Análisis de datos con R

PID_00268327

Jordi Mas Elias

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya

Jordi Mas Elias

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Jordi Mas Elias (2019)

Primera edición: septiembre 2019

Avda. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

Índice

Introducción

El objetivo de este módulo es introducir al estudiante en el análisis de datos por medio de métodos cuantitativos, orientados a la explotación estadística de los datos, generalmente en forma de números y categorías cuantificables, a partir de un conjunto de técnicas estandarizadas. Si bien las técnicas cuantitativas no son las únicas empleadas en ciencias sociales, sí que son unas de las más utilizadas y su estudio genera a menudo cierta ansiedad en los estudiantes. Por eso, este módulo intenta mostrar de una manera práctica los principales elementos que el estudiante necesita para iniciarse en el análisis de datos. La estructura del módulo está pensada para que el estudiante aprenda primero a hacer operaciones sencillas y acabe siendo capaz de moverse con relativa facilidad entre bases de datos de un tamaño considerable.

Para saber más

Para una visión general de los diferentes métodos utilizados en ciencias sociales, ved Bennett y George (2005, capítulo 1), Brancati (2018, capítulo 7), Goertz y Mahoney (2013, capítulos 1 y 2), Halperin y Heath (2016, capítulos 5 y 6).

Por cuestiones prácticas, el análisis de datos comporta necesariamente la utilización de software informático para emplear de manera ágil y rápida operaciones complejas. Este módulo utiliza el entorno de programación R por varios motivos:

1) Es un software libre accesible para todo el mundo y con uno de los lenguajes más populares en ciencia de datos.

2) Entre los softwares libres, R es lo más popular en ciencias sociales, está especialmente pensado para el autoaprendizaje y resulta más intuitivo que otros programas como Python.

3) Tiene una comunidad de usuarios muy activa que permite resolver las dudas de funcionamiento que se puedan tener.

Para saber más

Ved, por ejemplo, Stack Overflow (https://stackoverflow.com/), la comunidad de usuarios de RStudio (https://community.rstudio.com/) o la de DataCamp (https://www.datacamp.com/community), donde podéis acceder a tutoriales y guías del programa. En cuanto a los manuales de R, encontraréis en español un manual un poco antiguo de R Development Core Team (2000). También podéis consultar Grolemund y Wickham (2016).

4) Es un programa que trabaja con código. A pesar de que esto puede representar un obstáculo inicial para el aprendizaje, el utilizar un lenguaje de código comporta grandes beneficios para el estudiante y su investigación. Aprender un lenguaje de programación es una habilidad cada vez más importante en cualquier ámbito profesional.

Trabajo con código

Imaginémonos que descargamos una base de datos de internet, hacemos un análisis de datos y presentamos en público estos datos con unas diapositivas. ¿Qué ha pasado entremedias? Nadie sabe qué hemos hecho con los datos en nuestro pequeño laboratorio. Se asume que hemos sido honestos y no hemos hecho manipulaciones sospechosas 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 lo podemos transferir a otras personas para que puedan obtener los mismos resultados a los que hemos llegado. Esto mejora la transparencia de nuestros análisis y el rigor de nuestros resultados.

Este módulo explica el lenguaje de R que necesitamos aprender y el tipo de variables que tenemos que utilizar para el análisis 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 las pueda reproducir en el programa. Normalmente, el código se muestra dentro de una caja gris. Cuando, dentro de la caja gris, aparecen encabezados por el símbolo > significa que dentro mismo de la caja mostramos imprimido el resultado del código. Para aclarar una distinción 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 una manera específica que R utiliza para almacenar datos.

Para saber más

Para refrescar algunos conceptos básicos, véase el Cheat sheet IDE (https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf) o consúltense otros recursos que se proporcionan en este curso.

1. El lenguaje de R

Para aprender a utilizar programas de análisis de datos como R, Python, Stata o SPSS, lo más importante que nos hace falta inicialmente es dominar el lenguaje. En general, dominar un lenguaje de programación equivale a dominar 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 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.

A continuación, detallamos primero los objetos y los atributos, y en segundo lugar las funciones y los argumentos.

1.1. 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.

Los objetos de R

Hay otros tipos de objetos que no estudiaremos. Los más comunes son la lista, la matriz (*matrix*) o las variables indexadas (*array*). R también usa una clase especial de objeto para representar datos temporales, que son las *POSIXct* o *POSIXt*.

Un objeto de R 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 y quinto código 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 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 porque 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 países 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.

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()`.

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 ej., `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.

1.1.1. 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.

Otros tipos de vectores

Aparte de los cuatro que hemos mencionado, también hay vectores complejos, que pueden almacenar varios tipos de elementos, y vectores *raw* (brutos), que almacenan bytes *raw* de datos. Estos dos tipos de vectores no son necesarios en el análisis de datos en estudios internacionales.

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)`

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.

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.

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.

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.[1] 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.

[(1)]En R, el primer elemento en un vector tiene índice 1, no índice 0 como en otros lenguajes de programación, por ejemplo Python.

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

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 tres por el vector `c(3,4,5)` nos multiplicará el número tres 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.

1.1.2. 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.

Terminología de marco de datos

Hay varios términos empleados para denominar este mismo tipo de objeto. Lo podemos ver mencionado como hoja de datos, matriz de datos o marco de datos. Los dos primeros términos son fáciles de confundir con otros parecidos, por ejemplo hoja de cálculo o matriz (que es otro objeto de R). Para evitar confusiones, empleamos marco de datos, que es el término más similar al inglés *data frame*.

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 sabes, cada vector solo puede incluir un solo tipo de datos (numéricos, lógicos, caracteres...), pero en cambio un marco de datos puede incluir vectores de diferentes tipos.

Crear un marco de datos

Dentro de la función `data.frame()` hemos creado primero el vector *país*. 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.

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 podemos ver el código que hemos utilizado para crearlo.

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)
```

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 *país* 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, y también podemos comprobar cuál es la tipología de cada variable o vector (*chr* por carácter, *num* por numérico y *logi* por lógico).

[2]El primer vector es de carácter también, 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.

```
> str(md_agr)
'data.frame': 8 obs. of  5 variables:
 $ 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 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 que faltan, variables que faltan y tipos). En definitiva, tener un marco de datos en formato *tibble* es preferible a un marco de datos normal. Podemos probar de 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 ej., `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.

Todo este abanico de objetos que hemos estudiado hasta ahora no solamente tienen almacenado en su interior los datos propiamente dichos sino que también tienen 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.

1.2. 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 código siguiente: `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 intro-

ducimos 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 conocer 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] Las dos 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.

(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.

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 probabilidad que indica los pesos del vector x

Details

Por defecto, `size` tiene la misma longitud que x.

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 encontrar. 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 necesitamos refrescar cuáles son elementos principales de una función determinada. Por eso es muy importante prestar atención y saber 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 que estos dos elementos no van acompañados en Usage del símbolo igual 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 *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, 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()`, 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()`.

Apartados de ayuda

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.

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`

Función	Resultado
`sort(x = q)`	`9 6 5 3`

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, introducimos los argumentos con el mismo orden con 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 que 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 indicamos siempre los nombres de los argumentos acabaremos teniendo códigos más largos, pero tendremos la ventaja 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.

2. 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 recolectando 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.`

¿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,[4] 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).

[4]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.

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 al 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 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 hacemos sobre los datos. Como ya te habrás 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 `$` y finalmente el nombre de la variable[5] (ejemplo `marcodatos$variable`) .

[5] 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`.

2.1. Variables categóricas nominales

Las **variables categóricas nominales**, también dichas 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 país determinado es menor o mayor. Por lo tanto, con variables categóricas nominales podemos hacer pocas operaciones. 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.

Operaciones con variables categóricas nominales

Igual: `==`
No igual: `!=`

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

2.1.1. 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.

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"[6]. 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 ".

[6]Lo crearemos en formato *tibble* con este código: `eastg <- tibble(country = c("East Germany", "Germany", "France"))`. Recuerda que has de tener el paquete *dplyr* cargado.

```
cambios <- str_detect(eastg$country, "East Germany")
eastg$country[cambios] <- c("Germany")
str_replace(eastg$country, "East Germany", "Germany")
str_remove(eastg$country, "East ")
```

Fíjate que si tuviéramos otras categorías que también empezaran con "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 encontramos.

El paquete *janitor* como alternativa

Otra función muy recomendable para transformar vectores de carácter es `clean_names()` del paquete janitor, que limpia y homogeneiza los caracteres de manera casi automática. Por defecto, nos pondrá los caracteres en minúscula y la barra baja para separar palabras. Hay más opciones de transformación que puedes consultar en el cuadro de ayuda `?clean_names()`.

2.1.2. 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 entremedio de 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, suponemos 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 código siguiente. 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 como 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 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.

Pasar de factor a vector entero

Si queremos pasar el factor a entero tenemos dos posibilidades. Mediante la función `unclass()` nos eliminará el factor pero nos conservará las etiquetas, mientras que si utilizamos la función `as.integer()` nos eliminará también las etiquetas y conservará solo los números.

```
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)
```

Fíjate 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".[7]

[7]Hay que tener en cuenta que, aunque cada etiqueta corresponda a un número, 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 después de la función `levels()` la posición de los niveles.

```
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 es dentro mismo de la función `factor()`, donde indicaremos los nombres nuevos, por orden, en el argumento *labels*.

2.1.3. 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 dicha 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.

Paquete *forcats* para los factores

Cuando tengamos algún problema asociado con un factor es muy probable que lo podamos resolver con el paquete forcats. Por ejemplo, si queremos ordenar las categorías del factor según el número de frecuencias de cada una de ellas nos ayudarán las funciones `fct_infreq()` o `fct_rev()`. También podemos ordenar las categorías con `fct_relevel()`.

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.[8]

[8]Fíjate 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.

```
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$country == "Austria"` pedimos que nos devuelva un vector lógico cuyos valores sean verdad si el valor correspondiente a la variable *country* es igual en Austria y falso si no es igual. Como vemos, solo nos devuelve como TRUE el segundo valor y los otros nos los devuelve como FALSE. Con el código `pov$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 observación si el porcentaje de personas que viven por debajo de 1,90 dólares al día es inferior a 15.

Para saber más

Sobre la lógica de los operadores booleanos básicos, puedes mirar este vídeo (https://www.youtube.com/watch?v=6PpQS-YLWDQ&feature=youtu.be). Para una descripción más avanzada puedes consultar el apartado 5.2.2. de este manual (https://r4ds.had.co.nz/transform.html).

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 bien queremos que incluya o bien una palabra o bien 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 por ciento. 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 por ciento. 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
```

2.2. 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 quiere decir que las operaciones que podemos hacer con los valores es saber si son iguales, diferentes, más grandes o más pequeños.

Operaciones con variables categóricas ordinales

Igual: ==
No igual: !=
Mayor: >
Mayor o igual: >=
Menor: <
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 éstas 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 y 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.

2.2.1. 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.[9] Fíjate 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.

[9] 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.

```
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)

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.

2.3. 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 será inferior a un número positivo y el número 500 será superior al 450.

Operaciones con variables numéricas

Igual: `==`
No igual: `!=`
Mayor: `>`
Mayor o igual: `>=`
Menor: `<`
Menor o igual: `<=`
Sumar: `+`
Restar: `-`
Multiplicar: `*`
Dividir: `/`
Exponencial: `^`
Logaritmo: `log()`

Que tengan significado numérico también quiere decir 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 calcular la distancia entre valores. Esto nos permitirá hacer operaciones aritméticas como sumar, restar, multiplicar, dividir, sacar la raíz cuadrada o buscar el logaritmo neperiano.

Ejemplos de variables numéricas

La temperatura, la latitud, el porcentaje de tierra cultivable, la cantidad de ayuda oficial al desarrollo, la migración limpia, la tasa de crecimiento natural de la población o las emisiones de dióxido de carbono en 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 que 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.[10] A continuación mostramos dos maneras de crear un vector entero y ver el tipo de vector que ha creado.

[(10)]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)))
```

Como norma general, pues, 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 verdad o falso.
- En segundo lugar, asignaremos el valor que tendrá la nueva variable si la condición es verdad.
- 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(data$gdp >= 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(data$gdp, 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. Piensa en el porcentaje de superficie forestal de un país. La cifra podría ser, supongamos, un 15,69 por ciento. ¿Es un valor numérico continuo, puesto que puede aceptar decimales, pero que lo hace infinito en comparación con un valor numérico discreto? Encontraréis la respuesta en el cuadro de la derecha. Son variables numéricas continuas el PIB per cápita, la esperanza de vida, el tiempo medio de uso de Internet, la ayuda oficial al desarrollo recibida o el porcentaje de la población que tiene acceso a agua potable.

Infinidad

Al fin y al cabo, los valores numéricos se mueven en unos intervalos determinados. El PIB per cápita no puede ser negativo y es impensable que la esperanza de vida de una población sea de 300 años. ¿Qué hace, pues, que un valor sea infinito? Para entender porque las variables numéricas continuas son infinitas, pensemos en el espacio que hay entre el 15 por ciento y el 16 por ciento. Si lo pensamos bien, este espacio es infinito puesto que siempre podremos incluir cifras y cifras en la parte de los decimales. En cambio, en las variables numéricas discretas este espacio no existe. Entre 15 habitantes y 16 habitantes no hay ningún posible valor.

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 tres 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)`.

Resumen

En este módulo introductorio de R la voluntad ha sido empezar como una clase de idiomas. Para hablar con R tenemos que conocer la gramática básica y los elementos necesarios para construir frases y conseguir que el programa entienda lo que le comunicamos. Por eso hemos conocido los dos elementos imprescindibles para trabajar con R y cómo se relacionan entre sí: los objetos y las funciones. Hemos conocido y aprendido a crear los dos objetos principales que utiliza R en el análisis de datos en ciencias sociales: los vectores y los marcos de datos. Como analistas de datos, es evidente que nuestra función no será crear estos objetos sino que en la mayoría de los casos solo los tendremos que descargar de su fuente, normalmente en internet. Sin embargo, es importante saber crearlos para conocer su estructura.

El otro elemento imprescindible son las funciones. Quizás al principio nos sintamos abrumados por la gran cantidad de funciones que hay y la dificultad de aprender las tareas que hace cada una así como sus argumentos. A base de práctica y de consultar este u otros documentos iremos ampliando poco a poco nuestro vocabulario de funciones.

Finalmente, hemos aprendido cómo trasladar características de los fenómenos que queremos estudiar, las variables, a un objeto de R, los vectores. En la tabla 8 encontraréis un resumen de lo que hemos visto en la tercera sección de este módulo.

R trata diferente a los vectores diferentes

Para ver una primera diferencia de cómo R trata una variable según si está almacenada como vector, puedes intentar utilizar la función `summary()` para pedir un resumen de la variable. Pide primero un sumario de un vector de carácter de los que hemos tratado en este módulo, después de un vector y después de una variable numérica. Como podrás comprobar, la información que veremos cambiará según el tipo de vector que queramos resumir.

Tabla 8. Sumario de tipo de variables

Variable	Vectores	Ejemplo	Operaciones
Categórica nominal	Carácter, lógico, factor	`"Japan"`	`==, !=`
Categórica ordinal	Factor, lógico	`"Medium"`	`==, !=, >, >=, <, <=`
Numérica discreta	Entero, doble	`108`	`==, !=, >, >=, <, <=` `+, -, *, /, ^, etc.`
Numérica continua	Doble	`203445.39`	`==, !=, >, >=, <, <=` `+, -, *, /, ^, etc.`

Nuestro trabajo es ayudar a R para que tenga asociada cada variable a su vector correspondiente, puesto que no siempre es así. Cuando importamos bases de datos podemos encontrarnos con variables numéricas dentro de vectores de carácter, con variables ordinales que no están tratadas como factor ordinal o bien con variables categóricas nominales que se han guardado como factores y las queramos tener como vectores de carácter. Tener las variables asociadas al vector que corresponde nos facilitará mucho el trabajo en el análisis de los datos.

Ejercicios de autoevaluación

Para un mejor el 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) ¿De qué clase nos aparecerá este vector?

```
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) Indica el resultado de las operaciones siguientes:

```
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) Indica el resultado del código siguiente.

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

15) ¿Qué vector obtendremos en el código siguiente?

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

16) Indica el resultado lógico de la operación siguiente.

```
TRUE > FALSE
```

17) Indica el resultado lógico de la operación siguiente.

```
TRUE == FALSE
```

18) Indica el resultado lógico de la operación siguiente.

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

19) Indica el resultado lógico de la operación siguiente.

```
TRUE & FALSE
```

20) Indica el resultado lógico de la operación siguiente.

```
TRUE | FALSE
```

Solucionario

1) 20

2) 16

3) 9

4) 4 5 6 7 8

5) 25

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

Brancati, D. (2018). *Social Scientific Research.* Londres: Sage Publications Ltd.

George, A.; Bennett, A. (2005). *Case Studies and Theory Development in the Social Sciences*. Londres: MIT Press.

Goertz, G.; Mahoney, J. (2012). *A tale of two cultures: qualitative and quantitative research in the social sciences*. Princeton: Princeton University Press.

Grolemund, G.; Wickham, H. (2016). *R for Data Science*. Canadá: O'Reilly. https://r4ds.had.co.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*. https://cran.r-project.org/doc/contrib/r-intro-1.1.0-espanol.1.pdf