobre los valores de la segunda variable (calentamiento global, riqueza o voto a partido) porque creemos que están relacionadas. Llamaremos a la primera variable, que es la que pensamos que causa el efecto, *variable independiente* y la ilustraremos con el símbolo *x*. La segunda variable, que es la que creemos que está afectada por la variable independiente, la denominaremos *variable dependiente* y la ilustraremos con el símbolo *y*.

Otras denominaciones de las variables

Hay otras maneras de referirse a las variables independiente y dependiente. A la variable independiente se la denomina *variable explicativa*, mientras que la variable dependiente es llamada *variable explicada* o *de respuesta*.

Para visualizar y cuantificar estadísticamente una asociación bivariante, utilizaremos una técnica diferente según si las variables independiente y dependiente son numéricas o categóricas. En la tabla 13 resumimos la técnica utilizada según el tipo de variable y si ocupa la posición de independiente o dependiente. Si tanto la variable independiente como la dependiente son categóricas, utilizaremos una tabla de contingencia. Si, en cambio, tanto la variable independiente como la dependiente son numéricas, utilizaremos la regresión y la correlación. Si la variable independiente es categórica y la dependiente es continua, usaremos la diferencia de medias. Finalmente, si la variable independiente es continua y la variable dependiente es categórica, haremos una regresión logística.

Tabla 13. Técnicas de análisis bivariante

		Variable independiente	
		Categórica	Numérica
Variable dependiente	Categórica	Tablas de contingencia	(Regresión logística)
	Numérica	Diferencia de medias	Regresión y correlación

Fuente: elaboración propia.

En este capítulo no explicaremos la regresión logística, de forma que las páginas siguientes describirán las otras tres técnicas de análisis bivariante:

- la tabla de contingencia,
- la diferencia de medias, y
- la regresión y la correlación.

Cada apartado muestra primero cómo representar visualmente una relación bivariante según si las variables independiente y dependiente son numéricas o categóricas, y a continuación se explica la manera de cuantificar la relación.

Aunque las técnicas utilizadas cambien según los tipos de variable, todas las combinaciones tienen unos requisitos similares para indicar si dos variables tienen relación.

El primer requisito, que es el que trataremos con más profundidad en estas páginas, es que la asociación observada tenga apoyo estadístico. A veces podemos observar asociaciones poco consistentes entre variables. Por ejemplo, si echamos una moneda y sale dos veces cara, sería una inferencia débil decir que siempre saldrá cara. Desde un punto de vista estadístico, basado en las teorías de la probabilidad, para hacer inferencias a partir de

una muestra necesitamos un número bastante elevado de casos y/o que la variación de la variable dependiente cuando varía la independiente sea suficientemente grande. Por lo tanto, la relación observada tiene que ser lo bastante consistente como para que no sea muy probable pensar que realmente no hay relación en el fenómeno observado.

El segundo requisito se refiere a la temporalidad. Es decir, por lógica, la variable independiente se tiene que haber producido antes o más o menos al mismo tiempo que la variable dependiente.

Y, finalmente, para poder decir qué variable tiene un efecto sobre otra, no solamente hace falta que la estadística y la temporalidad nos den la razón, sino que la teoría también nos la tiene que dar. Este es el requisito más difícil de conseguir y por razones de espacio no trabajaremos de manera específica esta parte en este capítulo. Sin embargo, hay que tener claro que para sugerir que dos variables tienen relación hará falta que esta relación sea lógica y plausible y, por eso, es importante haber descartado las otras alternativas que, teóricamente, pueden explicar el fenómeno que queremos estudiar.

La necesidad de apoyo teórico en el análisis

La relación entre riqueza y democracia es un buen ejemplo para explicar la necesidad de teoría. Si los datos nos dicen que los países democráticos son más ricos que los países no democráticos, esto no quiere decir que haya un efecto entre ser una democracia y ser rico. Podría ser que el efecto fuera a la inversa: que ser rico te haga ser más democrático. O bien que haya otra variable que desconocemos que afecte a la riqueza y la democracia a la vez, pero que democracia y riqueza no

se afecten entre ellas. Encontraréis esta idea ampliada en Halperin y Heath (2016, págs. 369-370).

2. Relación entre variables categóricas

La tabla de contingencia es la manera de representar una relación bivariante cuando las variables independiente y dependiente son categóricas. Normalmente, representaremos los valores de la variable independiente en el eje horizontal y los valores de la dependiente en el eje vertical, de forma que cada celda de dentro de la tabla representa el recuento total de observaciones que caben en cada combinación entre niveles de las dos variables. Una vez tengamos en cada celda el número de frecuencias, calcularemos los porcentajes de cada frecuencia sobre el total de la columna.

Para ilustrar la tabla de contingencia, utilizaremos en esta sección el paquete unvotes (Voeten, 2017), que contiene información sobre las votaciones en la Asamblea General de las Naciones Unidas (AGNU), separadas en tres paquetes. Lo primero que haremos es instalar y cargar el paquete y pediremos la estructura de los tres marcos de datos.

```
str(un_votes)
str(un_roll_calls)
str(un_roll_call_issues)
```

En este apartado, utilizaremos la tabla de contingencia para responder a una pregunta sobre la información contenida en el paquete unvotes: como sabéis, Sudán del Sur es el último país que ha ingresado en la Organización de las Naciones Unidas (ONU)

después de conseguir la independencia en 2011. Partimos de la premisa de que Sudán del Sur se ha querido diferenciar respecto de Sudán en política internacional y queremos comprobar nuestra hipótesis mirando si estos dos países han votado de modo diferente en la AGNU a partir de la independencia de Sudán del Sur.

El paquete *unvotes*

El marco de datos `un_votes` tiene un registro de todas las votaciones en la AGNU según si cada estado votó a favor, en contra o se abstuvo. El marco de datos `un_roll_calls` tiene información más concreta sobre cada votación, como el número de resolución y una descripción del contenido de la votación. El marco de datos `un_roll_calls_issues` separa las votaciones según la temática.

2.1. Visualización con tabla

En la tabla de contingencia situaremos dos variables categóricas:

- La independiente es *country*, situada en las columnas, que puede adoptar los valores Sudán o Sudán del Sur.
- La dependiente es *vote*, situada en las filas, que puede tomar los valores Sí o No.

Sacar niveles de factores

Como la variable *vote* es un factor, cuando eliminamos todos los valores de una de las categorías (en este caso la categoría "abstain"), no nos elimina el nivel correspondiente. Para eliminarlo, tendremos que

incluir la función `droplevels()`. Mirad cómo se visualiza la tabla de contingencia si sacáis esta función del código.

Creemos que cada país tiene un patrón de voto diferente, de forma que, si conocemos los valores de la variable independiente *country,* podremos saber algo sobre los valores de la dependiente *vote*. Por eso, en el código siguiente hemos filtrado los datos por los dos países que queremos mirar y por el número de votación 5117 a partir de la cual Sudán del Sur participó en la AGNU. También hemos eliminado las abstenciones y hemos pedido solo las votaciones relacionadas con el desarrollo económico. Para visualizar la tabla de contingencia, solo hay que indicar los dos vectores categóricos como argumentos de la función `table()`.

```
sudan_ec_votes <- un_votes %>%
    inner_join(un_roll_call_issues) %>%
    filter(country == c("Sudan", "South Sudan"), rcid >= 5117,
           vote != "abstain", short_name == "ec") %>%
    select(country, vote) %>% droplevels()


> table(sudan_ec_votes$vote, sudan_ec_votes$country)
          South Sudan    Sudan
yes       11             16
no        1              3
```

Cada celda de la tabla que hemos generado representa una combinación del recuento total de observaciones entre categorías de las dos variables. Parece que Sudán ha votado más veces y más a favor que Sudán del Sur, puesto que ha votado de manera

favorable en 16 ocasiones y en contra tres veces. Sudán del Sur ha votado de manera favorable en once ocasiones y en contra en tres ocasiones.

Sumar filas o columnas

Si queremos ver la suma de las filas o las columnas de la tabla de contingencia, tendremos que introducir la función anterior dentro de la función `rowSums()` o `colSums()`. R nos devolverá un vector numérico con la suma de cada fila o columna.

Aunque desde un punto de vista descriptivo es bueno ver cuántas observaciones tenemos en cada combinación, el análisis de una tabla de contingencia lo haremos de manera más esmerada con proporciones. Si sabemos los porcentajes, nos será más fácil comparar los valores entre columnas y distinguir las probabilidades de que la variable dependiente tome un valor determinado si conocemos los valores de la independiente. Para ver las proporciones, utilizaremos la función `prop.table()` con el argumento `margin = 2`, que nos permitirá ver los porcentajes en columnas. También hemos multiplicado la tabla por 100 para ver el resultado en porcentajes y hemos utilizado `round()` para redondear los valores a un decimal.

Proporciones por columnas, filas o totales

La mejor manera de ver una tabla de contingencia es con las proporciones por columnas. Sin embargo, también podemos ver las proporciones por filas si introducimos el argumento `margin = 1`. Si no introducimos ningún argumento, veremos el porcentaje de cada celda sobre el total de celdas de la tabla.

```
> round(prop.table(table(sudan_ec_votes$vote,
    sudan_ec_votes$country), margin = 2) * 100, 2)
        South Sudan      Sudan
yes      91.67            84.21
no       8.33             15.79
```

Vista con proporciones, la tabla de contingencia nos da otro ángulo de los datos en comparación con la tabla de contingencia vista en frecuencias. Desde 2011, Sudán del Sur tiende a votar las resoluciones sobre desarrollo económico más favorablemente que Sudán. Por lo tanto, parece que de momento los resultados no contradicen nuestra hipótesis de partida.

2.2. Visualización gráfica

El diagrama de barras es la manera más habitual de visualizar gráficamente las tablas de contingencia. Si utilizamos el paquete ggplot2 usaremos la geometría `geom_bar()`. En primer lugar, visualizaremos el recuento de observaciones. En el código siguiente, intentad cambiar la posición por defecto de la geometría `position = "stack"` por `position = "dodge"`, que nos mostrará horizontalmente las diferentes categorías de cada columna.

```
sudan_ec_votes %>%
    ggplot(aes(x = country, fill = vote)) +
    geom_bar() + theme_minimal()
```

Si queremos visualizar la tabla de contingencia en proporciones, solo hará falta que cambiemos la posición de la geometría del código anterior. En este caso, situaremos `position= "fill"`.

Figura 20. Visualización gráfica de una tabla de contingencia



Fuente: elaboración propia.

En el código de esta tabla, también hemos disminuido la anchura de las barras con el argumento `width = 0.4` y hemos eliminado el nombre del eje horizontal con `xlab(NULL)`.

Diferencia de voto con las administraciones Carter y Reagan

¿Cómo cambió el voto de Estados Unidos en las Naciones Unidas entre las administraciones Carter y Reagan? En este código tenéis la respuesta.

```
un_roll_calls %>%
    inner_join(un_votes) %>%
    select(-c(country, descr, short)) %>%
    filter(country_code == "US", vote != "abstain",
```

```
        between(date, as.Date("1977-01-20"),
                as.Date("1989-01-20"))) %>%
  mutate(president = if_else(date < as.Date("1980-01-20"),
                             "Carter", "Reagan")) %>%
  ggplot(aes(x = president, fill = vote)) +
  geom_bar(position = "fill")
```

Fijaos en cómo hemos filtrado las votaciones para poder separar la muestra en dos periodos, el que gobernó Carter y el que gobernó Reagan.

2.3. Cuantificar

Ya hemos explicado que cuantificar una relación va más allá de describir y comparar dos variables (de esto ya se encarga el análisis univariante). Es importante remarcar que uno de los objetivos cruciales de la cuantificación bivariante es saber si los datos apoyan o no nuestra hipótesis inicial, que acostumbra a ser que las dos variables están relacionadas. En caso de que los resultados confirmen la hipótesis mediante los procedimientos que veremos a continuación, es muy importante no correr: esto no querrá decir que la hipótesis sea correcta. Si es correcta o no, nunca lo podremos saber en realidad. Lo único que querrá decir es que los resultados no parecen indicar que las variables son independientes entre sí y, por lo tanto, no tienen relación. Estas dos frases anteriores parecen un lío, pero tienen mucho sentido. Por eso es importante que entendamos el significado de la *hipótesis nula*, que es lo que realmente comprobamos estadísticamente en un análisis bivariante. La hipótesis nula acos-

tumbra a ser la hipótesis contraria a nuestra hipótesis inicial, también llamada *hipótesis alternativa*. Es decir, si decimos que hay relación entre *a y b*, la hipótesis nula que testaremos es que no hay relación entre *a y b*.

Hipótesis nula

Para entender la hipótesis nula, tenemos que partir de la base de que nunca podemos estar seguros de que lo que queremos probar sea cierto. Muchos hallazgos de grandes científicos que en su tiempo fueron un gran descubrimiento han sido falsificados por científicos más contemporáneos. Esto indica que tenemos que ir con cuidado con nuestra investigación y que, en lugar de probar que dos variables tienen relación, siempre es más prudente descartar lo contrario: que las dos variables no tienen relación. En el análisis de datos, pues, siempre comprobaremos la hipótesis nula, y los resultados nos permitirán descartar o no que no haya diferencia desde un punto de vista estadístico.

Cuando cuantificamos, lo que hacemos es tratar de aportar argumentos estadísticos para desmontar la hipótesis nula de que no hay relación entre las variables. El primer argumento que podemos aportar es la diferencia entre los porcentajes de la tabla de contingencia: observamos un 7,46 % de diferencia en el porcentaje de voto afirmativo respecto del voto negativo en temas de desarrollo entre Sudán del Sur y Sudán. Este argumento es más bien descriptivo y estadísticamente no es suficiente, aunque no deje de ser un número que nos pueda ayudar a explicar las diferencias entre categorías de la relación que analizamos.

Un segundo argumento que podemos aportar, más asociado con el análisis explicativo, es la fuerza de la relación entre las dos variables. Diremos que una relación entre variables cate-

góricas es muy fuerte cuando, por medio de los valores de la variable independiente, podemos saber con mucha exactitud los valores que tomará la variable dependiente. Hay varios métodos para medir la fuerza de la relación, como el test de Cramer o el de Lambda. La mayoría de estos métodos, que se llaman formalmente *coeficientes de asociación*, devuelven un valor cercano a 1 si la relación tiene mucha fuerza y un valor cercano a 0 si la relación no tiene fuerza. Para utilizar estas herramientas, tendremos que instalar y cargar el paquete *DescTools*. En el código siguiente hemos pedido el test de Cramer con `CramerV()`, donde dentro de la función introducimos un argumento por cada variable.

```
library(DescTools)

CramerV(sudan_ec_votes$vote, sudan_ec_votes$country)
```

Ejemplos de la fuerza de la relación

Si todos los países pobres del mundo fueran autocracias y todos los países ricos fueran democracias, entonces diríamos que la relación entre nivel de desarrollo y régimen político es muy fuerte. Es decir, si sabemos los valores de la variable *desarrollo* podremos adivinar con toda seguridad los valores de la variable *democracia*. Esto, sin embargo, raramente pasa, sobre todo, en Ciencias Sociales. Siempre tendremos casos de democracias pobres y autocracias ricas, así como también tendremos casos de personas ideológicamente de derechas que voten a partidos de izquierdas o de años en que los niveles de contaminación han aumentado, pero en cambio la temperatura del planeta ha disminuido.

La operación del código anterior nos habrá devuelto un coeficiente de asociación de Cramer cercano a 0,11. Esto significa que, si sabemos los valores de *x*, tenemos un 11 % de probabilidades de acertar los valores de *y*. Ciertamente, no es una probabilidad de acierto muy alta.

Aparte de la fuerza de la relación, la otra medida relevante para cuantificar una relación es saber hasta qué punto es significativo nuestro resultado. Volviendo al ejemplo anterior de la moneda, las conclusiones que podamos sacar sobre el comportamiento de una moneda si la lanzamos dos veces al aire serán poco significativas comparadas con las conclusiones que podamos sacar sobre la misma moneda si la lanzamos cien veces. Con el ejemplo anterior de los dos Sudanes nos pasa algo parecido. Supongamos que mañana hay una votación en la ONU y Sudán del Sur vota en contra de una resolución sobre desarrollo económico. Esto cambiaría los porcentajes de Sudán del Sur en la tabla de contingencia de una manera bastante significativa: los porcentajes pasarían a ser aproximadamente del 85 % a favor y del 15 % en contra, exactamente los mismos porcentajes que tiene Sudán. Si únicamente un solo caso adicional derroca nuestra hipótesis de que Sudán del Sur se ha querido diferenciar respecto de Sudán en política internacional, no podemos estar muy convencidos de que la diferencia observada en nuestro análisis sea generalizable de manera suficientemente significativa.

Para ayudarnos a decidir si el resultado obtenido es suficientemente significativo o no, la estadística tiene varias técnicas, que se denominan pruebas de significación, que nos ayudan a decidir si damos por buena la diferencia observada en nuestros datos. En otras palabras, un test de significación nos dice qué probabilidad hay de que la hipótesis nula sea cierta y que, en consecuencia, no haya realmente ninguna diferencia en nuestros resultados si los extrapoláramos a una población de casos más amplia.

El test de significación más habitual que se utiliza para tablas de contingencia es la prueba de Chi cuadrado de Pearson. Este test lo hacemos con la función `chisq.test()` y nos tendremos que fijar en el p-valor (*p-value*) de la última línea que devuelve la consola. Este valor nos indica la probabilidad de que la hipótesis nula sea cierta.

```
> chisq.test(sudan_ec_votes$vote, sudan_ec_votes$country)
X-squared = 0.0028326, df = 1, p-value = 0.9576
```

Según el test de significación que acabamos de aplicar, el 0,95 que hemos obtenido querrá decir que hay un 95 % de posibilidades de que la hipótesis nula sea cierta. En Ciencias Sociales, existe la convención de que los test de significación necesitan un p-valor inferior a 0,05 para rechazar la hipótesis nula. Un p-valor menor a 0,05 querrá decir que las probabilidades de que realmente no haya ninguna diferencia son muy bajas, inferiores a un 5 %, proporción que los científicos consideran suficiente para rechazar la hipótesis nula. En nuestro ejemplo, la hipótesis nula tiene más de un 95 % de probabilidades de ser cierta. Es un número demasiado alto, ya que necesitamos menos de un 5 % para descartarla y poder aceptar nuestra hipótesis alternativa. Diremos que no podemos asegurar con un 95 % de confianza que nuestra hipótesis sea cierta. Por tanto, en este caso, tendremos que aceptar la hipótesis nula.

Test de Fisher como alternativa

Otro test de significación para tablas de contingencias es el test de Fisher. Podemos pedirlo con la función `fisher.test()` y en el retorno a la consola también podremos observar un p-valor.

¿Estados Unidos vota de modo diferente si en el título de la resolución aparece la palabra «USSR»?

Nos hemos preguntado si el comportamiento de voto de Estados Unidos se escapa de lo habitual cuando aparece la palabra *USSR* en el título de la resolución. Por eso, hemos utilizado el paquete stringr para crear una nueva columna, que sea verdad cuando aparece USSR y falso cuando no aparece. Al acabar, hemos pedido la tabla de contingencia y el test de significación.

```
us_ussr_un <- un_roll_calls %>%
   mutate(ussr = str_detect(descr, "USSR")) %>%
   inner_join(un_votes) %>%
   filter(country_code == "US", vote != "abstain") %>%
   select(ussr, vote) %>%droplevels()


prop.table(table(us_ussr_un$vote, us_ussr_un$ussr), 2)
chisq.test(us_ussr_un$vote, us_ussr_un$ussr)
```

Podemos observar que la diferencia en el voto de Estados Unidos cuando en la resolución aparece «USSR» y cuando no aparece es significativa, con un 95 % de confianza. En otras palabras, la probabilidad de que realmente no haya diferencia en una población más elevada de casos es inferior al 5 %.

3. Relación entre variable categórica y numérica

Cuando queramos mirar la asociación entre una variable independiente categórica y una dependiente numérica utilizaremos la diferencia de medias. Para ilustrar esta técnica, en esta sección partimos de la hipótesis de que las democracias y las no democracias votan de modo diferente en la ONU. Lo que no tenemos claro es qué regímenes políticos votan más favorablemente las resoluciones en la AGNU, pero por ahora solo comprobaremos la hipótesis de que hay diferencia entre la media de voto favorable en las democracias en comparación con las no democracias.

Tenemos, pues, una variable independiente categórica, si un país es una democracia o no, y una dependiente numérica, la media de voto afirmativo de cada país por año. Para comprobar nuestra hipótesis, utilizaremos y juntaremos la base de datos unvotes con la Democracy Dictatorship (DD) dataset, la clasificación dicotómica de democracia de Cheibub y otros (2010). De la base de datos DD nos interesan las variables *democracy* y *un_continent_name*, que obtendremos con el código que podemos encontrar en el anexo del capítulo.[1]

El objetivo es construir un solo marco de datos que separe los países por año, que diga si cada año el país era una democracia o no lo era, y cuál es la proporción anual de votos favorable de cada país. Primero, hemos limpiado los marcos de datos que necesitamos de unvotes, transformando la variable *year* y convirtiendo en binaria la variable *vote*, de forma que adoptará el valor 1 cuando el voto sea favorable y el valor 0 en cualquier otro caso. Una vez

1. También podéis consultar códigos adicionales en la web del autor (https://www.jordimas.cat/courses/).

convertida en binaria, hemos pedido la media agrupada por año y país para obtener la proporción de voto favorable sobre los votos totales.

```
un_year_mean <- un_roll_calls %>%
    separate(date, "year", extra = "drop") %>%
    inner_join(un_votes) %>%
    mutate(year = as.numeric(year), vote = if_else(vote == "yes",
    1, 0)) %>%
    group_by(year, country, country_code) %>%
    summarize(mean = mean(vote))
```

Fijémonos en el marco de datos `un_year_mean` que hemos creado. Ahora ya tenemos la tabla limpia con los datos de unvotes que necesitamos. El segundo paso es unir el marco de datos `DDdata` por medio de las columnas que tienen en común: por código de país y por año. A continuación, seleccionamos las columnas que nos interesan y transformamos la variable binaria *democracy* en las categorías «Democracy» y «No Democracy».

Evitar unir los datos por nombre de país

Unir marcos de datos por el nombre del país es peligroso porque no podemos estar seguros de que los nombres estén tipificados igual. Por ejemplo, hay marcos de datos que aluden a Estados Unidos como *United States* y los que los llaman *United States of America*. Esto nos crearía dos categorías diferentes cuando, realmente, nos refiramos a la misma. Por tanto, siempre será mejor unir marcos de datos por medio de un código estandarizado de país o variables estables como el año.

```
un_dd_year <- un_year_mean %>%
    inner_join(DDdata, by = c("country_code" = "iso2c", "year"
                              = "year")) %>%
    select(country = country.x, country_code, year, mean,
           democracy, continent = un_continent_name) %>%
    mutate(democracy = if_else(democracy == 1, "Democracy",
                               "No democracy"))
```

El marco de datos `un_dd_year` está formado por las variables necesarias para visualizar y cuantificar una diferencia de medias. Tenemos una variable con el año, otra con la media de cada país por año y otra que indica si el país era o no una democracia.

3.1. Visualizar

Hay varias maneras de visualizar una diferencia de medias. Podemos, por ejemplo, utilizar un diagrama de barras en el que la altura de las barras represente la media de cada categoría. Dos herramientas también comunes para representar visualmente la diferencia de medias son el diagrama de cajas y el diagrama de dispersión. En este apartado, utilizaremos el diagrama de dispersión porque creemos que es una herramienta con más posibilidades visuales y que nos permite representar más variables en el gráfico. Si el código siguiente no os funciona, intentad cargar el paquete Hmisc.

```
un_dd_year %>%
    filter(year == 1946) %>%
    ggplot(aes(x = democracy, y = mean, col = continent)) +
```

```
    geom_jitter(alpha = 0.5) +
    stat_summary(geom = "point", fun.data = mean_se,
                 col= "red", size = 1.2) +
    stat_summary(geom = "errorbar", fun.data = mean_se,
                 col = "red", width = 0.1) +
    theme_classic()
```

Con el código anterior, hemos creado el diagrama de dispersión con *jittering* que observamos en la figura 21 y que contiene la variable categórica *democracy* en el eje de las *x* y la numérica *mean* en el eje de las *y*. Hemos querido saber si el primer año que empezaron las votaciones en la AGNU, en 1946, había diferencia entre la orientación del voto entre las democracias y las no democracias. Por eso hemos pedido un sumario estadístico de la media que toma la variable dependiente para cada categoría de la variable independiente, representado con un punto rojo. También hemos pedido un estadístico denominado *error típico de la media*, representado con líneas rojas, que nos da el intervalo donde podemos asegurar que se encuentra realmente la media con un 95 % de confianza.

Error típico de la media

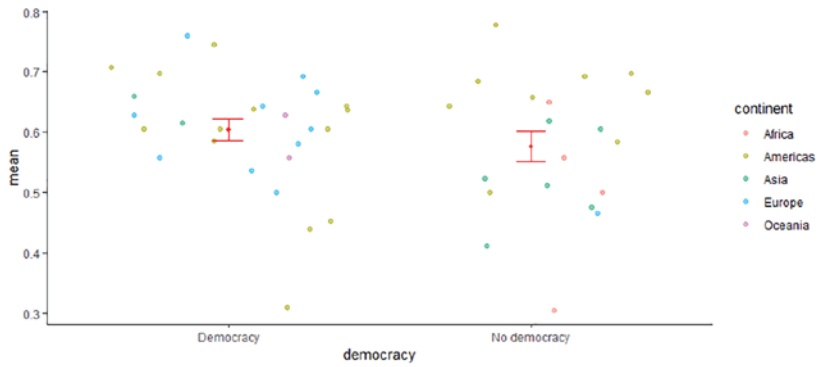
El error típico de la media nos ayuda a estimar en qué intervalos se puede encontrar realmente la media en la población según un nivel de confianza determinado. Cuantos menos casos tengamos y cuanto más dispersos estén estos casos alrededor de la media, mayor será el error típico y, por lo tanto, menos precisión tendremos para saber dónde se encuentra la media. Para obtener el intervalo donde se encuentra la

media con un 95% de confianza, tendremos que sumar dos errores típicos a la media y restarle dos errores típicos.

Evitar los diagramas de barras

Si sustituimos la altura de la barra por un punto tendremos la misma información y habremos gastado menos tinta. Es por eso por lo que los diagramas de barras no son una buena herramienta para visualizar la diferencia de medias. Utilizaremos siempre diagramas de cajas, que nos permiten ver más descriptivos en el gráfico, o diagramas de dispersión, que nos permiten observar la distribución de la variable dependiente y añadir más variables al gráfico si lo creemos oportuno. En algunos GitHub en internet, encontraréis comparaciones entre tipos de representaciones visuales.

Figura 21. Diferencia de media de voto por régimen político en 1946



Fuente: elaboración propia.

Fijémonos en que, de media, las democracias votaban ligeramente más a favor que las no democracias en las resoluciones de la AGNU en 1946. Pero fijémonos también en que

los intervalos de confianza parece que se encabalguen. Por lo tanto, aunque apreciamos una ligera diferencia entre las dos categorías, esta diferencia no es estadísticamente lo bastante significativa. Esto quiere decir que, si hubiera habido muchas más votaciones en 1946, es demasiado probable, según nuestros estándares científicos, que la diferencia que acabamos de observar realmente no existiera en una población más amplia de casos.

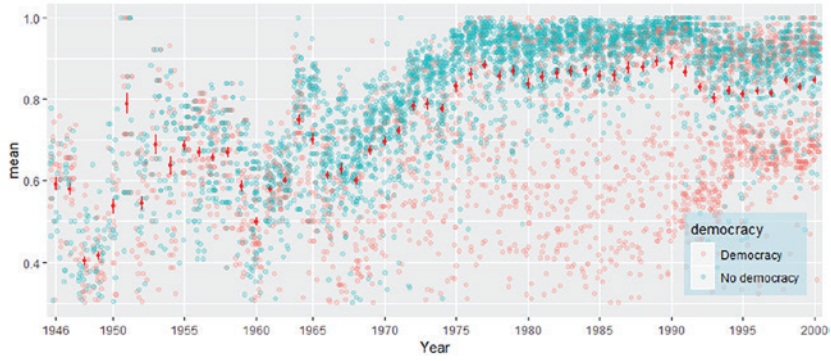
Hemos generado la figura 21 filtrando el código por el año 1946. Probad a modificar el código filtrando por otros años o por toda la muestra de años, y comprobad si hay diferencia de media entre el voto de democracias y no democracias en la AGNU. Intentad contestar si los resultados apoyan o no nuestra hipótesis inicial.

Cambios de tendencia en el tiempo

La diferencia de medias se utiliza especialmente para ver cambios de tendencia a lo largo del tiempo. Por ejemplo, podemos ver en qué años podemos decir que ha habido un cambio significativo de tendencia en el voto en la AGNU y en cuáles no podemos asegurar que el cambio observado sea generalizable con un nivel de confianza del 95%. En la figura 22 podemos observar en qué años ha habido un cambio lo bastante significativo de tendencia en el voto y cuáles no.

Fijémonos en los puntos azules y rojos. Todo parece indicar que la mayoría de años las no democracias votan más a favor de las resoluciones de la AGNU que las democracias.

Figura 22. Análisis longitudinal con diferencia de medias



Fuente: elaboración propia.

3.2. Cuantificar

Una de las ventajas de visualizar la diferencia de medias es que podemos observar las líneas rojas en que representamos los intervalos de confianza y, por lo tanto, podemos obtener una intuición visual bastante clara de los niveles de significación sin necesidad de cuantificar. Sin embargo, es pertinente ser precisos en nuestros análisis y tener también la estimación de la diferencia de medias cuantificada numéricamente. Esta cuantificación nos tiene que permitir responder a dos preguntas:

1) Si hay diferencia estadística en la media de la variable dependiente según las categorías de la variable independiente. Para encontrar si la respuesta es afirmativa o negativa, tendremos que mirar si podemos rechazar o no la hipótesis nula.

2) Saber cuánta diferencia hay entre una media y la otra desde un punto de vista estadístico, por lo cual también nos tendremos

que mover con intervalos de confianza para determinar dónde se encuentra la media en la población real.

Para responder ambas preguntas, utilizaremos la función `t.test()` donde introduciremos los vectores de las dos variables que estamos analizando en la figura 21. La primera pregunta se responde con el p-valor. Un p-valor inferior a 0,05 significará que podemos rechazar la hipótesis nula y afirmar con un 95 % de confianza que la diferencia de medias observada existe en la realidad. La segunda respuesta se responde con el intervalo de confianza en el 95 %, que se obtiene a partir de sumar y restar dos errores típicos a la media.

```
un_dd_year_1946 <- un_dd_year %>%
   filter(year == 1946) %>%
   mutate(democracy = if_else(democracy == "Democracy", 1, 0))


> t.test(un_dd_year_1946$mean, un_dd_year_1946$democracy)
data:  un_dd_year_1946$mean and un_dd_year_1946$democracy
t = 0.23618, df = 49.996, p-value = 0.8143
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
-0.1319858 0.1671619
sample estimates:
mean of x mean of y
0.5920561 0.5744681
```

Como podemos ver en el resultado general para este código, tenemos un p-valor muy superior a 0,05. Es decir, tal como ya

nos había confirmado visualmente el gráfico de la figura 21, no podemos asegurar que la diferencia entre medias no sea cero. Por lo tanto, en respuesta a la primera pregunta, no podemos asegurar con un 95% de confianza que haya diferencia en la orientación de voto entre democracias y no democracias en 1946.

En la segunda pregunta queríamos saber cuál era la diferencia real estimada entre las medias. La respuesta la encontramos en los números -0,13 y 0,16. Estos números indican el intervalo en que estimamos que se moverá la diferencia de medias en la población real con un 95% de confianza. Es decir, con la muestra de resultados que tenemos, podemos apostar con un 95% de acierto que la diferencia de medias en la población real se encontraría en el intervalo siguiente: en un extremo, podría ser que las democracias hayan votado afirmativamente un 16 % más que las no democracias, pero, en el otro extremo, podría ser que las no democracias hayan votado de forma afirmativa un 13 % más que las democracias. Como en realidad no lo podemos saber, lo único que podemos decir es que, si apostamos que la media real está entre estos dos intervalos, tenemos un 95 % de probabilidades de acertar.

Fijaos en que la posibilidad de que la diferencia de medias sea cero entra dentro de nuestros pronósticos. En esta situación, no podemos rechazar la hipótesis nula, puesto que es demasiado probable que sea cierta. Para contrastarlo con un caso en que sí que hay diferencia, probad a pedir el registro de voto[2] entre los años 1995 y 2000.

2. Si no encontráis la manera de pedir este registro de voto, encontraréis el código de visualización en el anexo.

El resultado de pedir el *t-test* en este caso es el siguiente:

```
data: un_dd_year_95$mean and un_dd_year_95$democracy
t = 16.575, df = 1245.1, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
0.2329403 0.2954871
sample estimates:
mean of x mean of y
0.8203988 0.5561851
```

En este caso, el p-valor es inferior a 0,05 y, por lo tanto, sí podemos rechazar la hipótesis nula. Parece, pues, que entre los años 1995 y 2000 hay una diferencia bastante significativa entre el tipo de régimen político y la orientación del voto. Concretamente, podemos estimar, con un 95% de confianza, que la diferencia real en la población se sitúa entre 0,23 y 0,29. En otras palabras, el voto de las no democracias en la AGNU es afirmativo, de media, entre un 23% y un 29% en relación con las democracias. En este caso, el 0% no figura entre nuestros pronósticos. Como la diferencia de medias no puede ser cero, podemos rechazar la hipótesis nula.

4. Relación entre variables numéricas

Cuando queremos explicar la relación entre dos variables numéricas, necesitamos varios instrumentos, tanto visuales como cuantitativos. Visualmente, la manera de representar gráficamen-

te estas dos variables es mediante un diagrama de dispersión, un gráfico de dos dimensiones en que las coordenadas de cada punto representan el valor que toma cada observación según las variables independiente y dependiente. Situaremos siempre la variable independiente *x* en el eje horizontal y la variable dependiente *y* en el eje vertical. El diagrama de dispersión se suele complementar con la recta de regresión, una técnica tanto visual como cuantitativa que nos permite dibujar la línea que mejor se ajusta a la relación que observamos en el gráfico y cuantificarla. Para cuantificar la relación, también utilizaremos el coeficiente de correlación, que describe de manera cuantitativa la fuerza y la dirección de la relación, y el coeficiente de determinación, que ya conocemos.

En conjunto, con las técnicas mencionadas podemos llegar a saber cinco aspectos principales de la asociación entre dos variables numéricas. Estos aspectos los resumimos aquí y los ampliamos en los apartados siguientes:

1) Forma: se refiere al dibujo que hace la relación entre *x* e *y*. Lo llamaremos *lineal* si esta forma dibuja una línea más o menos recta. Si, en cambio, este dibujo toma una forma curva, lo llamaremos *no lineal* o *cuadrática*.

2) Fuerza: nos indica cuán perfecta es la forma que hacen los puntos en el gráfico. Si, por ejemplo, las coordenadas de *x* y de *y* describen una línea recta perfecta, diremos que la fuerza de la relación es muy alta. A medida que la línea imaginaria que forman los puntos se desdibuja, tendremos unos puntos más dispersos y una relación con poca fuerza.

3) Dirección: diremos que la dirección es *positiva* cuando las dos variables se muevan en la misma dirección. Es decir, cuando una incrementa sus valores la otra también lo hace. Diremos

que la relación es *negativa* cuando las dos variables se mueven en dirección opuesta. Es decir, cuando los valores de una variable se incrementan, los valores de la otra disminuyen. Si no apreciamos una dirección positiva o negativa, diremos que es una *relación plana*.

4) Significación: la relación entre *x* e *y* es estadísticamente significativa si las probabilidades de que no haya relación, es decir, de que la línea sea plana, sean inferiores al 5 %.

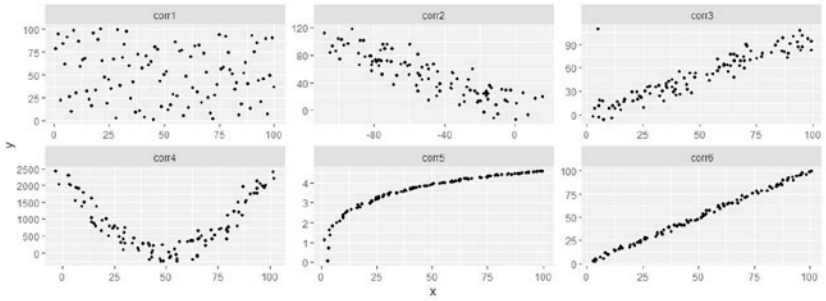
5) Casos extremos: diremos que la relación tiene casos extremos si hay observaciones que están muy alejadas de la forma que toma la relación.

4.1. Visualizar

La manera de visualizar una relación entre variables numéricas es por medio de un diagrama de dispersión, que podemos llamar con R por medio del paquete ggplot2 y la geometría `geom_point()`. En la figura 23 vemos seis ejemplos de relaciones entre variables con diferente forma, fuerza, dirección y presencia de casos extremos:

- en el gráfico *corr1,* observamos una relación sin forma, sin fuerza y de dirección plana;
- en el caso de *corr2,* hay una relación lineal, de fuerza débil y dirección negativa;
- *corr3* tiene una forma lineal, de fuerza mediana, dirección positiva y con un caso extremo;
- el caso de *corr4* es una relación no lineal, de fuerza media;
- *corr5* tiene una relación no lineal, muy fuerte y positiva;
- *corr6* tiene una relación lineal, muy fuerte y positiva.

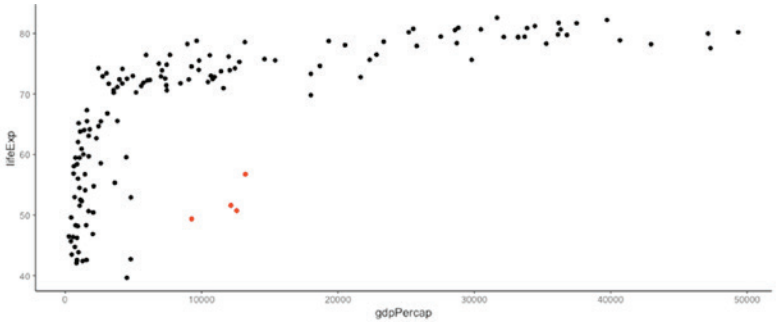
Figura 23. Visualización de la correlación entre variables



Fuente: elaboración propia.

Caso real de relación no lineal con casos extremos

Uno de los fenómenos más investigados en los estudios de desarrollo es la relación existente entre el PIB per cápita de un país y su esperanza de vida. Como observamos en los datos de 2007, la relación es no lineal, fuerte y positiva. También podríamos considerar la presencia de algunos casos extremos que se encuentran con más de 10.000 dólares per cápita y esperanza de vida alrededor de los 50 años, que hemos marcado en rojo. ¿Sabríais encontrar cuáles son estos cuatro casos? Encontraréis el código en los anexos.



Fuente: elaboración propia.

La línea de regresión es una técnica visualmente muy utilizada para representar la forma y la dirección de la relación entre dos variables numéricas. Hay diferentes métodos para representar esta relación, entre ellos el método Ordinary Least Squares (OLS) y el método Locally Estimated Scatterplot Smoothing (LOESS). La principal diferencia entre los dos es que, mientras que la línea OLS es una línea completamente recta, la LOESS se intenta ajustar a la distribución de los puntos. Nosotros utilizaremos preferentemente la línea OLS.

La característica principal de la línea OLS es que intenta minimizar con una línea recta la suma de la distancia al cuadrado entre la línea y los puntos. En otras palabras, es la recta que de media pasa más cerca de todos los puntos del gráfico. Cualquier otra línea que queramos trazar no conseguirá minimizar esta distancia. La línea OLS se representa con la función `geom_smooth()` junto con la geometría del diagrama de dispersión. Dentro de la función tendremos que especificar el método lm (*linear model*) y preferiblemente pondremos el error típico `se` (*standard error*) en FALSE, como mostramos en este código creado a partir del marco de datos `gapminder`.

```
gapminder %>%
    filter(year > 1990, lifeExp > 30) %>%
    ggplot(aes(gdpPercap, lifeExp)) +
        geom_point(aes(col = continent), alpha = 0.7) +
        geom_smooth(method = "lm", se = FALSE)
```

En este código, representado en la figura 24, hemos incluido el estético de color en el diagrama de dispersión, pero no lo hemos incluido en la recta de regresión OLS porque no queremos que genere una línea para cada continente. Podemos ver que

la relación entre el PIB per cápita y la esperanza de vida dibuja una forma más bien curva, positiva y más bien fuerte.

Figura 24. Diagrama de dispersión con recta OLS



Fuente: elaboración propia.

La línea LOESS es una herramienta utilizada también en análisis de regresión para ver la relación entre dos variables numéricas. Esta técnica es especialmente útil en líneas temporales, puesto que nos permite hacer previsiones de tendencia. Para ver la LOESS en el gráfico anterior, solo hace falta que eliminemos el argumento `method = "lm"`, puesto que `geom_smooth()` ya nos dibuja una LOESS por defecto. También podéis indicarlo explícitamente con `method = "loess"`. En la figura 25, visualizamos una LOESS con los atributos que tiene por defecto, entre ellos el intervalo de confianza al 95 % y la sensibilidad de la línea:

Sensibilidad de la línea y error típico

Podemos cambiar la sensibilidad de la línea con el argumento `span`, donde podemos indicar un número entre 0 y 1 y Cercano a 0, la sensibilidad será más alta y la línea tendrá cambios más abruptos. Cercano a 1, la línea será más armónica. El argumento `se` nos marcará el intervalo de confianza de la línea al 95 % de confianza.

Figura 25. Diagrama de dispersión con línea LOESS



Fuente: elaboración propia.

Podemos comprobar que la LOESS establece una línea suave que se va adaptando según la situación de los puntos en el gráfico. Las zonas grises marcan el intervalo de confianza de la línea. Podéis ver que el intervalo será más estrecho en las zonas de los gráficos donde haya más puntos y estén más juntos, mientras que el intervalo será mayor en las zonas donde menos puntos haya y más separados estén.

4.2. Cuantificar

Con un diagrama de dispersión ya podemos intuir visualmente algunos parámetros como la fuerza y la dirección de una relación entre variables numéricas, pero la cuantificación nos será más útil para plantearnos preguntas más precisas, como por ejemplo: «¿cuán fuerte?». Para poder cuantificar estos parámetros, tenemos principalmente dos medidas numéricas:

- el coeficiente de correlación, y
- el coeficiente de determinación.

El coeficiente de correlación, que a menudo veremos ilustrado con el símbolo *r*, es la manera que utilizaremos para cuantificar la fuerza y la dirección de una relación lineal entre dos variables numéricas. Cuantificar relaciones por medio del coeficiente de correlación nos permite comparar numéricamente las relaciones entre ellas. Este coeficiente es un número que varía entre -1 y 1. El signo nos indica si la relación es positiva o negativa. La magnitud del número se corresponde con la fuerza de la relación. Por lo tanto, un número cercano a 1 indicará una relación muy fuerte y positiva. Un número cercano a -1 indicará una relación muy fuerte y negativa, mientras que un número cercano a 0 indicará que no hay relación entre las dos variables.

Volvamos a fijarnos en la figura 23 e intentemos deducir el coeficiente de correlación de cada una de las relaciones. Después podemos comprobar si nos hemos acercado mediante la función `cor()`, que calcula el coeficiente de correlación a partir de dos argumentos: una variable en el primer argumento y la otra variable en el segundo argumento. No importa el orden en que especifiquemos las variables, puesto que las relaciones son simétricas y nos darán el mismo coeficiente. A continuación, hemos imprimido los coeficientes de correlación de cada uno de los seis gráficos. Como vemos, la relación más fuerte y también positiva la encontramos en *corr6*. También son relaciones fuertes y positivas *corr3* y *corr5*. En el caso de *corr1* y *corr4,* no hay relación entre las variables, mientras que en *corr2* la relación es fuerte y negativa.

```
      corr1     corr2     corr3     corr4     corr5     corr6
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     -0.0563   -0.843    0.923     0.0589    0.894     0.999
```

Es importante tener en cuenta que el coeficiente de correlación no puede capturar con precisión la fuerza de una relación no lineal. Es por eso por lo que, en el caso de *corr5*, donde los puntos describen una línea curva casi perfecta, no encontramos un coeficiente de correlación tan elevado como *corr3* o *corr6*. Para incrementar el coeficiente en el caso de estas relaciones cuadráticas, podemos buscar, por ejemplo, el logaritmo neperiano de una de las variables.

De una manera más general, podemos aplicar el coeficiente de correlación para ver cómo están relacionados, en tres continentes diferentes, el PIB per cápita y la esperanza de vida en varios años. A continuación, hemos pedido un sumario con el coeficiente de correlación por seis años diferentes en Europa, África y Asia.

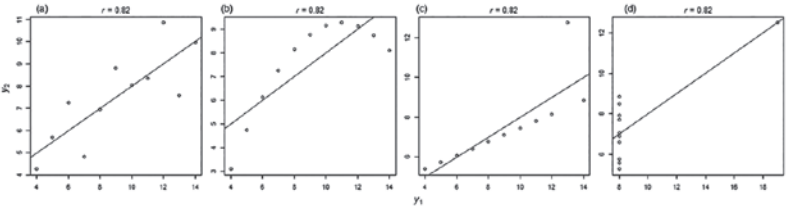
```
> gapminder %>%
    filter(year %in% c(1952, 1962, 1972, 1977, 1982, 1992, 2002),
           continent %in% c("Europe", "Africa", "Asia")) %>%
    group_by(continent, year) %>%
    summarize(cor = cor(gdpPercap, lifeExp), N = n()) %>%
    spread(year, cor)
# A tibble: 3 x 9
# Groups:   continent [3]
  continent  N     `1952` `1962` `1972` `1977` `1982` `1992` `2002`
  <fct>      <int>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Africa     52     0.322  0.346  0.488  0.570  0.663  0.684  0.515
2 Asia       33     0.513  0.555  0.621  0.649  0.710  0.815  0.814
3 Europe     30     0.865  0.795  0.747  0.746  0.722  0.770  0.846
```

En el código también hemos pedido un sumario del número de observaciones. Así podemos saber cuántos casos tenemos

en cada correlación e inferir hasta qué punto pueden ser significativos estos resultados. Cuantos más casos tengamos, más significativas acostumbran a ser las observaciones. En todas las combinaciones de la tabla, la relación entre variables es positiva aunque siempre ha sido más fuerte en Europa que en los otros continentes. En África esta correlación era especialmente baja a mediados del siglo pasado, pero ha aumentado ligeramente a lo largo del tiempo.

Francis Anscombe y el problema de cuantificar relaciones

Cuantificar relaciones lineales también puede tener inconvenientes, tal como detectó el profesor de estadística Francis Anscombe con una base de datos sintética construida en 1973. Ilustró cómo relaciones completamente diferentes podían tener el mismo número de puntos, la misma media y desviación típica por *x* e *y,* y la misma correlación. Fijémonos en los cuatro diagramas de dispersión siguientes, obtenidos por medio del marco de datos `anscombe`, donde los cuatro tienen una *r* de 0,82.



Fuente: elaboración propia.

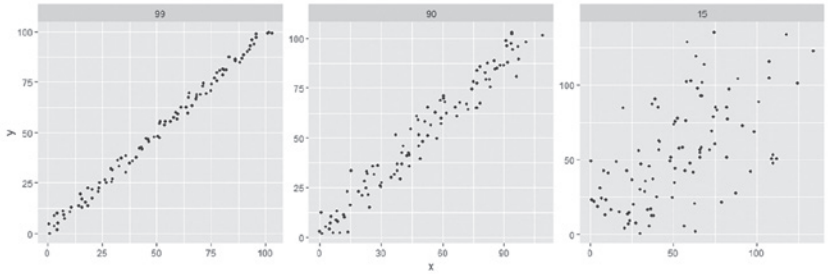
La conclusión más importante que podemos sacar de estos cuatro gráficos es que, aunque *r* nos pueda ayudar a cuantificar numéricamente una relación entre dos variables, siempre será pertinente visualizar la relación para evitar que los números nos jueguen una mala pasada. Por

ejemplo, en el último caso vemos que no hay ningún tipo de relación entre *x* e *y*, pero un solo caso extremo nos distorsiona el coeficiente de correlación. Podéis encontrar este marco de datos en los paquetes de base de R.

La segunda manera de poder cuantificar la relación entre dos variables es por medio del coeficiente de determinación, representado con R^2, que es un número entre 0 y 1 que nos determina la proporción de la variación en la variable dependiente que se puede explicar con la variación de los valores de la independiente. La suerte que tenemos es que R^2 es muy fácil de calcular. Solo hay que elevar al cuadrado el coeficiente de correlación. Lo que es más complicado, sin embargo, es comprender el significado de este número.

Para entender mejor cómo interpretar R^2, podemos considerar que su valor responde a una pregunta: ¿qué porcentaje de la variabilidad de *y* podemos explicar si sabemos la variabilidad de *x*? Fijémonos en la figura 26. A la izquierda vemos una correlación casi perfecta. Sabiendo cómo varían los valores de *x,* podríamos adivinar muy cuidadosamente los valores de *y*. Por ejemplo, si nos dicen que el valor de *x* es 64, sabemos que el valor de *y* también estará alrededor de 64 y no estaremos muy lejos de equivocarnos. Si nos preguntan cuán seguros estamos de acertar, entre 0 y 1, diremos que estamos un 0,99 convencidos de que no nos equivocaremos. La R^2 es 0,99 porque, a sabiendas de la variabilidad de *x*, podemos explicar un 99 % de la variabilidad de *y*. En el gráfico de la derecha, en cambio, nos será mucho más difícil acertar los valores de *y* si conocemos los valores de *x*. Si nos dicen que la *x* es 90, podremos estimar con mucha menos precisión los valores de *y*. Diríamos que nuestra R^2 estará alrededor de 0,15.

Figura 26. Coeficiente de determinación de tres correlaciones



Fuente: elaboración propia.

Cuando nos referimos a R^2, hablaremos siempre en porcentajes y nos expresaremos de una manera técnica. Por ejemplo, para describir el gráfico del medio de la figura anterior, diremos que «un 90 % de la variabilidad de la variable dependiente puede ser explicada por la independiente».

Hay dos maneras de encontrar el coeficiente de determinación. La primera es elevar el coeficiente de correlación al cuadrado, como mostramos en el código siguiente. Podéis encontrar los códigos de la figura 26 en el anexo.

Cuidado a la hora de interpretar R^2

Tenemos que ir con cuidado a la hora de interpretar R^2. Un coeficiente de determinación bajo no quiere decir necesariamente que tengamos un mal modelo y menos en Ciencias Sociales. A menudo podemos tener una R^2 baja, pero este pequeño porcentaje de explicación puede ser una aportación estadísticamente significativa por un problema complejo.

```
cor(x, y)^2
```

La segunda manera que tenemos para encontrar R^2 es a partir del modelo de regresión, destinado a construir modelos lineales, que explicamos en el apartado siguiente.

4.3. Modelar

Después de aprender a visualizar y cuantificar una correlación, en este apartado veremos cómo crear un modelo de regresión a partir de los datos disponibles. *Modelar* significa crear una fórmula matemática que nos permita predecir el valor de una variable si sabemos el valor de la otra. Esto lo podemos hacer con la recta de regresión OLS, que, como ya hemos explicado, tiene una doble virtud visual y cuantitativa. Visualmente, el método OLS nos muestra en un gráfico cuál es la manera más ajustada de describir la relación entre dos variables numéricas mediante una línea recta. De manera cuantitativa, la recta OLS nos permite predecir qué valor tomaría *y* cuando la recta pasa por los valores de *x*. Fijémonos en la figura 27. La recta nos ayuda a hacer una predicción del comportamiento de las variables *x* e *y* del marco de datos modelo, que encontraréis disponible con código en el anexo. La recta nos indica, por ejemplo, que si *x* tiene un valor de 2,5, *y* tendría que tener un valor cercano a 2.

En lugar de hacer estas estimaciones de manera visual, podemos cuantificar la recta OLS y obtener el modelo matemático a partir de dos parámetros: la inclinación de la recta de regresión y el valor que toma *y* cuando *x* es cero.

1) Por *inclinación* nos referimos a cuánto crece o decrece *y* si aumentamos *x* en una unidad. Si nos fijamos en el gráfico anterior, cuando *x* pasa de 0 a 1, *y* cambia aproximadamente en

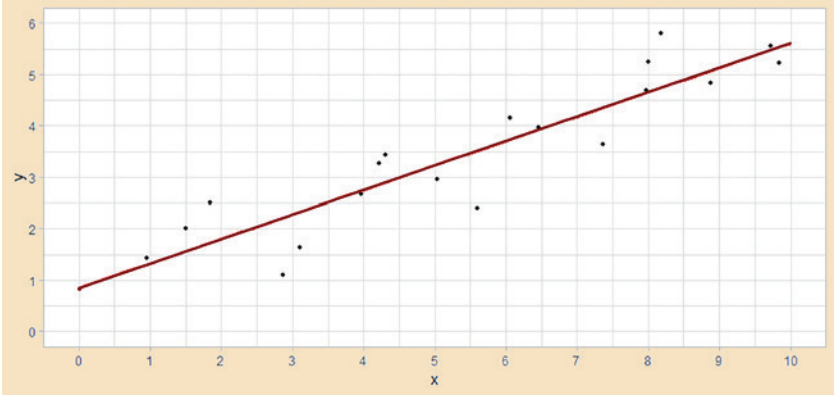
medio punto, puesto que el valor de *y* pasa de algo menos de 1 a algo menos de 1,5. Entonces, la inclinación, que es la misma para toda la recta de regresión, será aproximadamente de 0,5.

Interpretar correctamente la inclinación

Para interpretar correctamente el valor de la inclinación habrá que tener muy claro cuáles son las unidades de *x* y de *y*. Si por ejemplo la variable *x* son años e *y* son dólares per cápita, lo leeremos como «cada año (unidad de *x*) hay un cambio de dólares per cápita (unidad de *y*) del número que marque la inclinación».

2) El valor que toma *y* cuando *x* es 0 lo llamaremos *constante* o *intercept*. A primera vista, podemos ver cómo la constante de esta regresión es un poco inferior a 1, puesto que es el punto por donde la recta corta el eje de las *y* cuando la *x* es igual a 0.

Figura 27. Modelo lineal



Fuente: elaboración propia.

Para determinar con precisión estos dos parámetros, tendremos que utilizar la función `lm()`, donde indicaremos como

primer argumento el nombre de la variable dependiente seguido de una virgulilla y el nombre de la dependiente. El segundo argumento es el marco de datos. Si obtenemos el modelo de regresión del marco de datos `model`, vemos que no nos equivocábamos mucho. La inclinación es de 0,475 y la *intercept* es de 0,854.

El nuevo tipo de objeto *lm*

La función `lm()` nos devuelve un tipo de objeto que no hemos visto hasta ahora de clase *lm*, como podemos observar con la función `class()`. Si pedimos el `typeof()`, veremos que es una lista, otro tipo de objeto como sería un vector o un marco de datos. Para transformar este objeto en vector, pondremos toda la operación dentro de la función `coef()`.

```
> lm(formula = y ~ x, data = model)
(Intercept)         x
      0.854      0.475
```

Con este resultado ya tenemos los dos datos que necesitamos para interpretar el modelo. Si nos fijamos en el *intercept*, diremos que cuando la *x* es 0, *y* es 0,854. Si nos fijamos en la inclinación, diremos que la *y* crece en 0,475 unidades por cada incremento de una unidad de *x*. Estos dos valores los podemos utilizar para crear un modelo predictivo que nos permitirá conocer una estimación de los valores de *y* si sabemos el valor de *x*. En el código siguiente, podemos predecir los valores de *y* con una fórmula sencilla, en que multiplicamos la pendiente por el valor de *x* y sumamos la constante.

```
y = intercept + x * pendiente
y = 0.854     + x * 0.475
```

Podemos probar nuestro modelo predictivo sustituyendo la *x* por diferentes números. Por ejemplo, si la sustituimos por 5, la *y* será 3,229 como comprobaremos si entramos el código `0.854 + 5 * 0.475` en la consola. Si la *x* es 10, la *y* será 5,604.

Aparte del *intercept* y la pendiente, hay mucha más información que podemos obtener a partir del modelo lineal. En el código siguiente hemos creado primero el objeto `lm_model` y después hemos generado la información adicional a partir de la función `summary()`. R no solamente nos genera la *intercept* y la pendiente, sino que también nos proporciona información del modelo de regresión como los datos utilizados, la especificación del modelo, los residuos, el error estándar asociado, la significación estadística o el R^2 múltiple y el R^2 ajustado.

```
lm_model <- lm(formula = y ~ x, data = model)
> summary(lm_model)
Call:
lm(formula = y ~ x, data = model)

Residuals:
        Min          1Q      Median          3Q         Max
   -1.11730    -0.30072     0.03741    0.42807     1.07039

Coefficients:
                Estimate   Std. Error    t value      Pr(>|t|)
(Intercept)      0.85402      0.32640      2.616      0.0181 *
x                0.47501      0.05263      9.026    6.8e-08 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.6241 on 17 degrees of freedom
Multiple R-squared: 0.8274, Adjusted R-squared: 0.8172
F-statistic: 81.47 on 1 and 17 DF, p-value: 6.799e-08
```

En las líneas siguientes solo describiremos algunos de los elementos de este sumario. Al inicio de este nos aparece la fórmula que hemos utilizado, mientras que en la columna *Estimate* de la tabla de coeficientes podemos ver el *intercept* y la pendiente. El sumario también nos muestra el R^2 (*R-squared*), con la diferencia de que aquí vemos el R^2 múltiple y el R^2 ajustado. Si buscamos el R^2 de la manera que hemos explicado anteriormente por medio de `cor(model$x, model$y)^2`, veremos que su resultado coincide con el R^2 múltiple, que es la versión que utilizaremos en análisis bivariante para interpretar el R^2.

El elemento que nos faltaba ver, y que ya hemos visto en las otras técnicas de análisis bivariante, es el test de significación estadística, una convención matemática que nos permite descartar que los efectos observados en una relación lineal sean fruto del azar. Es decir, cuando observamos una relación entre dos variables, sea positiva o negativa, nos tenemos que preguntar si es posible que la relación observada no sea real cuando extrapolamos nuestra muestra a la realidad. Si no es real, quiere decir que hay una probabilidad superior al 5 % de que la pendiente de la recta de regresión sea cero y que, por lo tanto, la hipótesis nula sea cierta. Sabemos que, en Ciencias Sociales, normalmente se establece un valor inferior en 0,05 para rechazar la hipótesis nula y así descartar el efecto del azar en nuestros resultados. La significación estadística la encontramos indicada en la columna con nombre *Pr(>|t|)* de la tabla de coeficientes. Veamos la tabla nuevamente:

```
Coefficients:
                Estimate   Std. Error    t value      Pr(>|t|)
(Intercept)      0.85402      0.32640      2.616      0.0181 *
x                0.47501      0.05263      9.026   6.8e-08 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ejemplo de significación estadística

Supongamos que hemos viajado dos veces en la vida a Málaga y las dos veces ha llovido. Si, de acuerdo con nuestra experiencia, afirmamos que en Málaga siempre llueve, ¿qué validez tendría esta afirmación? Es muy posible que si incrementamos el número de experiencias en Málaga, nuestra afirmación sea más difícil de sustentar, puesto que encontraremos algún día que no llueva. Por lo tanto, es muy posible que nuestra afirmación inicial haya sido fruto del azar.

En la tabla de coeficientes nos fijaremos en el valor de *x* en la tabla *Pr(>|t|)*. Vemos que el número que nos aparece es 0,000000068, lo cual cumple el requisito de que el valor en cuestión sea inferior a 0,05. Por lo tanto, podemos descartar la hipótesis nula y asegurar con un 95 % de confianza que los resultados observados no son fruto del azar. Si los valores son superiores a 0,05, diremos que la relación no es estadísticamente significativa.

Interpretar los asteriscos

Para facilitar la lectura de los resultados, tenemos la significación de los coeficientes señalados con asteriscos. Si, como nuestro caso, la signifi-

cación está entre 0 y 0,001, se indicará con ***. Si la significación está entre 0,001 y 0,01 se indicará con ** mientras que si está entre 0,01 y 0,05 se indicará con *. En estos tres casos, diremos que la relación entre *x* e *y* es estadísticamente significativa.

5. Regresión múltiple

Como decíamos en la introducción de este capítulo, podemos argumentar que dos variables están asociadas si cumplen varios requisitos. En las anteriores secciones hemos aprendido, principalmente, a dar argumentos estadísticos por medio de la cuantificación, por ejemplo, de la fuerza y el nivel de significación de una relación. Estos argumentos nos permiten decir si es posible que la relación observada suceda también en la realidad y cuál es su magnitud. Otro requisito importante, sin embargo, es contraponer nuestra hipótesis con otras alternativas. Y esta es la gran virtud de la regresión múltiple, que es una técnica de análisis multivariante que permite añadir nuevas variables independientes al modelo de regresión. Esta técnica nos ayuda a considerar el efecto que pueden tener otras variables, como *z* o *w*, en nuestra afirmación de que *x* tiene un efecto sobre *y*. Así, la regresión múltiple nos permite hacer afirmaciones más robustas sobre la relación entre *x* e *y*, porque podemos comprobar si *x* e *y* tienen relación incluso cuando controlamos los efectos de *x* para otras variables independientes como podrían ser *z* o *w*. Esta lógica la explicamos a continuación y acabaremos la sección con un ejemplo aplicado que relaciona el tipo de régimen y el voto afirmativo a la ONU.

5.1. Cuantificar

Si sabemos cómo se construye un modelo de regresión simple, la regresión múltiple es muy sencilla, puesto que solo hay que añadir las variables adicionales con el signo + a la fórmula `lm()`. En el siguiente código hemos añadido la variable categórica binaria *z* a la relación y hemos pedido directamente un sumario. Hemos excluido de la impresión la parte de residuos.

```
> summary(lm(formula = y ~ x + z, data = model))
Coefficients:
               Estimate   Std. Error    t value        Pr(>|t|)
(Intercept)     0.94829      0.35723      2.655        0.0173 *
x               0.47556      0.05343      8.901    1.35e-07 ***
z             -0.20550    0.29111      -0.706      0.4904
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6335 on 16 degrees of freedom
Multiple R-squared: 0.8326, Adjusted R-squared: 0.8116
F-statistic: 39.78 on 2 and 16 DF, p-value: 6.173e-07
```

La lectura de este sumario, donde encontramos incluida la nueva variable *z*, es muy parecida a la lectura en el análisis con dos variables. En primer lugar, la constante o *intercept* muestra qué valor toma *y* cuando el valor de todo el resto de variables explicativas es 0. Es decir, cuando *x* toma el valor 0 y *z* toma el valor 0, el valor de *y* será de 0,94. En segundo lugar, el coeficiente de la variable numérica *x* nos indica que, por cada unidad que aumentamos de *x*, la *y* aumentará 0,48 y dejará constante el

resto de variables independientes. Esta relación es estadísticamente significativa. En tercer lugar, el coeficiente de la variable categórica *z* nos explica el efecto que tiene sobre *y* aumentar los valores de *z* de 0 a 1 dejando constante el resto de variables independientes. Esto quiere decir que cuando *z* pasa de 0 a 1, disminuye 0,20 en nuestro modelo. La relación es negativa y no es estadísticamente significativa, de modo que no podemos asegurar que esta relación de *z* sobre *y* exista si la generalizamos en una población más amplia.

En último lugar, en análisis multivariante nos fijaremos en el R^2 ajustado. Vemos cómo un 81 % de la variabilidad de *y* puede ser explicada por las dos independientes del modelo. Si comparamos estos resultados con el ejercicio anterior, vemos que la introducción de *z* en nuestro modelo no nos ha aportado más capacidad de explicar *y*. No es estadísticamente significativa ni tampoco nos ha hecho aumentar de manera sustancial el coeficiente de determinación, más bien al contrario. Por eso mismo, valdrá la pena descartar *z* de nuestro modelo. Antes, sin embargo, veremos cómo quedaría la fórmula si decidiéramos incluir la variable.

```
y = 0.94829 + 0.47556 * x -0.20550 * z
```

Es importante entender que la regresión múltiple tiene que servir esencialmente como un método que permite rechazar con más solidez la hipótesis nula. Es decir, este método nos permite decir que el efecto observado de la variable independiente principal sobre la variable dependiente se continúa produciendo con un intervalo de confianza superior al 95 % incluso cuando controlamos por el efecto otras variables explicativas rivales.

5.2. Caso de estudio

En los apartados anteriores hemos podido observar, en contra de nuestras expectativas, que justamente son las no democracias las que votan más afirmativamente en la ONU que las democracias. La figura 22 nos daba una pista sobre esta tendencia. En este apartado testaremos la hipótesis de que las no democracias tienden a votar más favorablemente las resoluciones de la AGNU en relación con las democracias. Por eso, hemos diseñado un análisis de regresión múltiple donde contrastamos nuestra hipótesis con otras hipótesis rivales, como encontraréis explicado en la tabla 14. La hipótesis que queremos comprobar es que ser una democracia (*x*) tiene un efecto negativo sobre el voto afirmativo en la AGNU (*y*). Pensamos, sin embargo, que quizás esto está determinado por el nivel de desarrollo del país (*w*). Las democracias acostumbran a ser ricas y las autocracias acostumbran a ser pobres, por lo que el hecho de ser rico podría ser el facotr que haga votar de manera menos favorable a la AGNU y no el hecho de ser una democracia. Así, en el análisis multivariante controlaremos los efectos de nuestra hipótesis principal por los efectos de la primera hipótesis alternativa, el nivel de desarrollo, medido por el PIB per cápita. Creemos que un PIB per cápita mayor afectará negativamente al voto afirmativo en la AGNU.

Como segunda hipótesis alternativa, hemos considerado que el voto afirmativo en la AGNU (*y*) puede estar condicionado por la proporción de democracias en la asamblea (*z*). Así, los años que haya más democracias en la ONU el voto afirmativo será más desfavorable, y los años que haya menos el voto afirmativo será más favorable. Por lo tanto, creemos que la proporción anual más elevada de democracias en la AGNU tendrá una incidencia negativa en el voto afirmativo de las democracias.

Tabla 14. Análisis multivariante del voto en la AGNU

Variable	Medida y fuente	Codificación	Hipótesis
Variable dependiente (*y*)	Proporción de voto afirmativo en la AGNU (*unvotes*)	Cercano a 1, más voto afirmativo; cercano a cero, menos voto afirmativo (*vote*)	
Variable independiente principal (*x*)	Clasificación dicotómica de democracia (DD)	1 = Democracia, 0 = No democracia (*democracy*)	Más democrático, menos voto afirmativo
Variable independiente alternativa (*w*)	Desarrollo del país (*wbstats*)	PIB per cápita PPP, precios constantes 2011 (*gdpcap*)	Más PIB per cápita, menos voto afirmativo
Variable independiente alternativa (*z*)	Proporción de democracias en la AGNU (*unvotes*)	Cercano a 1, más proporción de democracias; cercano a 0, menos proporción (*unga_prop)*	Más proporción de democracias, menos voto afirmativo

Fuente: elaboración propia.

Para poder hacer este análisis multivariante, hemos creado el marco de datos `un_wdi`, disponible en el anexo, con las variables *vote*, *democracy*, *gdpcap* y *unga_prop*.[3] A continuación, hemos pedido el sumario del modelo lineal. Podemos ver que *democracy* tiene una relación negativa y estadísticamente significativa con la variable dependiente cuando controlamos los efectos de las otras variables del modelo. Esto permite decir que hay relación con un 95 % de confianza y nos permite descartar la hipótesis nula. También vemos que el *gdpcap* tiene una relación negativa y estadísticamente significativa con la variable dependiente, controlando los efectos

3. Para cualquier duda, consultad la sección de cursos en la web del autor (http://www.jordimas.cat/courses).

de las otras variables. La variable *unga_prop* también tiene una relación negativa, tal como esperábamos, pero en este caso no tiene una relación estadísticamente significativa con *y*.

```
> summary(lm(formula = vote ~ democracy + gdpcap + unga_prop,
data = un_wdi))
Call:
lm(formula = vote ~ democracy + gdpcap + unga_prop, data =
un_wdi)

Residuales:
    Min           1Q       Median         3Q       Max
-0.91732      -0.07830     0.01938      0.10733     0.32181

Coefficients:
              Estimate     Std. Error   t value     Pr(>|t|)
(Intercept)   9.805e-01    5.026e-02    19.509      <2e-16 ***
democracy     -1.042e-01   5.226e-03    -19.946     <2e-16 ***
gdpcap        -2.247e-06   1.442e-07    -15.585     <2e-16 ***
unga_prop     -1.113e-01   8.921e-02    -1.248      0.212
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1441 on 3171 degrees of freedom
Multiple R-squared: 0.1903, Adjusted R-squared:  0.1895
F-statistic: 248.4 on 3 and 3171 DF,  p-value: < 2.2e-16
```

Lo más importante de este análisis es que nuestra hipótesis continúa siendo cierta con un nivel suficiente de confianza incluso cuando la contrastamos con dos hipótesis rivales. Por lo tanto, la afirmación de que las no democracias tienden a votar más favo-

rablemente las resoluciones de la AGNU que las democracias es más robusta después de este análisis multivariante. También podemos decir, observando el R^2, que nuestro modelo explica cerca de un 19% de la variabilidad de la variable dependiente. En análisis multivariante miraremos el R^2 ajustado. También es muy importante saber leer la columna *Estimate*, que nos dice el efecto de cada variable independiente sobre la dependiente. En el siguiente código traducimos esta columna a la fórmula matemática que nos permite tener un modelo predictivo.

```
y = 0.98 - 0.10*democracy - 0.000002*gdpcap – 0.11*unga_prop
```

Según nuestro modelo, en un hipotético país con valor cero en la variable democracia (por lo tanto, que sea una no democracia), con el PIB per cápita de valor cero (por lo tanto, absolutamente pobre) y en una situación con cero democracias en la AGNU, este hipotético país votará afirmativo un 98% de las veces. En cambio, si cambiamos el régimen del país a democrático (la variable *x* pasa de 0 a 1) y dejamos las otras variables constantes, este hipotético país votará a favor un 88% de las veces, puesto que en nuestro modelo hay un cambio del 10% en el voto según el tipo de régimen. En cuanto al nivel de desarrollo, a medida que el PIB per cápita se incrementa, la proporción de voto afirmativo disminuye. Concretamente, la proporción de voto afirmativo disminuye un 2% por cada 10.000 dólares adicionales per cápita que tenga el país. Finalmente, cuantas más democracias haya en la AGNU, menos votará a favor el país. Traducido en números: de una hipotética situación de una AGNU con solo democracias a otra con solo no democracias, el voto hipotético disminuye un 10%. Esta diferencia, sin embargo, no es estadísticamente significativa.

Ejercicios de autoevaluación

Para un mejor aprendizaje, intentad hacer mentalmente el máximo de ejercicios posible, sin utilizar R.

1) Deducid la técnica que utilizaremos según estas dos variables:

```
database$black_white, database$blue_red
```

2) Deducid la técnica que utilizaremos según estas dos variables:

```
database$gdp, database$unemployment
```

3) Pedid la tabla de contingencia de estas variables:

```
wdi$class, wdi$adult
```

4) Pedid la tabla de contingencia con las proporciones en las filas para estas variables.

```
wdi$gender, wdi$vote
```

5) Cambiad el siguiente código para visualizar las proporciones de la tabla de contingencia.

```
ggplot(wdi, aes(country, democracy)) + geom_bar()
```

6) ¿Cuál de los resultados siguientes no puede devolver un test de Cramer?

```
0.45, 0.15, 1.10
```

7) Hemos pedido el test de Chi cuadrado de Pearson. ¿Cuál de los p-valores siguientes es estadísticamente significativo?

```
0.01, 0.96, 0.06
```

8) Pedid la diferencia de medias entre estas dos variables:

```
wdi$country, wdi$vote
```

9) Cambiad los atributos de la geometría para pedir una recta LOESS.

```
geom_smooth()
```

10) Indicad cuál de las siguientes correlaciones es la más fuerte:

```
-0.921, 0.826, 0.656
```

11) ¿Qué R^2 tendremos con el siguiente coeficiente de correlación?

```
1
```

12) Decid cuál es el valor de *y* cuando la *x* es cero en esta ecuación:

```
y = 450 + x * -0.3
```

13) ¿Cuál es el efecto en *y* por cada unidad que aumentamos de *x*?

```
y = 450 + x * -0.3
```

14) Indicad los asteriscos que veremos con el siguiente p-valor:

```
0.002
```

15) Pedid el análisis multivariante de las siguientes variables:

```
mdata$conflict / mdata$ethnic, mdata$gdp, mdata$mspend
```

Solucionario

1) `Tabla de contingencia`

2) `Regresión lineal / correlación`

3) `table(wdi$class, wdi$adult)`

4) `prop.table(table(wdi$gender, wdi$vote), 1)`

5) `ggplot(wdi, aes(country, democracy)) + geom_bar(position = "fill")`

6) `1.10`

7) `0.01`

8) `t.test(wdi$country, wdi$vote)`

9) `geom_smooth()` (no se tiene que cambiar nada)

10) `-0.921`

11) `1`

12) `450`

13) `-0.3`

14) `**`

15) `lm(formula = conflict ~ ethnic + gdp + mspend, data = mdata)`

Glosario

`chisq.test()` Pide la prueba del Chi cuadrado de Pearson.

`colSums()` Suma las columnas de una tabla de contingencia.

`cor()` Devuelve el coeficiente de correlación de dos variables.

`CramerV()` Pide el test de Cramer V para tablas de contingencia (*DescTools*).

`t.test()` Pide el t-test para diferencia de medias.

`lm()` Devuelve el modelo de regresión lineal simple o múltiple, según el número de variables que utilizamos.

`prop.table()` Muestra las proporciones de una tabla de contingencia.

`rowSums()` Suma las filas de una tabla de contingencia.

`table()` Muestra una tabla de contingencia de dos variables categóricas.

Bibliografía

Agresti, A.; Finlay, B. (2009). *Statistical Methods for the Social Sciences*. Nueva Jersey: Pearson.

Babbie, E. R. (2013). *The Practice of Social Research*. Wadsworth: Cengage Learning.

Braumoeller, B. F.; Sartori, A. E. (2004). «The promise and perils of statistics in international relations». En: D. F. Sprinz; Y. Wolinsky-Nahmias, (eds.). *Models, Numbers, and Cases: Methods for Studying International Relations*. Ann Arbor, MI: University of Michigan Press (págs. 129-151).

Cheibub, J. A.; Gandhi, J.; Vreeland, J. R. (2010). «Democracy and Dictatorship Revisited». *Public Choice* (vol. 143, n.º 2-1, págs. 67-101).

Halperin, S.; Heath, O. (2016). *Political Research: Methods and Practical Skills*. Oxford: Oxford University Press.

Johnson, J. B.; Reynolds, H.; Mycoff, J. (2007). *Political Science Research Methods*. Washington, DC: CQ Press.

King, G.; Keohane, R. O.; Verba, S. (1994). *Designing Social Inquiry: Scientific Inference in Qualitative Research*. Princeton: Princeton University Press.

Mansfield, E. D.; Pevehouse, J. C. (2008). «Quantitative Approaches». En: C. Reus-Smit; D. Snida (eds.). *The Oxford Handbook of International Relations*. Oxford: Oxford University Press (págs. 481-498).

Sprinz, D. F.; Wolinsky-Nahmias, Y. (2004). *Models, Numbers, and Cases: Methods for Studying International Relations*. Ann Arbor, MI: University of Michigan Press.

Voeten, E. (2012, 17 de julio). «Data and Analyses of Voting in the UN General Assembly». SSRN [Fecha de consulta: 16 de julio de 2020] <https://ssrn.com/abstract=2111149>

Anexo

Código del apartado 3

```
library(readxl)
library(countrycode)
download.file(
"https://uofi.box.com/shared/static/d27425539c9d662a7041.xls",
"dd.xls")

DDdata <- read_xls("dd.xls") %>%
   filter(year > 1945) %>%
   mutate(iso2c = countrycode(uncode, "un", "iso2c")) %>%
   select(iso2c, country = ctryname, year, democracy,
          un_continent_name)
```

Código de la figura 21

```
un_dd_year %>%
   filter(between(year, 1995, 2000)) %>%
   ggplot(aes(x = democracy, y = mean, col = continent)) +
   geom_jitter(alpha = 0.5) +
   stat_summary(geom = "point", fun.data = mean_se,
                col= "red", size = 1.2) +
   stat_summary(geom = "errorbar", fun.data = mean_se,
                col = "red", width = 0.1) +
theme_classic()
```

Código de la figura 22

```
un_dd_year %>%
   filter(between(year, 1946, 2000)) %>%
   ggplot(aes(x = factor(year), y = mean, col = democracy)) +
   geom_jitter(alpha = 0.2) +
   stat_summary(fun.data = mean_se, col = "red", size = 0.2) +
   scale_y_continuous(limits = c(0.3, 1)) +
   scale_x_discrete(name = "Year",
                    breaks = c(1946, seq(1950, 2000, 5))) +
   theme(legend.position = c(0.9, 0.2),
         legend.background =
         element_rect(fill=alpha("light blue", 0.5)))
```

Código de la figura 23

```
q <- seq(1,100,1)
w <- (50 - q)^2
s <- sample(100)

visual_corr <- data.frame(graph1_x = q,graph1_y = s,
             graph2_x = -jitter(seq(1,100,1), 100),
             graph2_y = jitter(seq(1,100,1), 100),
             graph3_x = abs(c(jitter(seq(1,99,1), 50), 5)),
             graph3_y = jitter(seq(1,100,1), 50),
             graph4_x = jitter(q, 20), graph4_y = jitter(w, 1500),
             graph5_x = abs(jitter(q, 10)),
             graph5_y = abs(jitter(log(q), 10)),
             graph6_x = jitter(q, 10),
             graph6_y = jitter(q, 10))
```

```
visual_corr1 <- visual_corr %>%
   gather("graph", "value") %>%
   separate(graph, c("tabla", "codigo"), sep = "_") %>%
   group_by_at(vars(-value)) %>%
   mutate(row_id=1:n()) %>% ungroup() %>%
   spread(key=codigo, value=value) %>%
   select(-row_id)

visual_corr1 %>%
   ggplot(aes(x, y)) +
      geom_point() +
      facet_wrap(~ tabla, scales = "free")
```

Código del apartado 4.1

```
gpm7 <- gapminder %>%
   filter(year == 2007, between(lifeExp, 47, 60), gdpPercap >
   9000)

gapminder %>% filter(year == 2007) %>% ggplot(aes(gdpPercap,
lifeExp)) + geom_point() + geom_point(data = gpm7, aes(gdpPercap,
lifeExp), col = "red") + theme_classic()
```

Código de la figura 26

```
R2 <- data.frame(x_99 = abs(jitter(seq(1,100,1), 15)), y_99
           = abs(jitter(seq(1,100,1), 15)),
           x_90 = abs(jitter(seq(1,100,1), 50)),
           y_90 = abs(jitter(seq(1,100,1), 50)),
           x_15 = abs(jitter(seq(1,100,1), 200)),
           y_15 = abs(jitter(seq(1,100,1), 200)))


R2x <- R2 %>%
   gather("graph", "value") %>%
   separate(graph, c("codigo", "R2"), sep = "_") %>%
   group_by_at(vars(-value)) %>%
   mutate(row_id=1:n()) %>% ungroup() %>%
   spread(key=codigo, value=value) %>%
   select(-row_id) %>%
   arrange(desc(R2))


R2x %>%
   ggplot(aes(x, y)) +
   geom_point(size = 1) +
   facet_wrap(~ factor(R2, levels = c("99", "90", "15")), sca-
   les = "free")
```

Código de la figura 27

```
model <- data.frame(x = c(0.96, 1.51, 1.84, 2.87, 3.10, 3.96,
4.22, 4.31, 5.03, 5.60, 6.06, 6.46, 7.36, 7.98, 8.18, 8.01,
8.88, 9.84, 9.73), y = c(1.44, 2.00, 2.51, 1.10, 1.64, 2.67,
3.26, 3.44, 2.95, 2.41, 4.16, 3.96, 3.64, 4.68, 5.81, 5.24,
4.84, 5.22, 5.56))

ggplot(model, aes(x, y)) + geom_point(size = 1) +
  geom_smooth(method = "lm", col = "red",
              fullrange = TRUE, se = FALSE) +
  scale_x_continuous(breaks = c(seq(1,10))) +
  scale_y_continuous(breaks = c(seq(1,6)))
```

Código del apartado 5.2

```
library(wbstats)
wbsearch("gdp per capita.*constant")

gdp_wdi <- wb(indicator = "NY.GDP.PCAP.PP.KD",
              startdate = 1946, enddate = 2018)
gdp_wdi <- transmute(gdp_wdi, iso2c,
                     year = as.numeric(date), country,
                     NY.GDP.PCAP.PP.KD = value)

un_year_vote <- un_roll_calls %>% separate(date,
   "year", extra = "drop") %>%
   inner_join(un_votes) %>%
   mutate(year = as.numeric(year),
```

```
       vote = if_else(vote == "yes", 1, 0)) %>%
    group_by(country, country_code, year) %>%
    summarize(vote = mean(vote))

un_vote_dem <- un_year_vote %>%
    inner_join(DDdata, by = c("country_code" = "iso2c",
    "year" = "year")) %>%
    select(country = country.x, country_code, year, vote,
    democracy)

un_prop <- un_vote_dem %>%
    group_by(year) %>%
    summarize(unga_prop = mean(democracy))

un_wdi <- un_vote_dem %>%
    left_join(un_prop, by = "year") %>%
    inner_join(gdp_wdi, by = c("country_code" = "iso2c",
    "year" = "year")) %>%
    select(country = country.x, year, vote, democracy,
    gdpcap = NY.GDP.PCAP.PP.KD, unga_prop) %>%
    filter(gdpcap != is.na(gdpcap))

summary(lm(formula = vote ~ democracy + gdpcap + unga_prop,
data = un_wdi))
```

Capítulo V
Importar, limpiar, unir

1. Introducción

No siempre un analista de datos encuentra la información a punto para ser explorada. El trabajo principal del analista es, a menudo, mucho más pesado de lo que parece, puesto que los datos pueden estar desordenados, con nombres mal escritos difíciles de interpretar o separados en diferentes archivos de internet. Por esta razón, en algunos ámbitos, los científicos de datos tienen que dedicar la mayor parte de su tiempo a estas tareas (Lohr, 2014). El objetivo de este capítulo es ayudaros a convertir datos dispersos y desconectados entre sí en un solo marco de datos preparado para ser analizado.

En ciencias sociales, y más particularmente cuando se realiza investigación sobre temas de índole internacional, los datos provienen en la mayoría de ocasiones de las principales organizaciones y centros de estudios internacionales, de modo que ya han sido sistematizados previamente. Estas entidades se encargan de recoger, limpiar y ordenar los datos para presentarlos al público. Incluso algunos datos tienen disponible una librería específica en R para facilitarnos el tratamiento. Esto no resta importancia al hecho de aprender a adquirir y a preparar los datos para poderlos estudiar a fondo.

En este capítulo os enseñaremos tres pasos básicos para preparar los datos:

1) Importar los datos a RStudio. En este primer apartado os enseñaremos diferentes maneras de incorporar los datos en el programa.

2) Una vez incorporados, hay que saber limpiar los datos. Dedicaremos el segundo apartado del módulo a esta tarea, que consiste en modificar cuando sea necesario la estructura del marco de datos y el tipo de variables que tenemos, y a tomar decisiones sobre los valores perdidos y los casos extremos.

3) Finalmente, hay que unir diferentes marcos de datos. Este proceso es sumamente útil para nuestro trabajo, puesto que nos permite analizar información que proviene de fuentes diferentes. Por lo tanto, el proceso de unir bases de datos será un paso esencial previo al tratamiento.

2. Importar datos

Importar contenido es el primer paso que tenemos que hacer para trabajar con datos en RStudio. En internet tenemos multitud de bases de datos relacionadas con estudios internacionales, como World Bank, Eurostat, Correlates of War, etc.[1] En este apartado aprenderemos a importar a R algunas de ellas. Tenemos dos vías principales para importarlas: mediante un paquete de R, o bien importando un archivo localizado en nuestro ordenador o en la red.

1. Una de las listas más extensas de bases de datos en estudios internacionales que encontramos en la red la tenemos en este GitHub: https://github.com/erikgahner/PolData.

2.1. Importar un paquete

Dentro de los paquetes que ya tenemos actualmente cargados en R, ya hay una gran cantidad de bases de datos que nos pueden servir para practicar con el programa. Uno de los paquetes de base de R denominado *datasets* contiene cerca de un centenar de marcos de datos que podéis consultar tecleando `help = "datasets"`. Otros paquetes que ya conocemos, como dplyr o ggplot2, incorporan también, entre muchos objetos y funciones, algunos marcos de datos en el interior. Hay otros paquetes que están alojados en la biblioteca central de R, el CRAN, como es el caso de gapminder, que son fáciles de utilizar porque, cuando cargamos el paquete en R, se nos cargan también los marcos de datos de su interior. Para instalar un paquete, utilizaremos la función `install.packages()` con el nombre del paquete entre comillas y para cargarlo en R utilizaremos `library()` con el nombre del paquete sin las comillas. Una vez cargado, podemos ver las funciones y los objetos del paquete si introducimos su nombre seguido de dos puntos (por ejemplo, `gapminder::`). Nos aparecerá un desplegable con la información.

Marcos de datos en dplyr y ggplot2

El paquete dplyr, por ejemplo, tiene el marco de datos `starwars`. El paquete ggplot2 incorpora `presidential` (mandatos de los presidentes de Estados Unidos desde Eisenhower). Si queréis saber más detalles, poned el nombre en la consola con un interrogante delante (ejemplo: `?Titanic`).

Sin embargo, la mayoría de paquetes que tienen relación con los estudios internacionales son algo más sofisticados, puesto

que incorporan varias funciones que tenemos que utilizar para poder trabajar con los marcos de datos. Uno de los más utilizados es el paquete countrycode, que incorpora tres marcos de datos y la función `countrycode()`. Los marcos de datos contienen los nombres de los países en varios idiomas, así como los códigos que toman estos países según diferentes bases de datos. Ello nos sirve para adaptar fácilmente nuestros marcos de datos. Por ejemplo, podríamos pasar al español los nombres de los países de un marco de datos originalmente en inglés. También podríamos transformar rápidamente los códigos de país de la nomenclatura que usa el Banco Mundial a la que usa Correlates of War. Estas operaciones se realizan normalmente a través de la función `countrycode()`.

Un paquete fácil de utilizar es unvotes, que consta de tres marcos de datos y tiene registradas todas las votaciones históricas en la Asamblea General de las Naciones Unidas. El marco de datos principal del paquete es `un_votes`, que en la variable *rcid* contiene un registro del número de votación y en la variable *vote,* el sentido del voto de cada país: sí, no o abstención. En `un_roll_calls` tenéis la información más ampliada de cada votación, mientras que `un_roll_call_issues` nos ayuda a buscar por temática de cada votación.

El paquete WB Stats contiene centenares de indicadores que se encuentran alojados en la base de datos del Banco Mundial. Una vez que hayamos cargado el paquete, podremos barajar los indicadores con el motor de búsqueda `wbsearch()`, donde introduciremos la palabra clave que nos interesa. Si queremos buscar por más de una palabra clave, las separaremos con `.*`. En el ejemplo que veremos a continuación, la búsqueda por *military* nos devuelve cinco entradas. Hemos hecho una segunda búsqueda con *military* y *gdp*, que nos ha devuelto solo una entra-

da. En la consola veremos cómo R nos devuelve un marco de datos de dos columnas, el primero con el código de la variable y el segundo con su descripción. En la búsqueda que acabamos de hacer, el código de la variable es *MS.MIL.XPND.GD.ZS* y la descripción *Military expenditure (% of GDP)*. Para importar esta información, utilizaremos la función `wb()`, donde indicaremos el código de la variable y, opcionalmente, un vector con los países y los años que queremos que nos filtre.

```
wbsearch("military")
wbsearch("military.*gdp")
mil_exp <- wb(indicator = "MS.MIL.XPND.GD.ZS",
              country = c("GB"), startdate = 1970,
              enddate = 2005)
str(mil_exp)
names(mil_exp)[3] <- "mil_exp_uk"
plot(mil_exp$mil_exp_uk)
```

En el ejemplo, hemos creado el objeto `mil_exp` con los datos del gasto militar del Reino Unido entre los años 1970 y 2005. Después de comprobar que la tercera columna se refiere a los valores con una visualización de la estructura, hemos cambiado el nombre de la columna por otro más fácil de recordar y, a continuación, hemos visualizado los datos con un diagrama de dispersión.

El paquete eurostat funciona de una manera parecida, también con un motor de búsqueda `search_eurostat()`. Hay que estar atento a mayúsculas y minúsculas porque este motor es sensible a ellas. La búsqueda es algo menos intuitiva, por lo cual si lo preferís podéis buscar las variables que os interesen en la web de Eurostat. En el ejemplo siguiente hemos buscado

la palabra *pollution* y hemos obtenido dos entradas. A continuación, hemos seleccionado solo la primera columna de los datos para ver más claramente la descripción. Hemos comprobado que la segunda entrada *Environmental protection expenditure...* era la que más nos interesaba y hemos pedido la segunda fila de la segunda columna, que nos ha devuelto el código *env_ac_exp3*. A continuación, hemos importado la variable de interés con la función `get_eurostat()`y hemos denominado `pollution` al marco de datos resultante. Finalmente, hemos pedido un sumario.

```
search_eurostat("pollution")
search_eurostat("pollution")[, 1]
search_eurostat("pollution")[2, 2]
pollution <- get_eurostat("env_ac_exp3", time_format = "num")
summary(pollution$values)
```

Hay más bases de datos que tienen paquetes en R y funcionan de manera parecida a los descritos anteriormente: o bien podemos acceder directamente al marco de datos, o bien incorporan un motor de búsqueda con la función `search` y una función `get` para importar los datos. En la tabla 15 mostramos una selección de varios paquetes relacionados con los estudios internacionales.

Tabla 15. Paquetes de R de bases de datos

Base de datos	Código	Descripción
Banco Mundial	`wbstats`	Paquete de datos del Banco Mundial
Ciencia política	`psData`	Recopilación de bases de datos de Ciencia Política
Comercio OEC	`oec`	Paquete del Observatory of Trade Complexity (OEC)
Comercio UN	`comtradr`	Datos de comercio de las Naciones Unidas
Eurostat	`eurostat`	Paquete de la base de datos Eurostat
Estados	`states`	Datos de estados independientes en el mundo desde 1816
FMI (1)	`IMFData`	Paquete del Fondo Monetario Internacional (FMI)
FMI (2)	`imfr`	Alternativa para acceder al FMI
OCDE	`OECD`	Paquete para extraer datos de la OCDE
Votos en la ONU	`unvotes`	Votos de cada país en la Asamblea General de la ONU
WDI	`WDI`	Indicadores de desarrollo del Banco Mundial

Fuente: elaboración propia.

2.2. Importar un archivo

Para importar un archivo, lo podemos hacer manualmente por medio de File -> Import Dataset o mediante el código. En este apartado explicamos cómo crear el código para importar los archivos. Lo primero que tenemos que saber es que las bases de datos pueden estar almacenadas en diferentes tipos de archivo. Los más comunes los describimos en la tabla 16:

Tabla 16. Tipos de archivo para almacenar bases de datos

Tipo	Extensión	Paquetes y funciones
CSV	`.csv`	`utils::read.csv(), utils::read.csv2(), utils::read.table()`
Excel	`.xls / .xlsx`	`gdata::read.xls(),readxl::read_excel()`
R	`.Rdata`	`base::readRSD()`
SPSS	`.sav`	`haven::read_spss, foreign_read.spss()`
STATA	`.dta`	`haven::read_stata(), haven::read_data(), foreign::read.dta()`
TXT	`.txt, .tab, .tsv`	`utils::read.delim(), utils::read.delim2(), utils::read.table()`

En la columna de paquetes y funciones, hemos puesto el nombre del paquete que corresponde a cada función.
Fuente: elaboración propia.

Descarga desde el directorio de trabajo

Podemos saber dónde tenemos ubicado el directorio de trabajo con la función `getwd()`, mientras que con `dir()` veremos qué archivos tenemos guardados. Con `setwd()` o manualmente en la pestaña *Files,* podremos cambiar la ubicación del directorio de trabajo.

Lo segundo que tenemos que saber es que el primer argumento que introduciremos en todas estas funciones es la localización del archivo que contiene la base de datos. El archivo puede estar localizado en nuestro ordenador o en la red. Si está localizado en nuestro ordenador, recomendamos tenerlo ubicado en la carpeta que hemos indicado a R como nuestro directorio de trabajo y aplicar una de las funciones de la tabla anterior. Si, por el contrario, el archivo está localizado en la red, podemos actuar de varias maneras:

1) Descargar el archivo manualmente, ponerlo en nuestro directorio de trabajo y abrirlo con una de las funciones de la tabla anterior como si estuviera localizado en nuestro ordenador.

2) Por medio de la función `download.file()`. Con esta función indicamos la dirección web en el primer argumento y el nombre del archivo que crearemos en el segundo argumento. Esta operación nos guardará el archivo en el directorio de trabajo y después lo podremos importar como en los procedimientos que acabamos de explicar.

3) Importarlo directamente de internet con una de las funciones indicadas en la tabla 16 usando la dirección web como primer argumento. La opción elegida dependerá de vuestras preferencias y, a veces, de cómo esté ubicado el archivo en la red.[2]

2.2.1. Archivos planos (.csv, .txt, .tab, .tsv)

Entendemos por *archivos planos* los archivos de texto que contienen los valores separados por algún carácter no alfabético, como un espacio, un guion, un punto y coma o un punto. En este apartado distinguiremos dos tipos:

- los archivos Comma Separated Values (CSV), y
- los Tab-Separated Values (utilizaremos TXT como abreviación).

Estos archivos ocupan muy poco espacio de memoria de manera que el proceso para importarlos o guardarlos suele ser

2. A veces, encontraremos bancos de datos en los que solo podemos descargar una base de datos si estamos subscritos a ellos, por lo que no podremos descargar el archivo directamente.

muy rápido. Normalmente, los CSV están separados por comas, mientras que los TXT acostumbran a estar separados por tabulaciones o espacios. Si abrimos un CSV, tendría más o menos el aspecto siguiente:

```
pais,pibcap,habitantes,deuda,exportaciones,importaciones
Alemania,44469,82.79,64.1,1.33,1.08
Austria,47290,8.77,78.4,2.45,3.44
Bélgica,43323,11.35,103.1,4.66,5.32
```

Podemos ver que, en cada fila, la separación entre valores está marcada por la coma. El punto sirve para marcar los decimales, mientras que, en el caso del ejemplo, podemos observar que la primera fila representa los títulos de las columnas. Para importar estos tipos de archivo a R, tenemos diferentes paquetes. En esta asignatura enseñamos el más básico, que se encuentra en el paquete *utils*, que viene de serie con R.[3] La función para importar archivos CSV `read.csv()` lleva asociados por defecto los siguientes argumentos:

```
read.csv("archivo.csv", header = TRUE, sep = ",", dec = ".",
stringsAsFactors = TRUE)
```

En primer lugar, pondremos el nombre del archivo que habremos situado antes en el directorio de trabajo o la dirección web donde se encuentra. Es posible que con el primer argumento ya baste para importar la tabla y no tengamos que especificar ninguno más. Pero suele pasar que la primera fila

3. En el CRAN, podemos encontrar otros paquetes que sirven para importar archivos planos como *readr* o *data.table*.

no lleve el título de las columnas, de manera que tendremos que especificar `header = FALSE`. Si es así, dentro de la función podemos especificar un nuevo argumento `col.names`, donde indicaremos el título de las columnas (por ejemplo, `col.names = c("pais", "pibcap", "deuda", "exportaciones", "importaciones")`). El vector tendrá que tener tantos valores como columnas tiene el marco de datos.[4] En los archivos CSV, por defecto el carácter que separará los archivos es una coma, pero, si es otro carácter, lo tendremos que especificar. Por ejemplo, `sep = " "` indicará que el separador es un espacio. Los separadores de los archivos planos tienen mucho que ver con los sistemas de almacenar bases de datos. El sistema americano utiliza `read.csv()` y separa los decimales con el punto y los valores con la coma, mientras que el sistema europeo utiliza `read.csv2()` y separa los decimales con una coma y los valores con un punto y coma. Finalmente, todos los valores que nos detecte como vectores de carácter se convertirán en factores. Si queremos cambiarlo, podemos poner el argumento en `stringsAsFactors = FALSE`.

Para importar archivos TXT utilizaremos `read.delim()`, que hace exactamente lo mismo que la función que acabamos de ver, pero tiene por defecto el separador `sep = "\t"` (que significa tabulación). La función `read.delim2()` nos importará los archivos en sistema europeo. La función `read.table()` puede leer tanto CSV como TXT, pero normalmente le tendremos que especificar más argumentos. Su separador por defecto es `sep = "/"`.

4. Del mismo modo, R nos creará el tipo de vector asociado a cada variable por defecto, pero lo podemos especificar manualmente con el argumento `colClasses` (por ejemplo, `colClasses = c("character", "factor", "NULL", "numeric", "logical")`). R no nos importará la columna indicada como NULL.

2.2.2. Excel

Excel es un programa muy útil para trabajar con hojas de cálculo y bases de datos de pequeñas dimensiones. A menudo, algunas bases de datos se guardan en este formato por lo cual necesitaremos saber cómo importar archivos de Excel a R. En primer lugar, tenemos que tener en cuenta dos consideraciones importantes: los datos acostumbran a estar repartidos en varias hojas (sabréis que accedemos a ellos desde pestañas visualizables en la parte inferior de las hojas de cálculo) y muchas veces los datos no empiezan en la primera fila ni en la primera columna de la hoja de cálculo. Es por eso por lo que quizá valga la pena echar un vistazo previo al documento en Excel para hacernos una idea de cómo están distribuidos los datos de interés. Los archivos de Excel suelen tener las extensiones *.xls* o *.xlsx*. Para importarlos a R utilizaremos los paquetes *readxl* o *gdata*.

Dentro del paquete *readxl*, necesitaremos la función `read_excel()` para importar los datos. Si solo introducimos dentro de la función el nombre del documento, R nos importará la primera hoja. Podemos especificar la hoja que queremos importar con el argumento adicional `sheet`, donde indicamos el número de hoja o el nombre de la hoja entre comillas. El ejemplo siguiente indica a R que tiene que buscar el archivo `"docexcel.xls"` en el directorio de trabajo, importar la primera hoja de cálculo del documento, interpretar la primera fila como el título de las columnas y no eliminar ninguna fila.

```
read_excel("docexcel.xls", sheet = 1, col_names = TRUE, skip = 0)
```

En el argumento `col_names` podemos especificar TRUE si la primera fila representa el nombre de las columnas y FALSE si

no hay una fila con el nombre de las columnas, o bien podemos crear un vector de carácter indicando los nombres de las columnas. Con `skip` indicamos cuántas filas nos tenemos que saltar antes de empezar a importar datos.

Importar varias hojas

Si queremos bajar varias hojas de cálculo, tendremos que crear varios códigos, cada uno importando una hoja de cálculo diferente. Para saber el nombre de cada hoja podemos utilizar la función `excel_sheets()` y especificar el nombre del archivo del que queremos obtener la lista.

Unir varias hojas de cálculo de Excel

Una vez que hemos importado en varios marcos de datos diferentes hojas de cálculo de Excel, tenemos la opción de unirlos, siempre que los marcos de datos tengan el mismo número de filas. Con la función `cbind()`, podemos especificar el nombre de cada uno de los marcos de datos. Si las primeras columnas (por ejemplo, país y año) se repiten en todas las hojas de cálculo, podemos eliminarlas indicando el número de la columna entre corchetes (por ejemplo, `cbind(hoja1, hoja2[-2], hoja3[-1])`).

2.2.3. Stata y SPSS

También hay paquetes pensados para importar bases de datos de *software* parecido a R, como Statistics and Data (Stata) o Statistical Package for Social Sciences (SPSS). El programa Stata se utiliza principalmente para econometría y guarda los archi-

vos con *.dta*, mientras que SPSS se utiliza más bien en Ciencias Sociales y guarda los archivos con *.sav* o *.por*. Para importar estos archivos a R, podemos utilizar dos paquetes: *haven* o *foreign*.

1) El paquete *haven* utiliza `read_stata()` y `read_dta()` para importar archivos STATA de las versiones 8 a la 15, y `read_spss()` para importar archivos SPSS. La complicación principal para R con los archivos provenientes de estos programas estadísticos es que los vectores de carácter están almacenados como numéricos y guardan los caracteres como etiquetas. Con *haven* no es posible convertir los datos directamente a factores, de manera que lo tendremos que hacer manualmente después de la importación. Para ello, utilizaremos la función `as_factor()` de *haven* para transformar el vector en cuestión en factor y, si hace falta, después lo transformaremos en carácter con `as.character()`.

2) Como alternativa a descargar STATA y SPSS, tenemos el paquete *foreign*, que soporta muchos tipos de formatos diferentes aparte de los mencionados. En el caso de STATA utilizaremos `read.dta()` y en el caso de SPSS, `read.spss()`. El principal inconveniente que tiene la función `read.dta()` es que no soporta archivos STATA superiores a la versión 12. Sin embargo, esta función sí que nos permite convertir directamente las etiquetas de las variables en factores con el argumento ya establecido por defecto `convert.factors = TRUE`. La función `read.spss()` también tiene predeterminado que convierta las etiquetas de SPSS en factores mediante el argumento `use.value.labels = TRUE`. Es importante especificar siempre `to.data.frame = TRUE`, puesto que por defecto no nos crea un marco de datos sino otro tipo de objeto.

2.3. Guardar el marco de datos

Finalmente, después de haber hecho las transformaciones pertinentes en un marco de datos, podemos tener la necesidad de guardarlo en un archivo. El programa R tiene un formato propio con la extensión *.RData* y utiliza la función `saveRDS()`. La gran ventaja de guardarlo en este formato es que no perderemos información sobre el tipo de variables, el orden de los factores, etc. A continuación, veremos cómo guardar el marco de datos `md` en formato R, que denominaremos `«docR.xls»`. En segundo lugar, veremos la operación contraria: cómo importar el archivo hacia el marco de datos `md` mediante la función `readRDS()`:

```
saveRDS(md, "docR.xls")
md <- readRDS("docR.xls")
```

También podemos guardar el marco de datos en un formato más versátil que se pueda abrir en otros programas como Excel, SPSS o STATA. Una buena opción es hacerlo en CSV por medio de `write.table()`. En el código siguiente, hemos ilustrado un ejemplo en el que guardamos el marco de datos de nombre `md` en el archivo `"docR.csv"`. En el tercer argumento especificamos que queremos que nos separe los valores por comas, que el punto marque los decimales y que queremos que nos guarde los nombres de las columnas en la primera fila. Por defecto, `row.names` es `TRUE`, de manera que nos guardará una primera columna adicional con el número de cada fila. Es conveniente evitarlo; por lo tanto, pondremos el argumento en `FALSE`.

```
write.table(md, file = "docR.csv", sep = ",", dec = ".",
            col.names = TRUE, row.names = FALSE)
```

Los paquetes que hemos visto, como *readxl*, *haven* o *foreign,* tienen sus propias funciones para exportar los datos a archivos en formato Excel, SPSS o STATA. También es una buena opción importar y exportar archivos con el paquete *readr*, que no hemos visto en este apartado, pero que tendremos que conocer si queremos adquirir un dominio más avanzado en la importación y exportación de marcos de datos.

3. Limpiar bases de datos

Cuando importamos un marco de datos a R, no necesariamente nos llega limpio, con los datos a punto para ser explorados y analizados. Consideramos limpio un marco de datos si cumple cuatro requisitos (Wickham, 2014):

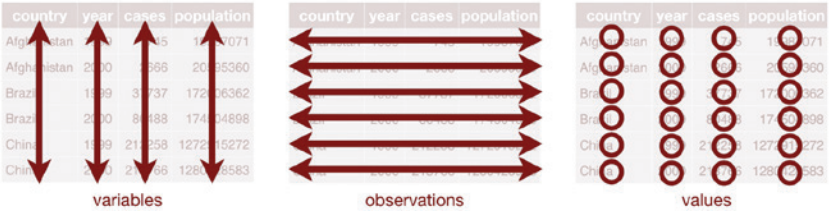
- Cada tipo de unidad observacional tiene que ser un marco de datos diferente.
- Las observaciones tienen que estar representadas en las filas.
- Las variables tienen que estar representadas en las columnas.
- Cada casilla nos tiene que indicar un valor.

El primer requisito implica que solo podemos tener una sola unidad de análisis en cada marco de datos. ¿Qué observa el marco de datos en cuestión? ¿Personas, países, datos de países, regiones? Lo que no tendría sentido es que los datos en un mismo marco se refirieran a veces a personas y a veces a países. En la mayoría de los casos trabajaremos con el país como unidad de análisis.

Los otros tres requisitos para tener un marco de datos limpio se refieren a la estructura de un marco de datos, que resumimos

en la figura 28. Un marco de datos es una colección de valores, cada uno de ellos representado en una casilla diferente. Cada valor pertenece a una variable y observación determinada. Cada variable contiene todos los valores que miden el mismo atributo en todas las unidades. Cada observación contiene todos los valores que miden la misma unidad mediante todos sus atributos. Si la tabla cumple estos requisitos, quiere decir que tenemos una tabla limpia.

Figura 28. Requisitos para una tabla limpia



Fuente: Grolemund, G.; Wickham, H. (2017). *R for Data Science*. O'Reilly. CC BY-NC-ND 3.0 OS.

Lo contrario de una tabla limpia es una tabla sucia. Puede ser que los nombres de las columnas sean valores en lugar de variables, que algunas variables no estén representadas por el tipo de vector que queremos en cada caso, que haya más filas o más columnas de las necesarias o que haya una gran cantidad de datos perdidos o valores extremos que pueden ser errores en los datos. Uno de los síntomas de una tabla sucia (aunque no es un requisito necesario) es que tengamos muchas variables y pocas observaciones.

3.1. Limpiar con tidyr

Si vemos que un marco de datos no cumple los requisitos de una tabla limpia, lo que necesitaremos es el paquete tidyr. Tendremos que asegurarnos que esté instalado y cargado en

nuestro ordenador. Este paquete[5] nos ayuda a transformar un marco de datos principalmente de dos maneras:

- convirtiendo nombres de columna en valores de una nueva variable, o
- convirtiendo valores de una variable en nombres de columnas.

En el primer caso, alargamos el marco de datos, puesto que reducimos el número de columnas y ampliamos el número de filas. Lo haremos con la función `gather()`. En el segundo caso, ensanchamos el marco de datos, puesto que reducimos el número de filas y ampliamos el número de columnas. Lo haremos con la función `spread()`. También aprenderemos dos funciones más de tidyr que van muy ligadas a estas dos: `separate()` y `unite()`. Encontramos resumida la utilidad de las cuatro funciones en la tabla 17.

Tabla 17. Funciones básicas con tidyr

`gather()`	Alarga la tabla transformando varias columnas en valores de una nueva columna.
`spread()`	Ensancha la tabla transformando los valores de una columna en nombres de nuevas columnas.
`separate()`	Separa los valores de una columna en varias columnas.
`unite()`	Une los valores de varias columnas en una columna.

Fuente: elaboración propia.

5. Sobre el paquete tidyr, el científico de datos de RStudio Garrett Grolemund ha preparado un GitHub con una muy buena guía (http://garrettgman.github.io/tidying/) para limpiar marcos de datos.

3.1.1. Las cuatro funciones de tidyr

Empezamos aprendiendo el funcionamiento de `gather()`. Fijaos en el marco de datos siguiente, `tabla_sucia`, que encontraréis disponible en el anexo, en el cual vemos el número de protestas ciudadanas que ha habido en varios países, separadas por el año en el que se han producido en cada columna. ¿Os parece que es una tabla limpia?

```
> tabla_sucia


    pais      X2016    X2017    X2018
1   España    64       83       95
2   Francia   43       34       33
3   Italia    56       53       67
4   México    35       101      120
5   Japón     12       19       18
```

Parece que no. En las columnas vemos indicados los años 2016, 2017 y 2018, que realmente tendrían que ser valores de la variable *año*.[6] Siguiendo los criterios de lo que consideramos una tabla limpia, necesitaríamos eliminar las columnas existentes y crear dos columnas nuevas, la columna *año* y la columna *protesta*. Lo podemos hacer fácilmente con la función `gather()`, donde primero indicaremos el marco de datos que queremos transformar, seguido por los nombres de las dos nuevas columnas: el primero indicará el nombre de la nueva columna que repre-

6. Recordad que los nombres de columnas no pueden empezar con un número, de manera que, en R, ponemos una X delante. Más adelante ya la quitaremos.

sentará los valores que hasta ahora eran títulos de columna (la llamaremos *columna clave*), mientras que el segundo nos indicará el nombre de la nueva columna que representará los datos que estaban en las observaciones (la llamaremos *columna valor*). En último lugar, indicaremos los nombres de las columnas que hay que agrupar. Si son todas las columnas del marco de datos, no hace falta que indiquemos nada. Si son varias columnas, lo tendremos que especificar con un concatenado. Es muy posible que nos sea más cómodo indicar con el símbolo negativo las columnas que no se tienen que agrupar. Las dos posibilidades, que especificamos a continuación, darán el mismo resultado.

```
gather(marco de datos, columna clave, columna valor, columnas
que se tienen que agrupar)
tabla_limpia <- gather(tabla_sucia, ano, protestas, c(X2016,
X2017, X2018))
tabla_limpia <- gather(tabla_sucia, ano, protestas, -pais)
```

Ahora tenemos el nuevo marco de datos `tabla_limpia`, que cumple los requisitos de tener las observaciones en las filas y las variables en las columnas. Para acabar de limpiar la tabla correctamente todavía tendremos que dar un paso más, puesto que en la columna de año tenemos los valores con una X al principio: X2016, X2017 y X2018. Una manera de quitar esta X es con la función `separate()`, que separa los valores de una columna por el carácter que indicamos. Dentro de la función tenemos que indicar el nombre del marco de datos, seguido de la columna que contiene los valores que queremos separar. A continuación, introducimos un vector con el nombre acompañado de comillas que tomarán las nuevas columnas. Finalmente, con el atributo *sep* indicaremos,

también entre comillas, cuál es el carácter o caracteres que hacen de separadores y que, por lo tanto, serán eliminados.

Limpiar vectores con stringr

Otra opción que permite eliminar parte de un vector de carácter es con la función `string_replace()` del paquete stringr. En el ejemplo, tendríamos que introducir la operación siguiente: `str_replace(tabla_limpia$ano, "X", "")`.

```
separate(marco de datos, columna, "nombre nuevas columnas", sep
= " ")
tabla_sep <- tabla_limpia %>%
   separate(ano, c("nada", "ano"), sep = "X") %>%
   select(-nada) %>%
   head(8)
```

Si solo utilizamos la función `separate()`, R nos creará dos columnas: la columna *año* con el año correspondiente y la columna *nada*, que tendrá todos los valores en blanco, puesto que no hay nada a la izquierda de la X que hemos indicado como separador.[7] Para poder eliminar la columna *nada*, en la función anterior hemos abierto una *pipe* para el objeto `tabla_limpia`. Después de separar la columna de interés, hemos utilizado la función `select()` de dplyr para eliminar la columna *nada*. Finalmente, hemos imprimido las ocho primeras filas de la tabla resultante.

7. Una manera más rápida de quitar la X es con `separate(tabla_limpia, año, into = "ano", extra = "drop", sep = "X")`.

El paquete tidyr también nos permite hacer el procedimiento contrario al que acabamos de ver. La función `spread()` hace lo contrario que `gather()`, es decir, podemos tomar los valores categóricos de una columna y convertirlos en títulos de columna. La función que indicamos en el siguiente código nos ilustra este procedimiento, en el cual introducimos primero el nombre del marco de datos y, en segundo lugar, la columna clave del marco de datos original que contiene los datos que queremos que se conviertan en los títulos de las nuevas columnas. En tercer lugar, introduciremos el nombre de la columna del marco de datos original que contiene los valores de las columnas que crearemos. Recordad que, al referimos en estos procesos a nombres de variables existentes, no nos hace falta especificar el nombre entre comillas.

Seleccionar el separador

Utilizar el argumento `sep` es, en muchas ocasiones, opcional. R interpretará de manera automática que el separador es un carácter no alfabético como un espacio, un guion, una barra baja, etc. y lo eliminará. En el supuesto de que el símbolo sea un punto (`.`), lo tendremos que indicar así: `"\\."`.

```
spread(marco de datos, columna clave, columna valores)
spread(tabla_limpia, ano, protestas)
```

Del mismo modo que `spread()` hace lo contrario que `gather()`, la función `unite()` hará lo contrario que `separate()`. Es decir, esta función nos une los datos de dos columnas en una misma columna. Por defecto, `unite()` unirá los valores indicados con un guion bajo. Si queremos que utilice otro símbolo

para unir los datos, lo tendremos que indicar expresamente. En el ejemplo que veremos a continuación, hay un marco de datos creado anteriormente, `tabla_sep`, que tiene los valores *año* y *mes* separados en dos variables diferentes.

```
> tabla_sep
    pais   ano    mes   protestas
1   España 2016   6     42
2   España 2017   1     33
3   España 2017   6     83
4   Francia 2016  6     42
5   Francia 2017  1     44
6   Francia 2017  6     34


unite(marco de datos, columna nueva, columna1, columna2 ...,
sep = "_")
unite(tabla_sep, ano_mes, ano, mes, sep = "/")
```

Si queremos juntar los valores de las columnas *año* y *mes* con una barra inclinada, tenemos que indicar dentro de `unite()` el marco de datos, seguido del nombre de la columna que queremos crear. A continuación, introduciremos el nombre de las dos columnas del marco de datos que queremos unir. En este caso, no tenemos que indicar el nombre de las columnas entre comillas. Finalmente, introduciremos *sep* seguido del símbolo que queremos que una los valores.

3.2. Convertir variables

El segundo problema que podemos tener con un marco de datos es que las variables no tengan asignado el tipo de vector que queremos. Por ejemplo, podemos encontrar que algunas de las variables categóricas que queremos como factores estén codificadas como vectores de carácter o que algunas numéricas sean dobles en lugar de números enteros. Parte del problema lo podemos solucionar cuando descargamos la base de datos con atributos que tienen algunas funciones de tipo `StringAsFactors = FALSE`, pero es posible que una vez descargadas queramos solucionar algunos casos.

Las funciones que tenemos que utilizar para convertir variables ya las conocemos: `as.numeric()`, `as.character()`, `as.integer()`, `as.logical()`, etc. Sin embargo, tenéis que tener en cuenta algunas consideraciones, sobre todo cuando convertimos factores en otros tipos de variables. Como sabéis, los factores son vectores enteros tratados como variables categóricas. Como tales, cada categoría es realmente un número entero con una etiqueta con información categórica. Cuando pasamos un factor a carácter, convertiremos las etiquetas en *strings* y eliminará los números enteros. Por el contrario, cuando pasamos un factor a numérico convertiremos los números enteros en numéricos y eliminará las etiquetas. En el supuesto de que las etiquetas de un factor sean números y nos interese guardarlas como tales, tendremos que hacer una conversión doble: primero, transformar el factor en carácter para obtener los números y, después, transformarlo en numérico: `as.numeric(as.character(factor))`.

Estas conversiones no hace falta que las hagamos una por una, sino que también las podemos hacer a gran escala. La función `mutate_at()`del paquete dplyr permite tomar muchas variables

de un marco de datos y convertirlas de tipos en una sola operación. Supongamos que queremos cambiar a vector numérico las columnas 6 a 50 del marco de datos `pol`. Introduciremos `mutate_at(pol, vars(6:50), funs(as.numeric))`. Podemos indicar indistintamente el nombre de la columna o el número de la columna.

Quizás en algún momento durante la limpieza del marco de datos también querréis crear una columna que sea sencillamente el orden de las filas. Para crear la hipotética columna `num` en el marco de datos `md`, tendríais que utilizar el código siguiente: `md$num <- 1:nrow(md)`.

3.3. Eliminar filas y columnas

Una parte importante de la limpieza de los marcos de datos es eliminar las filas y las columnas que sabemos que no usaremos. Eliminarlas facilita el trabajo y significa más velocidad para R a la hora de procesar la información. Como ya sabéis, dplyr tiene la función `filter()` para eliminar filas y `select()` para eliminar columnas. Alternativamente, también podéis utilizar los corchetes para suprimir las filas y las columnas que no nos interesen. Recordad que, para seleccionar un marco de datos, dentro del corchete tenemos que introducir un vector con la selección de filas y, separado por una coma, el vector con la selección de columnas. El símbolo negativo indica que, en lugar de seleccionarlas, las queremos eliminar. En el siguiente código tenéis varios ejemplos de selecciones:

```
md <- md[-c(4, 23:50), ]

md <- md[ ,-c(14:ncol(md))]

md <- md[1000:nrow(md), -c(1, 5:7, 10:20)]
```

En el primer código eliminamos la fila 4 y de la 23 a la 50. En el segundo ejemplo pedimos suprimir desde la columna 14 hasta el final. En el tercero queremos conservar desde la fila 1.000 hasta el final y eliminar las columnas 1, 5, 6, 7 y de la 10 a la 20. Las funciones `ncol()` y `nrow()` nos devuelven el número total de filas y columnas respectivamente.

3.4. Anomalías en los datos

Cuando cargamos una base de datos de la red, también es habitual que tenga un contenido incompleto. No siempre todas las casillas tienen datos y, a veces, estos datos no son correctos. En el caso de casillas sin datos (datos perdidos), tendremos que detectarlos haciendo una observación rápida al marco de datos una vez lo hayamos descargado. La función más genérica y útil para detectar datos perdidos es la de `summary()`. En caso de que haya datos perdidos, R nos devolverá una fila extra en la descripción de la variable indicando el número de datos perdidos. En cuanto a los errores, no hay ninguna solución mágica para encontrarlos, pero con un sumario podemos detectar, algunas veces, valores que se escapan de toda lógica.

3.4.1. Datos perdidos

En R veremos los datos perdidos codificados como NA (*Not Available*, no disponible).[8] Hay infinidad de razones por las que

8. Hay que distinguir los datos perdidos (NA) de los valores infinitos, señalizados con *Inf*, y de los valores que no son números, señalizados con *NaN* (*Not a Number*).

podemos tener datos perdidos en una base de datos. En los estudios internacionales, la presencia de datos está muy asociada a la capacidad de cada estado de recolectarlos (Stone, 2008). Así, si buscamos dentro del banco de datos del Banco Mundial, es muy probable que tengamos mucha información de Estados Unidos, pero, en cambio, variables vacías de información sobre países pequeños o en desarrollo como Tonga o Angola. Los datos perdidos también pueden ser aleatorios debido a errores al codificar la información.

Para saber si tenemos NA en nuestro marco de datos, lo primero que podemos hacer es especificar el marco de datos en cuestión dentro de la combinación de funciones `any(is.na())`. R nos devolverá TRUE si tenemos algún dato perdido y FALSE si no tenemos ninguno. Si R nos dice que no tenemos ninguno, ya podemos saltar al paso siguiente. Pero si R nos dice que sí que tenemos, podemos pedir que cuente la cantidad de datos perdidos del marco de datos con la combinación de funciones `sum(is.na())`. R sumará la cantidad de TRUE del marco de datos.

La función `is.na`

Si preguntamos simplemente `is.na()`, R devolverá el mismo marco de datos entero, pero con valores TRUE o FALSE en cada casilla, de manera que tendremos marcados los datos perdidos como TRUE y los otros como FALSE. Este método no es recomendable con grandes marcos de datos.

Una vez sabemos si tenemos o no NA, el segundo paso es localizar dónde están. A ello nos puede ayudar la función `summary()`. Si la aplicamos al marco de datos, obtendremos los estadísticos descriptivos de cada variable y, si alguna variable tiene

datos perdidos, veremos el número de NA al final del sumario de la variable. A continuación, hemos pedido un sumario hipotético de la variable *md$gdp_cap*. R nos devuelve lo siguiente:

```
> summary(md$gdp_cap)
Min.      1st Qu.   Median    Mean      3rd Qu.   Max.       NA's
-241.2    1413.1    4291.9    7215.3    9325.5    213623.1   4
```

Tal como vemos al final del sumario, esta variable tiene cuatro datos perdidos. Para identificar en qué fila se encuentra exactamente cada uno de estos NA, podemos utilizar la función `which(is.na())`, que devolverá un vector numérico con la posición de las filas que tienen datos perdidos de la variable indicada.[9] Ahora solo tenemos que visualizar las filas en cuestión seleccionándolas del marco de datos. Supongamos que la operación anterior ha devuelto las filas 34, 567, 1.234 y 1.520. Podemos crear un vector con estos números y seleccionarlos como filas en el marco de datos.

```
> na_md <- c(34, 567, 1234, 1520)
> md[na_md, ]
    pais      continente   ano     gdp_cap
1   Chad      Africa       1993    NA
2   Myanmar   Asia         2005    NA
3   Senegal   Africa       2001    NA
4   Tonga     Oceania      1999    NA
```

9. Es mejor hacer esta operación con vectores, no con marcos de datos. Si indicamos todo el marco de datos, R devolverá el número que tienen las casillas donde hay NA. Esta información nos será menos útil.

Llegados a este punto, la gran pregunta es: «¿Qué hacemos con estos datos?». Tenemos dos opciones principales: eliminar la fila entera o sustituir los datos perdidos por otros. No hay una única respuesta a esta pregunta y dependerá en buena parte de la posibilidad de averiguar cuáles son los datos que faltan o de la validez de los métodos para imputar un valor a los NA. El manual *Handbook on Constructing Composite Indicators* de la OCDE (2008) ofrece varias posibilidades a la hora de imputar nuevos valores. Si decidimos eliminar las filas, el paquete tidyr nos ofrece una solución rápida con la función `drop_na()`, que devolverá un nuevo marco de datos con las filas que no tengan valores perdidos.[10]

Si lo que queremos es sustituir los datos perdidos por unos valores concretos, tenemos dos posibilidades:

- la opción automática que nos ofrece tidyr, o
- la opción manual con corchetes.

Si aplicamos la función `fill()` de tidyr a un vector, llenará la fila con el valor más reciente, mientras que `replace_na()` hará una estimación a partir de los valores superior e inferior de la casilla donde tengamos el NA.

Añadir NA a un marco de datos

Es posible que encontremos variables con filas vacías que queramos cambiar a NA. Para convertirlas en NA aplicaremos `md$var[md$var == ""] <- NA`.

10. La función `na.omit()` del paquete de base *stats* hace el mismo procedimiento.

La opción manual consiste en identificar la columna y el número o números de las casillas que tengan datos perdidos y asignar individual o conjuntamente un nuevo valor. Como veremos a continuación, en el primer caso hemos asignado el valor 2.045 a la fila 34 de la variable *md$gdp_cap*, mientras que en el segundo caso hemos aprovechado el vector `na_md` creado previamente para asignar el valor 2.045 a las cuatro filas con datos perdidos.

```
md$gap_cap[34] <- 2045
md$gap_cap[na_md] <- 2045
```

3.4.2. Valores extremos

Otro tipo de anomalía que podemos encontrar en un marco de datos son los valores extremos. Un valor extremo es un valor fuera de lo normal en nuestra distribución. Es posible que este valor sea perfectamente válido, como el caso del PIB per cápita de Kuwait en el marco de datos `gapminder`.[11] Por lo tanto, que sea extremo no significa que no sea «real», pero siempre será motivo de sospecha. Los valores extremos son normalmente consecuencia de errores en el proceso de creación de los datos o fruto de codificaciones especiales.

Las codificaciones de Polity IV

La base de datos Polity IV codifica los países en una escala de -10 a +10 según si son más autocráticos o más democráticos. Pero los

11. Si tenéis cargado el paquete dplyr, haced la prueba: `arrange(filter(gapminder, year == 1952), desc(gdpPercap))`.

valores toman codificaciones especiales cuando son regímenes en transición (código -88), interrupción del régimen (código -66) o periodo *interregnum* (código -77). Usar Polity IV comporta tener que tomar decisiones sobre cómo recodificar estos valores.

Como ya hemos explicado, no hay ninguna fórmula mágica para identificar valores extremos. Lo más importante es hacer una lectura atenta de los datos, explorar cada variable y pensar si tiene sentido que los datos tengan el valor que tienen. Algunos errores son fáciles de detectar. Si sabemos, por ejemplo, que la esperanza de vida mediana oscila normalmente entre 50 y 82 años, nos sorprenderá mucho si encontramos un valor de 900. También será difícil que un país tenga un PIB per cápita negativo. Por lo tanto, lo más importante es hacer un esfuerzo inicial para conocer a fondo qué miden las variables que queremos trabajar, qué unidades de medida utilizan y entre qué valores acostumbran a oscilar estas unidades.

Fijémonos bien en el sumario que hemos hecho en la sección anterior. El valor mínimo de la variable *md$gdp_cap* es negativo y, por lo que sabemos, los PIB per cápita no suelen ser negativos. También observamos que el valor máximo supera los 200.000 dólares por habitante y ningún país acostumbra a superar los 100.000. Nuevamente podemos utilizar la función `which()` para identificar la fila o filas que tienen valores poco habituales. En el primer caso hemos seleccionado las filas con valores negativos de la variable *md$gdp_cap*, mientras que en el segundo caso hemos buscado los valores extremos de dos maneras: mirando qué valores son superiores a 100.000 o directamente yendo a buscar el número concreto que habíamos encontrado a partir de la función `summary()`.

```
md[which(md$gdp_cap < 0), ]
    pais       continente   ano    gdp_cap
1   Singapur   Asia         1991   -241.2

> md[which(md$gdp_cap > 100000), ]
> md[which(md$gdp_cap == 213623.1), ]
    pais       continente   ano    gdp_cap
1   Canada     Africa       2005   213623.1
```

Una manera alternativa de identificar casos extremos es visualizando la distribución de una variable. Normalmente con `hist()` o `plot()` podremos comprobar si hay valores que se salen de la norma. Una vez los tenemos identificados, el procedimiento vuelve a ser el mismo que para el caso de los NA.[12]

4. Unir marcos de datos

Los analistas de datos a menudo quieren analizar la relación entre dos variables, pero resulta que estas variables están separadas en dos bases de datos diferentes. Por ejemplo, podría ser que en una base de datos tuviéramos información sobre democracia y en otra tuviéramos información sobre conflictos. Nos gustaría saber si hay alguna relación entre democracia y conflicto, pero para ello, antes que nada, tendríamos que unir ambas bases de

12. Por ejemplo, hemos encontrado en otra fuente que el PIB per cápita de Singapur en 1991 era de 11.463 dólares. Podemos utilizar este valor en el marco de datos: `md$gap_cap[which(md$gdp_cap < 0)] <- 11463`.

datos en una. Unirlas puede ser un proceso complicado, no solamente por el gran volumen de información que puede contener cada una de ellas, sino por varios obstáculos que puede haber: que la unidad de análisis no coincida o que el nombre de los países que queremos unir está registrado en idiomas diferentes. Normalmente, ante este tipo de situación, la solución que se acaba adoptando es unirlas manualmente, fila a fila, en un proceso que puede tardar horas y horas. Por suerte, programas como R tienen recursos para unir tablas con cierta facilidad. En este apartado descubriremos la segunda gran virtud del paquete dplyr: aparte de manipular datos, esta librería también tiene como segunda especialidad unir marcos de datos.

Métodos para unir marcos de datos

Las funciones del paquete dplyr tienen una sintaxis muy intuitiva y que puede aplicarse a otros objetos aparte de a un marco de datos. Podéis ver algunos ejemplos en este enlace: https://dplyr.tidyverse.org/ reference/join.html). Aparte de dplyr, hay una función de base de R, `merge()`, que también puede unir tablas.

Para juntar dos marcos de datos con dplyr, el escenario ideal que nos gustaría encontrar es que hubiera una columna común de referencia a las dos tablas. Es decir, como veremos en el ejemplo siguiente, el marco de datos `md_dem` (que indica si han tenido un régimen democrático en los últimos treinta años) y el marco de datos `md_war`[13] (que indica hipotéticamente el número de disputas militares en los últimos treinta años) tienen una columna

13. Los códigos de los marcos de datos `md_dem` y `md_war` están disponibles en el anexo.

común, *country*, que tiene las categorías con el mismo nombre. Francia es *France* en los dos marcos de datos, España es *Spain*, y así sucesivamente. En esta situación ideal, juntar marcos de datos es sencillo con dplyr.

```
> md_dem                    > md_war
     country   dem          country   war
1    France    TRUE     1   France    21
2    Spain     TRUE     2   Spain     13
3    Germany   TRUE     3   Germany   9
4    China     FALSE    4   China     24
5    Yemen     FALSE    5   Yemen     14
6    Haiti     FALSE    6   Congo     42
```

A la columna que ambos marcos de datos tienen en común, la denominaremos *columna clave* y será la que nos servirá para juntar las bases de datos. Para unirlas, siempre entenderemos que hay un marco de datos de referencia (que recibirá los datos del otro marco de datos) y un marco de datos secundario. Supongamos, en el caso que nos ocupa, que queremos traer los datos de `md_war` hacia `md_dem` y como referencia tenemos la columna *country*. A la columna *country* de `md_dem` la llamaremos *columna clave primaria*, mientras que a la columna *country* de `md_war` la denominaremos *columna clave secundaria*.

La distinción es importante, puesto que en algunas operaciones R tendrá que decidir si conserva algunos valores, y en estos casos siempre conservará los valores de la clave primaria.

Más de una columna clave

En la unión de dos marcos de datos podemos identificar más de una columna clave. Aunque en los ejemplos hemos trabajado con una sola columna clave, muchas veces utilizaremos dos. Trabajar con dos columnas clave es habitual, por ejemplo, cuando tenemos los datos agrupados por país y año. En este caso, la sintaxis será: `xxxx_join(x, y, by = c("pais", "ano"))`.

En la tabla 18 vemos las seis funciones que estudiaremos de dplyr. Todas ellas tienen la misma sintaxis, formada por tres argumentos dentro de la función: el nombre del primer marco de datos, el nombre del segundo marco de datos y el argumento `by` seguido del nombre de la columna clave entre paréntesis.

Tabla 18. Unir marcos de datos con dplyr

`left_join()`	Devuelve todas las filas del primer marco de datos y las filas del segundo marco de datos que coincidan con la columna o las columnas clave.
`right_join()`	Devuelve todas las filas del segundo marco de datos y las filas del primer marco de datos que coincidan con la columna o las columnas clave.
`inner_join()`	Devuelve las filas del primer marco de datos y del segundo que coincidan con la columna o las columnas clave.
`full_join()`	Devuelve todas las filas y columnas del primer marco de datos y del segundo.
`semi_join()`	Devuelve las filas del primer marco de datos que coincidan con la columna o las columnas clave del segundo marco de datos.
`anti_join()`	Devuelve las filas del primer marco de datos que no coincidan con la columna o las columnas clave del segundo marco de datos.

Fuente: elaboración propia.

La única diferencia entre `left_join()` y `right_join()` es que la primera utiliza el primer marco de datos como primario, mientras que la segunda utiliza el segundo. Con `left_join()`, R devolverá todas las filas del primer marco de datos y añadirá cualquier fila del segundo marco de datos que encaje con la columna clave. Con `right_join()` R hará exactamente lo contrario: devolverá todas las filas del segundo marco de datos y añadirá cualquier fila del primer marco de datos que encaje con la columna clave.

La sintaxis de las funciones `join`

La sintaxis en código sería `xxxx_join(x, y, by = "columna")`. Si el nombre de la columna clave no coincide en los dos marcos de datos, lo especificaremos de la siguiente manera: `xxxx_join(x, y, by = c("nombre_col1" = "nombre_col2"))`. R mantendrá el nombre del primer marco de datos. Si no indicamos el argumento `by` en la sintaxis, R mirará los dos marcos de datos y combinará las columnas que encuentre con el mismo nombre. En la consola, R explicará qué procedimiento ha seguido.

Fijaos cómo, a continuación, unimos los marcos de datos `md_dem` y `md_war` de dos maneras diferentes. Con `left_join()`, hemos conservado Haití porque estaba en el primer marco de datos, pero, al no tener datos para Haití de la columna *war*, nos ha devuelto un NA. Con `right_join()` conservamos los datos del segundo marco de datos y, por lo tanto, tenemos el Congo en lugar de Haití. R ha buscado los datos de democracia del Congo, pero, al no encontrarlos en la clave secundaria, nos devuelve un NA.

```
> left_join(md_dem, md_war)
    country    dem     war
1   France     TRUE    21
2   Spain      TRUE    13
3   Germany    TRUE    9
4   China      FALSE   24
5   Yemen      FALSE   14
6   Haiti      FALSE   NA

> right_join(md_dem, md_war)
    country    dem     war
1   France     TRUE    21
2   Spain      TRUE    13
3   Germany    TRUE    9
4   China      FALSE   24
5   Yemen      FALSE   14
6   Congo      NA      42
```

Las dos funciones que acabamos de ver, `inner_join()` y `full_join()`, tienen una lógica muy parecida. En el caso de `inner_join()`, R solo devolverá las filas que aparezcan en los dos marcos de datos, mientras que en el caso de `full_join()` obtendremos los valores que aparezcan tanto en un marco de datos como en el otro. Lógicamente, `full_join()` producirá muchos NA. Como podemos ver a continuación, el código de la izquierda nos une los marcos de datos de manera exclusiva, puesto que eliminará todo lo que no tenga información en los dos marcos de datos, como sucede en el caso de Haití y del Congo.

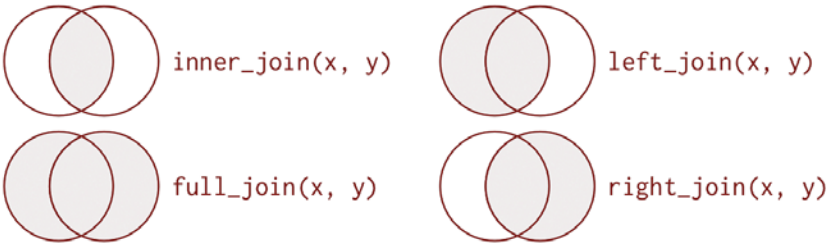
La segunda función es inclusiva, puesto que conserva los valores, aunque solo aparezcan en un marco de datos.

```
> inner_join(md_dem, md_war)
     country    dem     war
1    France     TRUE    21
2    Spain      TRUE    13
3    Germany    TRUE    9
4    China      FALSE   24
5    Yemen      FALSE   14

> full_join(md_dem, md_war)
     country    dem     war
1    France     TRUE    21
2    Spain      TRUE    13
3    Germany    TRUE    9
4    China      FALSE   24
5    Yemen      FALSE   14
6    Haiti      FALSE   NA
7    Congo      NA      42
```

Estas cuatro funciones de dplyr que acabamos de ver están resumidas visualmente en la figura 29. En esta imagen vemos la lógica de cada función y observamos claramente que `inner_join()` es más exclusiva que `full_join()`.

Figura 29. Lógica de unión de las funciones de dplyr



Fuente: Grolemund, G.; Wickham, H. (2017). *R for Data Science*. O'Reilly. CC BY-NC-ND 3.0 OS.

Lo que hemos visto hasta ahora con estas cuatro funciones de dplyr ha sido cómo unir valores de dos marcos de datos diferentes en una o varias columnas clave. A continuación, veremos dos funciones que cumplen una tarea parecida, pero que no incorporan los valores del segundo marco de datos. Es decir, lo que hacen es filtrar el primer marco de datos con criterios del segundo, pero no incorporan los valores del segundo marco de datos en el primero. Estas dos funciones son `semi_join()` y `anti_join()`. La función `semi_join()`filtra el primer marco de datos según los valores que coinciden con el segundo marco de datos. La función `anti_join()` hace todo lo contrario que `semi_join()`: filtra el primer marco de datos según los valores que no coinciden con el segundo marco de datos. En el primer caso, el único valor que no coincidía con el segundo marco de datos es Haití, por lo cual lo ha filtrado del marco de datos original. En el segundo caso, ha procedido a la inversa. Como Haití era el caso que no coincidía con el segundo marco de datos, lo ha mantenido.

```
> semi_join(md_dem, md_war)
   country   dem
1  France    TRUE
2  Spain     TRUE
3  Germany   TRUE
4  China     FALSE
5  Yemen     FALSE

> anti_join(md_dem, md_war)
   country   dem
1  Haiti     FALSE
```

A simple vista, parece que `semi_join()` y `anti_join()` no tengan demasiada utilidad práctica, pero en realidad es más bien lo contrario. La función `semi_join()` va bien para conservar aquellos datos que cumplan unos requisitos determinados. La función `anti_join()` suele ser muy útil para diagnosticar problemas. Supongamos que tenemos dos marcos de datos que queremos unir, pero hay filas que no nos une correctamente. Seguramente esto se debe a que estos marcos de datos no son exactamente iguales. Las filas en cuestión serán fáciles de identificar con un *anti join*.

Ejercicios de autoevaluación

Para un mejor aprendizaje, intentad hacer mentalmente el máximo de ejercicios posible sin utilizar R.

1) Cread el marco de datos `const_pol` pidiendo la siguiente variable de Polity IV.

```
exconst
```

2) Buscad la siguiente palabra en la base de datos de WB Stats y pedid los siete primeros resultados.

```
women
```

3) Elegid la mejor función para descargar el siguiente archivo en formato europeo.

```
preguntasexamen.csv
```

4) Descargad la segunda hoja de este documento. Tened en cuenta que no hay una fila con los nombres de columna.

```
solucionesPEC.xls
```

5) Transformad en columnas los valores de la primera variable con parámetros de la segunda variable.

```
wbtrade$regions, wbtrade$gdp
```

6) El siguiente marco de datos contiene la variable *date* con los meses y los años separados por un guion bajo. Separad las variables.

```
wbtrade
```

7) Convertid los nombres de columna del siguiente marco de datos en mayúscula.

```
un_votes
```

8) Detectad qué valor de la variable *age* puede ser un error.

```
35, 40, NA, 42, 15, 17, 49, 65, 37, -2, 47, 21
```

9) Unid de manera exclusiva los siguientes marcos de datos.

```
un_votes / un_roll_call_issues
```

10) Queremos juntar los marcos de datos siguientes por la columna *country,* de manera que nos filtre las filas del primer marco de datos según los valores del segundo.

```
europe, efta
```

Solucionario

1) `const_pol <- PolityGet(vars = "exconst")`

2) `wbstats("women")[1:7,]`

3) `read.csv2("preguntasexamen.csv")`

4) `read_excel("solucionesPEC.xls", sheet = 2, col_names = FALSE)`

5) `spread(wbtrade, regions, gdp)`

6) `separate(wbtrade, date, c("month", "year"), sep = "_")`

7) `toupper(names(un_votes))`

8) `-2`

9) `inner_join(un_votes, un_roll_call_issues)`

10) `semi_join(europe, efta)`

Bibliografía

Lohr, S. (2014). «For Big-Data Scientists, 'Janitor Work' is Key Hurdle to Insights». *The New York Times*. [Fecha de consulta: 15 de julio de 2020] <https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html>

Stone, R. W. (2008). «The Scope of IMF Conditionality». *International Organization* (vol. 62, n.º 4, págs. 589-620).

Wickham, H. (2014). «Tidy Data». *Journal of Statistical Software* (vol. 50, n.º 10, págs. 1-23).

Anexo

Código del apartado 3.1.1

```
tabla_sucia <- data.frame(pais = c("España","Francia","Italia",
                                   "México", "Japón"),
                          `2016` = c(64, 43, 56, 35, 12),
                          `2017` = c(83, 34, 53, 101, 19),
                          `2018` = c(95, 33, 67, 120, 18))
```

Código del apartado 4.1

```
md_dem <- data.frame(country = c("France", "Spain", "Germany",
"China", "Yemen", "Haiti"),
                     dem = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE))
md_war <- data.frame(country = c("France", "Spain", "Germany",
"China", "Yemen", "Congo"),
                     war = c(21, 13, 9, 24, 14, 42))
```

Conclusión

Hay una frase atribuida al primer ministro británico Benjamin Disraeli y popularizada por el célebre escritor Mark Twain que dice:

«Hay tres tipos de mentiras: mentiras, mentiras podridas y la estadística.»

La estadística es una herramienta muy poderosa para explicar asociaciones entre fenómenos, pero también es una herramienta muy peligrosa si no interpretamos su utilidad correctamente. Al fin y al cabo, lo único que nos sugiere esta metodología es que parece que los datos nos den la razón, pero esto no quiere decir que la tengamos. Pensémoslo con un ejemplo. Podemos decir que, a medida que un país se hace más democrático, acostumbra a generar también más riqueza. Esta afirmación puede ser confirmada con las técnicas de análisis de datos que hemos aprendido en este libro. Lo que no podemos afirmar con seguridad es si es la democracia la que causa la riqueza o bien si es la riqueza la que causa la democracia. En nuestro análisis multivariante, donde habremos incluido varias hipótesis alternativas en nuestra hipótesis principal, tampoco podemos decir en realidad que no haya otras hipótesis rivales y plausibles que no hayamos incluido en el modelo y que puedan hacer tambalear nuestras conclusiones. Es por eso por lo que el análisis de datos estadístico es un primer paso, necesario pero no suficiente, para responder a las preguntas que nos formulamos.

En este sentido, la interpretación de los datos tiene un papel fundamental: debemos tratar de no sugerir erróneamente que la

correlación entre dos fenómenos implica que uno cause el otro. Tampoco hay que llegar al punto de obsesionarse con observar solo los niveles de significación estadística y obviar otra información sustantiva (Braumoeller; Sartori, 2004, pág. 131). Los datos pueden darnos resultados significativos, pero con efectos estadísticamente minúsculos, simplemente por el hecho de que disponemos de una población enorme de casos. Por eso, tenemos que apreciar también las bondades que tiene la visualización de los datos, que nos puede ayudar a observar rasgos sustantivos que nos ayuden a formular nuevas preguntas relevantes.

Este libro, en resumen, ha explicado los procedimientos esenciales para la obtención y el tratamiento de datos en estudios internacionales, así como algunas de las principales técnicas estadísticas para analizar dicha información. Pero tan importante es utilizar buenas técnicas, sustentadas matemáticamente, como utilizar buenas razones, sustentadas teóricamente, para tratar de demostrar un argumento. Es por eso por lo que una interpretación humilde de los datos y una comunicación clara y transparente ayudarán a añadir valor a nuestros hallazgos como analistas de datos.

Correlaciones absurdas

Podéis ver algunos ejemplos de correlaciones absurdas en esta web (http://www.tylervigen.com/spurious-correlations), que muestra que el número de personas que se ahogaban en una piscina anualmente entre 1999 y 2009 estaba correlacionado con el número de películas anuales en que aparecía Nicolas Cage.

Bibliografía

Braumoeller, B. F.; Sartori, A. E. (2004). «The promise and perils of statistics in international relations». En: D. F. Sprinz e Y. Wolinsky-Nahmias (eds.). *Models, Numbers, and Cases: Methods for Studying International Relations*, págs. 129-151. Ann Arbor, MI: University of Michigan Press.

JORDI MAS ELIAS

ANÁLISIS DE DATOS

CON R EN ESTUDIOS INTERNACIONALES

DERECHO Y CIENCIA POLÍTICA

La gestión de datos de tipo cuantitativo es cada vez más importante en las Ciencias Sociales. Este libro muestra cómo tratar datos mediante el programa RStudio, a través de ejemplos del ámbito de los estudios internacionales y la Ciencia Política. Incluye una iniciación básica al entorno R y una familiarización con los principales procesos actualmente imprescindibles en el análisis de datos: descargar y limpiar bases de datos, unir datos procedentes de diferentes orígenes y transformar bases de datos para examinarlas con técnicas de análisis univariante y bivariante. El manual hace especial énfasis en la visualización y la comunicación de los resultados.

Con este libro aprenderás sobre:

✓ análisis de datos; ✓ metodología cuantitativa; ✓ estadística; ✓ RStudio; ✓ R; ✓ relaciones internacionales; ✓ estudios internacionales; ✓ Ciencia Política



Jordi Mas Elias

Profesor de los Estudios de Derecho y Ciencia Política en la Universitat Oberta de Catalunya (UOC). Sus principales intereses tanto docentes como investigadores se centran en la Economía Política Internacional y la metodología en Ciencias Sociales.

EDITORIAL UOC