Anàlisi de dades amb R

PID_00265508

Jordi Mas Elias

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Jordi Mas Elias

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Jordi Mas Elias (2019)

Primera edició: setembre 2019

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

Introducció

L'objectiu d'aquest mòdul és introduir l'estudiant en l'anàlisi de dades per mitjà de mètodes quantitatius, orientats a l'explotació estadística de les dades, generalment en forma de nombres i categories quantificables, a partir d'un conjunt de tècniques estandarditzades. Aprendre a analitzar dades de forma quantitativa genera sovint certa ansietat als estudiants. Per això, aquest mòdul intenta oferir d'una manera pràctica els principals elements que l'alumne necessita per a iniciar-se en l'anàlisi de dades. L'estructura del mòdul està pensada perquè l'estudiant aprengui primer a fer operacions senzilles i acabi sent capaç de moure's amb relativa facilitat entre bases de dades d'una mida considerable.

Per a saber-ne més

Per una visió general dels diferents mètodes utilitzats en ciències socials, vegeu Bennett i George (2005, capítol 1), Brancati (2018, capítol 7), Goertz i Mahoney (2013, capítols 1 i 2), Halperin i Heath (2016, capítols 5 i 6).

Per qüestions pràctiques, l'anàlisi de dades comporta necessàriament la utilització de programari informàtic per tal d'emprar de manera àgil i ràpida operacions complexes. Aquest mòdul utilitza l'entorn de programació R per diversos motius:

1) És un programari lliure accessible per a tothom que té un dels llenguatges més populars en ciència de dades.

2) Entre els programaris lliures, R és el més popular en ciències socials, està especialment pensat per a l'autoaprenentatge i resulta més intuïtiu que altres programes com Python.

3) Té una comunitat d'usuaris molt activa que permet resoldre els dubtes de funcionament que es puguin tenir.

Per a saber-ne més

Vegeu, per exemple, Stack Overflow (https://stackoverflow.com/), la comunitat d'usuaris d'RStudio (https://community.rstudio.com/) o la de DataCamp (https://www.datacamp.com/community), on podeu accedir a tutorials i guies del programa. Pel que fa als manuals d'R, trobareu en castellà un manual una mica antic de R Development Core Team (2000). També podeu consultar Grolemund i Wickham (2016).

4) És un programa que treballa amb codi. Malgrat que això pot representar un obstacle inicial per a l'aprenentatge, el fet d'utilitzar un llenguatge de codi comporta grans beneficis a mig termini per a l'estudiant i les seves habilitats de recerca. Aprendre un llenguatge de programació és una habilitat cada vegada més important en qualsevol àmbit professional.

Els beneficis de treballar en codi

Imaginem-nos que descarreguem una base de dades d'internet, fem un anàlisi de dades i presentem en públic aquestes dades amb unes diapositives. Què ha passat entremig? Ningú no sap què hem fet amb les dades en el nostre petit laboratori. S'assumeix que hem estat honestos i no hem fet manipulacions sospitoses perquè quadri allò que volíem demostrar. El codi permet tenir un registre de tots els nostres passos, la qual cosa facilita la reproducció de la recerca. Tot allò que hem generat a partir d'unes dades ho podem transferir a altres persones perquè puguin obtenir els mateixos resultats als quals hem arribat. Això millora la transparència de les nostres anàlisis i el rigor dels nostres resultats.

Aquest mòdul explica el llenguatge bàsic d'R que necessitem aprendre i el tipus de variables que hem d'utilitzar per a l'anàlisi de dades. D'entrada, el mòdul s'assumeix que l'estudiant ja té un domini mínim inicial d'R: hi ha accedit alguna vegada, coneix la utilitat de cada element de la interfície i sap com reproduir codis a dins el programa. Les línies de codi que apareixen en les pàgines següents estan pensades per a ser copiades i enganxades a RStudio per tal que l'estudiant les pugui reproduir en el programa. Normalment, el codi es mostra a dins d'una caixa gris. Quan, dins de la caixa gris, apareixen encapçalats pel símbol > significa que a dins mateix de la caixa mostrem imprès el resultat del codi. Per a aclarir una distinció terminològica, ens referim a una base de dades com una ubicació genèrica per a emmagatzemar dades. En canvi, ens referim a un marc de dades com una manera específica que R utilitza per a emmagatzemar dades.

Per a saber-ne més

Per refrescar alguns conceptes bàsics, vegeu el Cheat sheet IDE (https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf) o consulteu altres recursos que us proporcionaran en aquest curs.

1. El llenguatge d'R

Per a aprendre a utilitzar programes d'anàlisi de dades com R, Python, Stata o SPSS, el més important que ens cal inicialment és dominar-ne el llenguatge. En general, dominar un llenguatge de programació equival a dominar un idioma. Si compartim el mateix idioma, podrem comunicar-nos amb el programa i fer que aquest entengui les nostres ordres. Passa el mateix quan volem aprendre alemany: haurem de dominar la gramàtica alemanya, les seves paraules bàsiques i saber com es conjuguen els verbs. A més, tard o d'hora, haurem de ser capaços de defensar-nos oralment i tenir una bona pronunciació.

Amb llenguatges de programació com R, la sort que tenim és que són llenguatges escrits i no pas orals. Per tant, això ens estalvia una part important de l'aprenentatge, ja que només haurem d'aprendre la part escrita. L'entorn en què s'utilitza el llenguatge d'R és també molt limitat. No ens cal usar-lo ni a l'aeroport, ni al supermercat, ni per a presentar-nos a altres persones, tal com passa en la majoria d'idiomes, que els hem d'aprendre a usar en una gran quantitat de contextos. Aquests llenguatges només són necessaris en situacions molt concretes, de manera que no ens cal conèixer gaires paraules ja que constantment repetim les mateixes idees, com ara:

«filtra les columnes d'una base de dades»
«construeix un gràfic a partir d'aquestes variables»
«suma aquests dos valors»
«digues quina és la mitjana dels valors d'aquesta variable»...

Si sabem anomenar els principals objectes de què disposem (variable, base de dades, valor...) i els principals verbs que utilitzarem (filtrar, ordenar, crear gràfic), i tenim els coneixements gramaticals necessaris per a construir frases amb sentit, R ens entendrà i podrem fer grans virgueries amb les dades.

Com li parlem, doncs, a R? Sempre acostumem a dir "[R] fes això", " [R] fes allò". Per tant, totes les ordres ja tenen implícites el subjecte, ja que és R qui ho fa. I això és un altre avantatge. Si, en la majoria d'idiomes, les frases acostumen a tenir subjecte i predicat, en la gramàtica d'R no ens caldrà subjecte perquè R ja entén qui ha de fer l'acció. El que sí que ens cal és el predicat, per la qual cosa en el llenguatge d'R hi ha sobretot verbs, adverbis i complements del verb:

1) als verbs els anomenarem funcions,
2) als adverbis els anomenarem arguments,
3) als complements del verb els anomenarem objectes i
4) a la informació complementària associada als objectes els anomenarem atributs.

A continuació detallem primer els objectes i els atributs, i en segon lloc les funcions i els arguments.

1.1. Objectes i atributs

Per **objecte** ens referim a qualsevol dada que tinguem guardada dins d'R, mentre que per **atribut** entenem la informació complementària associada a aquestes dades.

Els objectes d'R

Hi ha altres tipus d'objectes que no estudiarem. Els més comuns són la llista, la matriu (*matrix*) o les variables indexades (*array*). R també fa servir una classe especial d'objecte per a representar dades temporals, que són les *POSIXct* o *POSIXt*.

Un objecte d'R pot prendre la forma d'un nombre, d'una cadena de valors o d'un marc de dades, entre altres tipus. Per a crear un objecte hem de fer servir el símbol `<-`, on en primer lloc posem el nom que tindrà l'objecte, seguit de `<-` i finalment la forma de l'objecte. En el quadre que veiem a continuació, hem generat diversos codis per a crear objectes. Primer, hem creat l'objecte `tres`, que està format pel número tres. En segon lloc, hem demanat a R que ens guardi, amb el nom d'`operacio`, el resultat de `(6 + 4) / 2`. El mateix li demanem amb `operacio_nova`, on ens multiplica per tres l'objecte `operacio` que hem creat fa un moment. El quart i cinquè codi ens mostren objectes una mica més sofisticats, que estudiarem de seguida. L'objecte `paisos` conté un tipus d'objecte anomenat vector, de longitud cinc, ja que està format per una cadena de cinc valors que contenen noms de diversos països. Hi ha diversos tipus de vectors. A aquest l'anomenarem *vector de caràcter*, ja que en lloc de guardar nombres com en els exemples anteriors, guarda caràcters, que sempre aniran separats per cometes. Finalment, el darrer objecte que veiem és un marc de dades (*data frame*), que hem anomenat `md_hdi`. El marc de dades es caracteritza perquè agrupa diversos vectors de la mateixa longitud. Per a conèixer la longitud d'un vector hem d'utilitzar la funció `length()`. Per exemple, `length(paisos)` ens mostrarà la longitud del vector països que acabem de crear. El primer vector `pais` és un vector de caràcter, el segon i el tercer, `pnb` i `e_vida`, són vectors numèrics, mentre que el darrer vector `dem` és un vector lògic, que pot adoptar els valors veritat (`TRUE`) o fals (`FALSE`).

```
tres <- 3
operacio <- (6 + 4) / 2
operacio_nova <- operacio * 3
paisos <- c("Alemanya", "Argentina", "Espanya", "Marroc", "Sudan")
md_hdi <- data.frame(pais = c("Alemanya", "Argentina", "Espanya", "Marroc", "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))
```

Els vectors i els marcs de dades seran els dos principals objectes que utilitzarem amb R. Com observem en el marc de dades `md_hdi` que acabem de crear, tots els vectors que el conformen tenen la mateixa longitud. En aquest cas, és un marc de dades format per quatre columnes (el nombre de vectors) i cinc observacions (la longitud dels vectors).

Abans de repassar més a fons aquests dos tipus d'objectes, hem d'assenyalar dues consideracions relacionades amb la seva creació:

1) Hem de tenir en compte que quan creem un objecte, la consola no ens donarà cap senyal que l'haguem creat, ja que l'únic que fem és emmagatzemar l'objecte en la memòria. Si volem visualitzar l'objecte un cop creat, el que hem de fer és teclejar-ne el nom. Si, per contra, volem visualitzar l'objecte a la vegada que el creem podem posar tota la línia de comandament entre parèntesis. En aquest cas, R no només ens crearà l'objecte `tres` sinó que també ens reproduirà el seu contingut a la consola.

Visualitzar objectes a la consola

Si teclegem `md_hdi` després de guardar-lo amb aquest nom, podrem veure l'objecte imprès a la consola. Per a visualitzar l'objecte a la vegada que el creem, l'hem de posar entre parèntesis, com per exemple `(tres <- 3)`. Moltes vegades utilitzarem el terme *imprimir l'objecte*, que significarà teclejar-ne el nom per a visualitzar-lo a la consola.

2) Hem de saber que quan guardem un objecte, aquest ens apareixerà al panell Environment d'RStudio acompanyat d'una descripció breu. Podem visualitzar els objectes directament al panell Environment o bé també podem consultar un llistat dels objectes creats teclejant indistintament les funcions `ls()` o `objects()`.

Posar nom a un objecte

A un objecte li podem donar gairebé quasi qualsevol nom que vulguem. Les úniques limitacions són que no pot començar amb una xifra (per ex., `1objecte <- 34`) ni tampoc pot contenir cap dels símbols següents: ^, !, $, @, +, -, /, *. Si anomenem un objecte amb el nom d'un altre objecte creat prèviament, R ens sobreescriurà l'objecte sense avisar-nos prèviament. R és sensible a les minúscules i a les majúscules, de manera que entendrà `Casa` i `casa` com dos objectes diferents. Recomanem utilitzar minúscules sempre que es pugui i fer servir la barra baixa en cas de voler separar paraules per a anomenar un mateix objecte.

1.1.1. Vectors: objectes d'una dimensió

Un **vector** és una cadena de valors, ordenats en una sola dimensió, que pot tenir una longitud diversa, des d'un sol valor fins a milers de valors.

Anteriorment hem creat un petit marc de dades, `md_hdi`, en què cada vector és una variable diferent. La majoria d'objectes d'R s'organitzen a partir de vectors, i per tant, tenir molt clar què és un vector, quina funció fa i quins tipus de vectors podem crear amb R ens facilitarà molt la feina com a analistes.

El **vector** és l'estructura bàsica d'R i la forma que, en R, pren una variable dins un marc de dades. En la taula 1 hem creat diversos vectors i hem assignat un nom diferent a cada un d'ells. Els vectors poden emmagatzemar fins a sis tipus diferents de dades, encara que en aquest només n'utilitzarem quatre: numèric, enter, caràcter i lògic.

Altres tipus de vectors

A part dels quatre que hem esmentat, també hi ha vectors complexos, que poden emmagatzemar diversos tipus d'elements, i vectors crus, que emmagatzemen bytes crus de dades. Aquests dos tipus de vectors no són necessaris en l'anàlisi de dades en estudis internacionals.

Taula 1. Tipus de dades que pot emmagatzemar un vector

Numèric o doble	`vector_numeric <- c(78.2, 56.3, 72.4, 64.6, 84.1)`
Enter	`vector_enter <- c(1L, 5L, 7L, 4L, 4L, 4L, 7L, 8L)`
Caràcter o *string*	`vector_caracter <- c("blau", "groc", "verd", "blau")`

Lògic	`vector_logic <- c(TRUE, FALSE, FALSE, FALSE, TRUE)`

Per a crear un vector que conté més d'un valor, posarem els valors entre parèntesis encapçalats per la funció `c()`, que és una abreviació de la paraula concatenat. A dins de la funció, hi introduirem els valors corresponents segons el tipus de vector que vulguem crear. El vector numèric i l'enter tenen una aparença molt semblant. Els dos emmagatzemen nombres, però mentre que el vector numèric accepta decimals, l'enter (*integer* en anglès) només accepta nombres enters. Per defecte, R ens emmagatzemarà qualsevol nombre com a vector numèric, tingui decimals o no. Si volem que ens el guardi com a enter, ho haurem d'especificar posant una L majúscula al davant o transformant el vector numèric a enter. Ara no podem apreciar gaire bé la diferència pràctica entre numèric i enter, però més endavant en aquest mòdul veurem com aquesta distinció ens serà molt útil.

La L del vector enter

Encara que introduïm una L majúscula quan creem un vector enter, R no ens visualitzarà la L. Només entendrà que aquest vector l'ha de guardar com a enter. Una opció més fàcil per a crear un vector enter és crear un vector numèric sense decimals de la manera següent: `as.integer(c(3, 5, 1, 7))`.

El vector de caràcter, que veureu molt sovint amb el nom d'*string*, emmagatzema text. Aquest text ha d'anar sempre entre cometes i a dins de les cometes podem guardar lletres, nombres, caràcters especials, etc. Finalment, el vector lògic (també dit vector Booleà) ens guarda valors que poden ser veritat o fals. Els valors que són veritat els emmagatzemem com a `TRUE` o senzillament amb una `T`, mentre que els que són falsos els guardarem com a `FALSE` o amb una `F` (sempre indicat en lletres majúscules). És important saber que en un mateix vector no podem barrejar valors que continguin diferents tipus de dades. Si ho provem, R guardarà les dades amb el tipus de vector que permeti conservar la quantitat més gran d'informació possible.

Vectors de caràcter que no ho semblen

Un valor en un vector de caràcter; podria ser "22" o bé "·= %/". Només val que vagi indicat entre cometes.

Exercici 1. Coerció de les dades

Proveu de fer l'exercici següent per a esbrinar com R tria el vector amb què guardarà la informació en cas que s'hagin introduït diversos tipus de valors. Primer, emmagatzemeu cada un dels vectors i després mireu quin tipus de vector heu creat amb la funció `class()`.

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

Què descobrim amb aquest exercici? Si intentem posar diferents tipus de dades en un sol vector, R convertirà els elements del vector en valors d'un sol tipus. Com decideix R aquesta conversió? R sempre intentarà preservar el màxim de dades possibles i la manera com menys informació es pot perdre és en un vector de caràcter. Dins de les cometes podem guardar tot el que vulguem. En canvi, en un vector numèric no hi podem emmagatzemar lletres. Els caràcters, doncs, sempre tindran prioritat sobre els altres tipus de vector en cas de conflicte. Entre numèrics i lògics, R guardarà un vector numèric, ja que sempre podem interpretar TRUE com 1 i FALSE com 0. Entre numèrics i enters, R guardarà numèrics.

Per a treballar amb vectors, siguin del tipus que siguin, ens serà molt útil aprendre a seleccionar només una part dels seus elements. La manera com podem seleccionar-los és mitjançant claudàtors `[ ]` després del nom del vector. A dins dels claudàtors, hi indicarem la posició dels valors que volem seleccionar. El símbol `:` ens servirà per a indicar una selecció en cadena, en què a l'esquerra

[(1)] En el programari R, el primer element en un vector té índex 1, no pas índex 0 com en altres llenguatges de programació, com ara Python.

del símbol posarem la posició del primer valor i a la dreta la del darrer valor.[1]El símbol negatiu exclou una selecció. Seguint aquesta lògica, a la taula 2 podem veure exemples de com funciona la selecció d'elements d'un hipotètic vector.

Taula 2. Selecció d'elements d'un vector

`vector[1:3]`	Del primer al tercer valor
`vector[c(2,4)]`	El segon i el quart valor del vector
`vector[c(1:3,6)]`	Del primer al tercer valor i el sisè valor del vector
`vector[-c(5,8)]`	Tots els valors del vector menys el cinquè i el vuitè

A l'hora de fer operacions matemàtiques, cal tenir en compte que si fem una operació d'un nombre sobre un vector, l'operació s'aplicarà a tots els valors d'aquest vector. Si, en canvi, multipliquem vectors d'igual longitud, multiplicarà el primer valor del primer vector pel primer valor del segon vector, el segon valor del primer vector pel segon valor del segon vector, i així successivament. Si fem operacions amb vectors de longitud diferent a 1 i a la longitud del vector ens donarà error.

Exemple

Si multipliquem el número tres pel vector `c(3,4,5)` ens multiplicarà el número tres per cada un dels números de dins el vector i donarà com a resultat 9, 12 i 15. En canvi, si multipliquem `c(3,4,5)` per `c(3,4,5)` donarà 9, 16 i 25.

1.1.2. Marcs de dades: objectes de dues dimensions

Un **marc de dades** és un conjunt de vectors d'igual longitud agrupats en una taula de dues dimensions formada per columnes i files. En la dimensió de les files, cada element representa una observació de les dades. En la dimensió de les columnes, cada columna està formada per un vector.

Terminologia de marc de dades

Hi ha diversos termes emprats per a denominar aquest mateix tipus d'objecte. El podem veure esmentat com a full de dades, matriu de dades o marc de dades. Els dos primers termes són fàcils de confondre amb d'altres de similars, com ara full de càlcul o matriu (que és un altre objecte d'R). Per a evitar confusions, emprem marc de dades, que és el terme més similar a l'anglès *data frame*.

El marc de dades és l'estructura principal que farem servir per a l'anàlisi de dades. Abans d'iniciar-nos amb R, haurem vist estructures semblants en fulls de càlcul d'Excel o en altres programes informàtics. Acostumem a dir-ne *taula*,

base de dades. Com ja sabeu, cada vector només pot incloure un sol tipus de dades (numèriques, lògiques, caràcters...), però en canvi un marc de dades pot incloure vectors de diferents tipus.

Per a crear un marc de dades hem d'utilitzar la funció `data.frame()` i introduir els vectors que volem que incorpori, separats per comes, en l'ordre en què els visualitzarem, d'esquerra a dreta. Abans de cada vector podem especificar el nom que tindrà la columna seguit del símbol igual. En la taula 3, tenim representat el marc de dades `md_agr` i a la part inferior podem veure el codi que hem utilitzat per a crear-lo.

Crear un marc de dades

A dins de la funció `data.frame()` hem creat primer el vector *país*. Com que repetim dues observacions per a cada país, una l'any 1980 i l'altra el 2010, hem de repetir dues vegades el nom del mateix país. Recordeu que heu de posar primer el nom de la columna, seguit d'un igual i un concatenat amb els valors corresponents, i sobretot que, per passar al vector següent, cal tancar parèntesi després d'introduir tots els valors de cada vector.

Taula 3. Creació d'un marc de dades

pais	any	dem	pib_cap	agr
<caràcter>	<numèric>	<lògic>	<numèric>	<numèric>
"França"	1980	TRUE	12672.18	56.8
"França"	2010	TRUE	40638.33	52.8
"Regne Unit"	1980	TRUE	10032.06	76.8
"Regne Unit"	2010	TRUE	38893.02	71.2
"Polònia"	1980	FALSE	5241.75	64.5
"Polònia"	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("França", "França", "Regne Unit", "Regne Unit", "Polònia",
"Polònia", "Congo", "Congo"), any = 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 imprimim el marc de dades que acabem de crear teclejant `md_agr`, veurem que consta de cinc columnes corresponents a cada un dels vectors que hem creat. El vector *pais* de la primera columna és de caràcter, ja que R ha entès que si els valors estan entre cometes haurà de crear un vector de caràcter.[2] R ha interpretat els vectors *any*, *pib_cap* i *agr* (que representa el percentatge de terra conreable) com a variables numèriques, mentre que ha interpretat el vector *dem* (democràcia) com a vector lògic. Podem veure algunes característiques addicionals del marc de dades si demanem l'estructura amb `str(md_agr)`. La funció `str()` ens permet observar que es tracta d'un marc de dades amb cinc variables o vectors i vuit observacions, i també podem comprovar quina és la tipologia de cada variable o vector (*chr* per caràcter, *num* per numèric i *logi* per lògic).

[2]El primer vector és de caràcter també perquè hem introduït l'argument `stringsAsFactors = FALSE` al final del codi. Sempre haurem de posar aquest argument si volem que ens mantingui els vectors entre cometes com a caràcters. Més endavant en aquest mòdul veurem la diferència entre vectors *strings* i vectors factors.

```
> str(md_agr)
'data.frame': 8 obs. of  5 variables:
 $ pais   : chr  "França" "França" "Regne Unit" "Regne Unit" ...
 $ any    : 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 l'hora de treballar amb els marcs de dades d'R hem de tenir en compte dues consideracions importants. Si us fixeu, quan imprimim el marc de dades, les observacions són a les files i els vectors a les columnes. En canvi, quan imprimim l'estructura amb `str()` ho trobem al revés: els noms del vector estan a les files mentre que les observacions estan desplegades en horitzontal. Això és així per qüestions pràctiques: normalment treballarem amb marcs de dades enormes, amb milers de files, però en canvi amb pocs vectors. Això fa que ens sigui més còmode imprimir l'estructura per a fer una visualització ràpida de la consola, fer una ullada als noms de les variables i el tipus, i veure una mostra de les dades que conté la variable. Per a visualitzar-ho és més intuïtiu desplaçar-se per la consola en vertical que no pas en horitzontal. La segona consideració, relacionada amb la primera, és que, a diferència d'altres programes informàtics, ens haurem d'acostumar a treballar amb les dades però sense les dades. És a dir, el marc de dades serà al nostre cap però no a la pantalla. Com a analistes de dades veureu que acabarà sent una manera més eficient de treballar, ja que no hi ha cap necessitat de tenir les dades visualment disponibles quan treballem amb bases de dades de grans dimensions.

Visualització clàssica d'una taula

La funció `View()` ens permet visualitzar tot un marc de dades de manera clàssica, però avisem que en pocs dies no la trobarem a faltar. En aquest mòdul aprendrem a treballar sense necessitat de tenir disponibles totes les dades en pantalla.

Tibble: La nova generació de marcs de dades

Un *tibble* és un tipus especial de marc de dades, que ofereix, entre altres avantatges, millors visualitzacions a la consola que un marc de dades normal. El *tibble* adapta el marc de dades a la consola, de manera que segons l'amplada que tingui la pantalla del nostre ordinador podrem visualitzar més o menys observacions. Tot allò que no ens pugui ensenyar ens ho mostrarà de manera resumida al final (files que falten, variables que falten i tipus). En definitiva, tenir un marc de dades en format *tibble* és preferible a un marc de dades normal. Podem provar de canviar el marc de dades `md` a *tibble* indistintament mitjançant la funció `tibble()` o `tbl_df()`, que es troba dins el paquet de *dplyr*. La funció `as.data.frame()` fa l'operació inversa i canvia el *tibble* a un marc de dades normal.

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

Un altre dels grans avantatges de *tibble* és que ens permet crear marcs de dades de manera horitzontal amb la funció `tribble()`. Com veiem en l'exemple següent, amb *tribble* podem entrar els marcs de dades per files enlloc de per columnes, de manera que ens permet visualitzar el codi més intuïtivament.

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

Per seleccionar només una part del marc de dades també ens poden ajudar els claudàtors. La única diferència en relació amb la selecció de vectors que hem vist anteriorment és que en el cas dels marcs de dades hi ha una coma a dins dels claudàtors que separa les indicacions de la posició de les files de les indicacions de la posició de les columnes que volem seleccionar (per ex., `md[files,columnes]`). Si deixem un cantó de la coma en blanc, R entendrà que volem conservar totes les files o les columnes. Si introduïm un símbol negatiu, R entendrà que volem tota la selecció excepte el valor o valors indicats. Il·lustrem aquestes característiques en la taula 4.

Taula 4. Selecció de files i columnes d'un marc de dades

`md[3, ]`	La tercera fila i totes les columnes
`md[2:4,c(1,4)]`	De la segona a la quarta fila i la primera i quarta columna
`md[ ,-4]`	Totes les files i totes les columnes excepte la quarta
`md[-c(2:4,6), "any"]`	Totes les files excepte de la segona a la quarta i la sisena, la columna amb nom "any". Ens retornarà un vector.

Tot aquest ventall d'objectes que hem estudiat fins ara no solament tenen emmagatzemat en el seu interior les dades pròpiament dites sinó que també hi tenen lligada informació addicional. D'aquesta informació en diem **atributs**, que podem conèixer per mitjà de la funció `attributes()`. Si quan demanem els atributs a la consola R ens retorna NULL voldrà dir que l'objecte no té cap atribut. Si en té, R ens retornarà una llista amb diferents vectors per a cada tipus d'atribut.

Observem, per exemple, els atributs del marc de dades `md_agr`. En primer lloc, veiem que els noms de les columnes són atributs. És a dir, no formen part directament dels valors del marc de dades però sí que són informació addicional sobre el marc. Els noms de les columnes també els podem obtenir amb la funció `names()`, que ens serà de gran utilitat quan vulguem canviar els noms de columna. Com mostra el codi següent, podem canviar tots els noms de columna del marc de dades `md_agr` si creem un vector de la mateixa longitud a la quantitat de columnes i l'inserim en els noms del marc de dades. Si només ens interessa canviar una sola columna, podem seleccionar el número de columna amb els claudàtors i fer la mateixa operació.

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

Altres atributs que trobarem lligats als objectes són la classe d'objecte, que podem obtenir directament amb la funció `class()`. Aquesta funció és molt útil per a determinar el tipus de vector amb què estem treballant. La funció `typeof()` ens retorna una descripció més genèrica del tipus d'objecte. Finalment, també podem obtenir el nom de les files amb `row.names()`, encara que és una funció que utilitzarem en comptades ocasions.

1.2. Funcions i arguments

Les **funcions** són els verbs d'R i ens permeten fer operacions amb els seus objectes.

En l'apartat anterior ja hem pogut veure algunes funcions, com `data.frame()` o `str()`, que fan accions com crear un marc de dades o visualitzar l'estructura del marc. R té milers de funcions que fan operacions molt

diverses: des de tasques senzilles com arrodonir un nombre amb decimals (`round()`), sumar els valors d'un vector (`sum()`) o dir-nos quin és el valor més alt d'un vector (`max()`). Utilitzar aquest tipus de funcions és molt senzill. Només cal posar a dins del parèntesi l'objecte sobre el qual volem executar la funció. Mitjançant aquestes funcions, en el primer cas hem arrodonit els nombres del `vector_numeric` que hem creat anteriorment, en el segon cas hem sumat els nombres d'un vector i en l'últim hem buscat el nombre màxim del `vector_enter`.

```
round(vector_numeric)
sum(c(5, 1, 9, 8, 2))
max(vector_enter)
```

Exercici 2. Funcions

Amb les funcions que coneixeu, proveu d'arrodonir la divisió de 17 entre 3. Feu la suma d'una cadena de vectors entre l'1 i el 10. Busqueu el nombre més alt del `vector_numeric`. Finalment, primer imprimiu a la consola el número `pi` que R ja té guardat per defecte i després arrodoniu-lo.

Les funcions també poden operar a dins d'altres funcions. És a dir, la funció operarà a partir del resultat d'una funció que ha operat prèviament. Posem pel cas que introduïm el codi següent: `round(sum(vector_numeric))`. R primer ens sumarà tots els valors del vector numèric i a continuació ens arrodonirà el resultat, traient-li els decimals.

A dins de les funcions hi ha arguments, que serien els complements del verb. Un **argument** és un element que la funció necessita per a desenvolupar una acció.

Les funcions poden tenir més d'un argument i sempre aniran separades per comes a dins del parèntesi de la funció. En molts casos, només necessitarem un argument per a fer l'acció. En les funcions del quadre anterior, per exemple, n'hi ha hagut prou a situar un objecte dins del parèntesi. En altres ocasions, però, haurem d'incloure més d'un argument per a fer una acció. Fixem-nos en la funció `sample()`, que ens permet obtenir una mostra aleatòria d'un conjunt de dades. Si l'únic argument que introduïm a dins de la funció és un nombre, R ens retornarà aleatòriament un vector de nombres enters diferents amb longitud igual al nombre que hem indicat. Si introduïm un vector com `vector_numeric`, la funció ens retornarà els elements del vector ordenats de manera aleatòria.

Exercici 3. La funció `sample()`

Proveu de repetir les funcions del quadre següent més d'una vegada. Veureu que cada vegada que executeu l'acció us retorna les dades amb ordre diferent.

```
sample(10)
sample(vector_numeric)
sample(10, 3)
```

```
sample(vector_numeric, 2)
```

Provem ara d'introduir un segon argument. En el tercer exemple del quadre, li diem a R que agafi els nombres enters de l'1 al 10 i en seleccioni només tres aleatòriament. El mateix podem fer amb els valors de `vector_numeric`. R ens retornarà dos valors a l'atzar del vector.

Cada funció, doncs, pot tenir diversos arguments. Normalment, com en el cas de `sample()`, el primer argument serà l'objecte sobre el qual volem que s'apliqui la funció. Molts altres arguments poden funcionar de manera implícita, ja que cada funció acostuma a estar dissenyada amb arguments per defecte, que operen sense que els veiem. Per exemple, `sample()` té com a segon argument implícit que ens retorni un vector de longitud igual a l'objecte del primer argument.

Cada funció té els seus propis arguments i per tant, per poder treure el màxim rendiment d'una funció, cal conèixer-ne l'estructura interna. La manera més directa de conèixer els arguments que té una funció determinada és aplicar `args()`.

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

Si volem conèixer els arguments d'una manera més extensa podem posar el signe d'interrogació `?` abans de la funció o bé podem utilitzar la funció `help()`.[3] Les dues opcions obren una fitxa d'informació sobre la funció en la pestanya d'ajuda situada a la part inferior dreta d'RStudio. En la taula 5 no reproduïm tota la informació que apareix en la fitxa d'informació, però sí que en descrivim els elements més rellevants.

[3] També tenim el doble interrogant `??`, que, situat al davant de qualsevol nom, ens fa una cerca per la paraula clau que indiquem i ens reprodueix un llistat al quadre d'ajuda amb els resultats més rellevants que ha trobat.

Feu servir constantment el quadre d'ajuda

Fer servir el quadre d'ajuda és molt útil per a obtenir més informació o per a ajudar-nos a desencallar algun problema que puguem trobar. És impossible memoritzar tots els arguments que té cada funció, per la qual cosa ens serà molt útil consultar el quadre d'ajuda cada vegada que necessitem refrescar quins són elements principals d'una funció determinada. Per això és molt important parar atenció i saber entendre les fitxes d'ajuda.

Taula 5. Quadre d'ajuda resumit de la funció *sample*

```
> ?sample()
> help(sample)
```

Usage	
`sample(x, size, replace = FALSE, prob = NULL)`	
Arguments	
`x`	vector d'un o més elements
`size`	número no-negatiu d'items que seleccionarà de la mostra
`replace`	si torna a incloure a la mostra els números que ja han sortit
`prob`	vector de probabilitat que indica els pesos del vector x
Details	
Per defecte, size té la mateixa longitud que x.	

Com veiem a la taula, l'apartat Usage ens mostra els arguments principals de la funció `sample()`: *x*, *size*, *replace* i *prob*. Si volem saber-ne més detalls, podem anar a l'apartat Arguments, que ens ofereix una descripció de cada argument. A *x* hem de posar l'objecte en qüestió i a *size* el número no-negatiu d'ítems que seleccionarà de la mostra. Fixem-nos que aquests dos elements no estan acompanyats a Usage del símbol igual i un altre paràmetre, però sí que van acompanyats els arguments *replace* i *prob*. Això vol dir que *replace* i *prob* tenen assignat un valor per defecte. En altres paraules, si no indiquem el contrari, R entendrà que *replace* és fals (per tant, no tornarà a incloure el número en el bombo un cop hagi sortit) i *prob* és nul (tots els elements tenen la mateixa probabilitat de sortir seleccionats). A Details, veiem que *size* també té un valor assignat per defecte, però aquest no és fix, sinó que depèn del valor d'*x*. Això vol dir que en el cas que només introduïm un sol argument en la funció `sample()`, R ens retornarà tants valors com elements hi hagi en el valor en qüestió. En cas que volem que ens retorni un nombre diferent de valors ho haurem d'indicar expressament. Al principi ens costarà entendre aquestes pàgines d'ajuda però de mica en mica, a mesura que ens anem familiaritzant amb la lògica de les funcions, ens resultaran cada vegada més útils i indispensables.

Mateix nom de funció, diferents paquets

Ens podem trobar, com en el cas de la funció `sample()`, que diversos paquets tinguin una funció amb el mateix nom. Per especificar que volem la funció d'un paquet concret podem utilitzar el símbol `::`. Per exemple, `base::sample()`.

Apartats d'ajuda

Hi ha altres apartats a l'ajuda que no hem mirat tant en detall, com *Description* (breu sumari del que fa la funció), *Details* (descripció avançada de com està programada la funció) o *See also* (ens llista funcions similars a la que estem inspeccionant). L'últim apartat, *Examples*, ens pot servir de gran ajuda per a fer-nos una idea de com operen les funcions en qüestió amb codis reals.

Exercici 4. Consulteu els arguments d'altres funcions

Proveu d'investigar les funcions `seq()`, `rep()` i `sort()` mitjançant el quadre d'ajuda. Intenteu construir un codi per a cada funció per tal que us retorni a la consola el resultat que s'indica a continuació. En les dues últimes funcions haurem d'indicar, com a argument *x*, l'objecte `q <- c(3, 9, 5, 6)`, i intentar esbrinar la resta d'arguments.

Funció	Resultat
`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`

Un darrer aspecte important que cal tenir en compte quan especifiquem els arguments dins d'una funció és l'ordre amb què els posem. R espera que, si no indiquem el contrari, introduïm els arguments amb el mateix ordre amb què ens els mostra en el quadre d'ajuda. És a dir, com en la primera línia del codi següent, ens llegirà la funció de manera que `vector_numeric` correspon a la *x*, 2 correspon a *size*, TRUE correspon a *replace* i el vector que hem creat, que és igual a la longitud del vector indicat a *x*, assenyala les probabilitats que tindrà cada element de *x* de sortir elegit. També podem posar els noms dels arguments com veiem en la segona línia de codi, però si seguim l'ordre predeterminat no seria necessari introduir-los. De la mateixa manera, també podem alterar l'ordre dels arguments, però en aquest cas sí que serà necessari indicar a R quin és cada argument.

```
sample(vector_numeric, 2, TRUE, c(0.3, 0.1, 0.3, 0.2, 0.1))
sample(x = vector_numeric, 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_numeric)
```

Si indiquem sempre els noms dels arguments acabarem tenint codis més llargs. Això pot ser vist com un inconvenient, ja que podem preferir codis simples que continguin la informació el més simplificada possible. No obstant, un codi llarg també té l'avantatge que serà més fàcil i ràpid llegir i interpretar, tant per a nosaltres com per a terceres persones. Indicar el nom dels arguments ajuda a detectar i prevenir errors.

2. Variables i tipus

Una **variable** és una característica del fenomen que estudiem que varia entre els casos i, per tant, pot prendre diversos valors.

Suposem que tenim una base de dades de tots els països del món i estem recol·lectant diverses característiques d'aquests països. Què podria ser i què no podria ser una variable en aquest estudi? Un color qualsevol, el planeta del país, la gravetat o la presència d'oxigen no serien una variable. Un color qualsevol no és cap propietat del fenomen que estudiem. Tots els països es troben al mateix planeta, per la qual cosa el valor de planeta sempre serà el mateix: la Terra. Com que no hi haurà variació, planeta no és una variable. En essència, la gravetat també és la mateixa entre països i també a tots els països hi ha presència d'oxigen. Tampoc, doncs, són variables. Sí que serien variables, en canvi, el nom del país, la seva extensió territorial, el nombre d'habitants o l'edat mitjana d'aquests habitants. Aquests elements són variables perquè tindrem països amb noms diferents, de grandàries territorials diferents, més o menys poblats i amb uns habitants més o menys envellits.

No totes les variables varien igual, i és per això que en podem trobar de diferents tipus. Normalment s'acostuma a establir una primera distinció entre variables categòriques i numèriques.

1) En direm variables categòriques si els valors que varien no són nombres, sinó categories, com el país, la raça de gos o el color del cabell.

2) En direm variables numèriques si els valors que varien són nombres, com la temperatura o l'edat.

Dins de cada grup de variables també podem fer dues distincions més, tal com veiem a la taula 6. Anomenarem variables categòriques nominals a aquelles variables categòriques, discretes i no ordenables. Anomenarem variables categòriques ordinals a aquelles variables categòriques, discretes i ordenables. Entre les variables numèriques, distingirem entre les variables numèriques discretes i les variables numèriques contínues.

Taula 6. Tipus de variables

Variable	Característiques	Operacions
Categòrica nominal	Categòrica, discreta, no ordenable	`==, !=`
Categòrica ordinal	Categòrica, discreta, ordenable	`<, <=, >, >=, !=, ==`

Variable	Característiques	Operacions
Numèrica discreta	Numèrica, discreta, ordenable	`<, <=, >, >=, !=, ==, +, -, *, /, etc.`
Numèrica contínua	Numèrica, no discreta, ordenable	`<, <=, >, >=, !=, ==, +, -, *, /, etc.`

Què vol dir exactament que les variables siguin ordenables o discretes? La diferència principal entre els dos tipus de variables categòriques és l'ordenabilitat. No es poden ordenar variables com el sexe o els països,[4] ja que no hi ha manera lògica d'establir un ordre de categories que vagi de major a menor. Les categòriques nominals emmagatzemen noms i només podem fer operacions d'igualtat entre aquestes variables (dir si són iguals o no ho són).

[4]No podem ordenar Argentina i Brasil pel seu nom per a establir quin valor és superior a l'altre. Només són noms de països i, com a molt, podrem classificar-nos alfabèticament. Sí que els podríem ordenar pel PIB, per exemple, però llavors no utilitzaríem la variable país, sinó la variable PIB per a ordenar-los.

Les categòriques ordinals, en canvi, sí que es poden ordenar de manera lògica. Segons una de les classificacions del Banc Mundial, un país pot ser molt pobre, pobre, ric o molt ric. Aquestes quatre categories tenen una forma lògica d'ordenar-se, de menys a més. Molt pobre seria la categoria més baixa i molt ric la més alta. Una altra variable ordenable seria la classificació de potencies mundials que ordena els països segons si són potències mitjanes, grans potències, superpotències, etc. A més de poder dir si aquestes variables són iguals o no entre elles, també podrem fer altres operacions com determinar si una categoria és superior o inferior a l'altra.

Les variables numèriques sempre les podrem ordenar de major a menor. Hi podrem fer operacions d'igualtat, dir si un valor és superior o inferior a l'altre i a més fer tot tipus d'operacions algebraiques com sumar, dividir o elevar al quadrat. La diferència principal entre els dos tipus de variables numèriques rau en si són o no discretes. Que una variable sigui discreta no vol dir que no faci soroll quan varia. La paraula discret té una altra accepció que significa que té un nombre finit de categories. Per a saber si una variable és discreta acostumem a observar si pot contenir decimals. Les persones poden tenir un fill o dos, però no un fill i mig. El nombre de fills d'una persona és una variable *numèrica discreta*. També la població d'un país o el nombre d'armes nuclears són variables discretes. No podem tenir mitja arma nuclear. La riquesa d'un país, en canvi, mesurada normalment per mitjà del PIB per càpita, pot tenir un nombre infinit de valors ja que la distància entre un dòlar i dos dòlars és grandiosa i pot acceptar tants decimals com siguin necessaris. La temperatura o l'esperança de vida també serien, en aquest cas, variables *numèriques contínues*.

Com veurem en aquest apartat, és molt important que R sàpiga amb quin tipus de variables tractem. La manera com definim el tipus de variable marcarà algunes configuracions per defecte d'R, el tipus d'operacions que podem fer i les visualitzacions, i també determinarà els mètodes estadístics més apropiats per a respondre a les preguntes que ens fem sobre les dades. Com ja us haureu adonat, una variable en R pren la forma d'un vector, de manera que

[5]Podem fer la prova de visualitzar algunes variables del marc de dades `md_agr` que hem creat en l'apartat anterior i imprimir-les en forma de vector amb `md_agr$pais`, `md_agr$any`, `md_agr$dem`.

podrem determinar el tipus de variable indicant si és un vector numèric, enter, de caràcter o lògic. Per utilitzar el vector com a variable dins un marc de dades l'haurem de cridar amb el nom del marc de dades, seguit del símbol `$` i finalment el nom de la variable (exemple `marcdades$variable`).[5]

2.1. Variables categòriques nominals

Les **variables categòriques nominals**, també dites qualitatives, són variables discretes no ordenables.

Aquest tipus de variables són discretes perquè prenen un número limitat de categories. Quan omplim qüestionaris no podem escollir entre sexes infinits, només tenim les caselles d'home o dona. Si ens pregunten pel nostre estat civil, tampoc podem escollir entre diversos valors. Normalment oscil·larà entre quatre: solter/a, casat/da, divorciat/da o vidu/a. En els estudis internacionals, els països, els continents o el tipus de règim (democràcia o no) són variables categòriques nominals. A part de discretes, aquestes variables no són ordenables perquè no podem dir si un valor és més alt o més baix que la resta. Podem classificar els països per continent o un país, però no podem dir si el fet de pertànyer a un continent o a un país determinat és menor o major. Per tant, amb variables categòriques nominals podem fer poques operacions. No les podem sumar ni restar. Tampoc podem dir si una és més gran que l'altra. L'únic que podem fer amb elles és classificar-les amb criteris d'igualtat.

Operacions amb variables categòriques nominals

Igual: ==
No igual: !=

Les variables categòriques nominals es poden emmagatzemar en R com a vectors de caràcter, com a vectors lògics o com vectors enters en forma de factors. Normalment, emmagatzemarem les variables categòriques com caràcters o com factors, encara que en determinats casos també ho podrem fer amb vectors lògics.

2.1.1. Caràcter

Com sabem, els vectors de caràcter o *strings* emmagatzemen tot tipus de caràcters, des de lletres fins a símbols especials. Sempre utilitzarem cometes per a referir-nos a aquests vectors. L'únic tipus de variable que poden representar els vectors de caràcter és la variable categòrica nominal. Com que no són ordenables pel seu valor, la manera com R ens ordena els valors serà per ordre alfabètic.

El més important que hem de saber sobre els vectors de caràcter és que hi ha un paquet a R, anomenat *stringr*, que ens pot ajudar a manipular-los amb facilitat. Sovint ens podem trobar que volem canviar tots els caràcters a lletra minúscula o que hi ha un nom en particular que volem anomenar d'una altra manera. Les funcions d'aquest paquet, que tenen la particularitat que totes comencen amb *str*, ens poden ajudar a fer aquests canvis. En la taula 7 podem

veure algunes de les funcions principals. La sintaxi és molt intuïtiva, ja que en el primer argument indiquem el nom del vector, en el segon els caràcters que seleccionem i en el tercer, si cal, els nous caràcters.

Taula 7. Funcions principals del paquet *stringr*

`str_detect()`	Retorna un vector lògic que indica si ha trobat el conjunt de caràcters que hem especificat.
`str_replace()`	Canvia en un vector un conjunt de caràcters per un altre.
`str_remove()`	Elimina el conjunt de caràcters que indiquem.
`str_to_upper()`	Converteix tots els caràcters a majúscules.
`str_to_lower()`	Converteix tots els caràcters a minúscules.

Per il·lustrar com funcionen aquestes funcions del paquet *stringr* podem crear un marc de dades de nom `eastg` amb un vector de caràcter: la columna *country* que té les categories "East Germany", "Germany" i "France"[6]. Si volem que "East Germany" i "Germany" estiguin en la mateixa categoria, les tres primeres funcions ens permeten fer-ho a través de processos diferents. La funció `str_detect()` ens crea vectors lògics, pels quals podem crear l'objecte `canvis`, que ens indicarà TRUE tots els valors del vector que continguin "East Germany". A continuació, utilitzarem els claudàtors per a seleccionar aquelles columnes de la variable `eastg$country` que siguin TRUE i li afegirem un vector amb el nom de "Germany". Amb `str_replace()` el procés és molt més directe, ja que simplement introduïm el nom del vector, el nom que volem reemplaçar i el nom nou. Finalment, `str_remove()` també ens permet fer el mateix procés, però en aquest cas hem d'indicar quins caràcters ha d'eliminar amb "East ".

[6]El crearem en format *tibble* amb aquest codi: `eastg <- tibble(country = c("East Germany", "Germany", "France"))`. Recordeu que heu de tenir el paquet *dplyr* carregat.

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

Fixeu-vos que si tinguéssim altres categories que també comencessim amb "East " no podríem utilitzar `str_remove()`, ja que també ens eliminaria els caràcters en aquestes categories. Totes aquestes funcions tenen la seva utilitat segons la situació en què ens trobem.

El paquet *janitor* com a alternativa

Una altra funció molt recomanable per a transformar vectors de caràcter és `clean_names()` del paquet janitor, que neteja i homogeneïtza els caràcters de manera gairebé automàtica. Per defecte, ens posarà els caràcters en minúscula i la barra baixa per a separar paraules. Hi ha més opcions de transformació que podeu consultar al quadre d'ajuda `?clean_names()`.

2.1.2. Factors

Un **factor** és un vector enter que emmagatzema informació categòrica.

Comprendre què és exactament un factor exigeix paciència. Podem pensar que és un entremig entre un vector enter i un vector de caràcter. Dels vectors enters sabem que no poden contenir decimals i que són ordenables. Dels vec-

tors de caràcter sabem que poden contenir el text que li vulguem posar. Doncs bé, un factor té les propietats d'un vector enter però se'ns presenta camuflat com si fos un vector de caràcter mitjançant els atributs que associem al vector que, recordem, proporcionen informació complementària sobre l'objecte en qüestió. D'aquesta manera, enlloc de visualitzar els nombres del vector enter, el que visualitzarem serà la informació que cada valor té etiquetada. Per exemple, suposem que creem un vector enter i associem el número 1 a l'etiqueta Argentina, el número 2 a Brasil, el 3 a Colòmbia i així successivament.

Diferència entre factors i caràcters

A diferència dels vectors de caràcter, els factors són una classe de vector molt més versàtil, ja que ens permeten emmagatzemar a la vegada categories i nombres. Els factors poden portar a confusió perquè tenen l'aparença d'un caràcter però realment són un número enter amb una etiqueta amb caràcters. Aquesta diferència l'hem de tenir molt en compte no solament quan fem operacions, sinó quan descarreguem bases de dades. Un factor amb caràcters pot ser interpretat fàcilment per R com un vector de caràcter quan realment nosaltres el volem utilitzar com un factor.

Per a crear un factor utilitzarem les funcions `factor()` o `as.factor()`. Normalment els factors es creen a partir dels vectors de caràcter, però per a comprendre bé la naturalesa d'un factor il·lustrarem primer com es crea a partir d'un vector enter.

L'argument stringsAsFactors

Moltes funcions tenen incorporades un argument que ens permet triar si volem que les variables categòriques siguin *strings* o factors. Tant `data.frame()` com la majoria de funcions per a importar marcs de dades com `read.csv()` tenen l'argument *stringsAsFactors*. Per defecte, aquest argument és TRUE, de manera que convertirà automàticament els caràcters en factors. Altres funcions, en canvi, com les del paquet *readr*, tenen per defecte que no converteixen els caràcters en factors.

Crear un factor a partir d'un vector enter

Fixem-nos en el codi següent. Hem creat un vector enter de nom `enter`. Si demanem tipus, classe i l'imprimim, no veurem res de significatiu: és un vector enter format per nombres enters. Ara transformem-lo a factor. Si tornem a demanar tipus i classe, veiem com continua sent un vector enter, però ara R el reconeix com a factor. Si l'imprimim, R ara ja sap que és un factor i ens diu que el vector té quatre nivells, corresponent a les quatre «categories» diferents del vector. A continuació associem un nom categòric a cada un d'aquests quatre nivells amb `levels()`. Si imprimim de nou el vector veurem com la consola ens ensenya els nivells del vector en lloc dels nombres. El valor 1 està associat amb el nivell Argentina, de manera que visualitzarem tots els valors 1 com a Argentina, en concret els dos primers valors de la sèrie. Fixeu-vos també que l'ordre dels factors estarà associat amb l'ordre dels números enters, no amb l'ordre alfabètic dels nivells.

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

Rarament, però, construirem un factor a partir d'un vector enter. L'exemple anterior ens ha servit per comprendre millor la naturalesa d'un factor. Però per comoditat normalment crearem els factors a partir de vectors de caràcter. En l'exemple següent, hem creat el vector `nuclears`, que respon a una variable categòrica binària que classifica sis estats indeterminats segons si són potències nuclears o no mitjançant dues categories: "Nuclear" i "No Nuclear". Podem repetir de manera semblant les operacions anteriors, primer examinant que efectivament és un vector de caràcter amb `typeof()` i `class()`. A continuació, el convertim a factor, en aquest cas, creant un nou objecte anomenat `factor_nuclears`. Amb `typeof()` podem comprovar que efectivament aquest nou objecte és un vector enter i amb `class()` veurem que és un factor.

Passar de factor a vector enter

Si volem passar el factor a enter tenim dues possibilitats. Mitjançant la funció `unclass()` ens eliminarà el factor però ens conservarà les etiquetes, mentre que si utilitzem la funció `as.integer()` ens eliminarà també les etiquetes i conservarà només els nombres.

```
nuclears <- c("Nuclear", "Nuclear", "No Nuclear", "Nuclear", "No Nuclear", "Nuclear")
typeof(nuclears)
class(nuclears)
factor_nuclears <- factor(nuclears)
typeof(factor_nuclears)
class(factor_nuclears)
factor_nuclears
attributes(factor_nuclears)
```

Fixeu-vos que, com que venim d'un vector de caràcter, aquesta vegada ens ha creat els nivells per ordre alfabètic. El primer nivell és "No Nuclear", malgrat que no és el valor que apareix primer en la seqüència del vector, mentre que el segon nivell és "Nuclear". Llavors podem dir que R, per ordre alfabètic, ha assignat el valor 1 a "No Nuclear" i el valor 2 a "Nuclear".[7]

[7] Cal tenir en compte que, encara que cada etiqueta correspongui a un número, això no vol dir que R entengui que la categoria assignada al número 1 és inferior a la categoria assignada al número 2. Per ara, R només ha posat etiquetes als nombres, de manera que aquestes variables continuen sent categòriques nominals. Més endavant aprendrem com ordenar-les de més gran a més petita.

Podem canviar els noms dels factors de dues maneres. Per a il·lustrar-ho, hem creat el factor `paisos` amb les categories següents: "BRA", "ARG", "VEN", "URU" i "PAR". Si volem canviar aquests noms, podem fer-ho en primer lloc amb la funció `levels()`. Amb `levels(paisos)` mirem primer l'ordre dels nivells. Veiem con estan ordenats alfabèticament. Un cop sabem l'ordre, hem de crear un vector amb els noms nous, en el mateix ordre que els nivells del factor i assignar-lo als nivells de l'objecte `paisos`. Si només volguéssim canviar una part dels nivells, llavors haurem de crear un vector amb els noms que vulguem canviar, per ordre, i seleccionar amb els claudàtors després de la funció `levels()` la posició dels nivells.

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


factor(paisos, labels = c("Argentina", "Brazil", "Paraguay", "Uruguay", "Venezuela"))
```

La segona manera de canviar els noms dels factors és a dins mateix de la funció `factor()`, on indicarem els noms nous, per ordre, en l'argument *labels*.

2.1.3. Lògics

Les variables categòriques nominals també poden prendre la forma d'un vector lògic, en què la variable només pot tenir dos valors discrets i normalment no ordenables: veritat (TRUE) i fals (FALSE). En aquest cas es tractarà d'una variable binària (també dita dicotòmica, booleana o *dummy*), que normalment serveix per a indicar la presència o absència d'un determinat concepte. Quan el concepte és present, la variable prendrà el valor lògic TRUE mentre que quan el concepte està absent prendrà el valor lògic FALSE.

La vida pot ser binària

Tota variable es pot convertir en un vector lògic. Per exemple, podem establir una variable lògica, *democràcia*, que prengui el valor TRUE si el país observat és una democràcia i el valor FALSE si no ho és. El mateix podem fer amb la variable *dona* (TRUE si és dona, FALSE si no ho és), *vermell* (TRUE si el color és vermell, FALSE si és qualsevol altre color), *any_2007* (TRUE si les dades són de 2007, FALSE si no ho són) o *paisos_rics* (TRUE si un país té un PIB per càpita superior a un llindar determinat i FALSE si és inferior).

Paquet *forcats* per als factors

Quan tinguem algun problema associat amb un factor és molt probable que el puguem resoldre amb el paquet forcats. Per exemple, si volem ordenar les categories del factor segons el nombre de freqüències de cada una d'elles ens ajudaran les funcions `fct_infreq()` o `fct_rev()`. També podem ordenar les categories amb `fct_relevel()`.

Les variables lògiques no són gaire freqüents en els marcs de dades, però són molt útils com a variable operativa quan hàgim de seleccionar la presència o absència de determinades condicions en una variable. A continuació hem creat el marc de dades `pov`, que conté les variables `country` i `poverty`. La primera variable és un vector de caràcter que representa els noms dels països, mentre que la segona variable és un vector numèric que representa la pobresa de cada país mesurada com el percentatge de persones que viuen per sota d'1,90 dòlars al dia.[8]

[8] Fixeu-vos que hem creat el marc de dades d'una manera diferent a la dels anteriors exemples: primer hem creat els vectors i després els hem unit amb la funció `data.frame()`. R interpreta els noms dels vectors com el nom que prendran les columnes del marc de dades.

```
country = c("Armenia", "Austria", "Benin", "Bolivia", "Brazil",
"Colombia", "El Salvador", "Ethiopia", "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
```

Amb el codi `pov$country == "Austria"` demanem que ens retorni un vector lògic els valors del qual siguin veritat si el valor corresponent a la variable *country* és igual a Àustria i fals si no és igual. Com veiem, només ens retorna com a TRUE el segon valor i els altres ens els retorna com a FALSE. Amb el codi `pov$poverty < 15` demanem un vector lògic que tingui els valors TRUE si el valor corresponent a la variable *poverty* és inferior a 15 i FALSE si és igual o superior. Com veiem, el nou vector lògic és una variable dicotòmica que ens diu per a cada observació si el percentatge de persones que viuen per sota de 1,90 dòlars al dia és inferior a 15.

Per a saber-ne més

Sobre la lògica dels operadors booleans bàsics, podeu mirar aquest vídeo (https://www.youtube.com/watch?v=6PpQS-YLWDQ&feature=youtu.be). Per a una descripció més avançada podeu consultar l'apartat 5.2.2. d'aquest manual (https://r4ds.had.co.nz/transform.html).

Operadors booleans

Els operadors booleans (OR, AND i NOT) són una manera comuna de treballar quan agrupem condicions lògiques. Quan fem cerques avançades per internet fem servir operadors booleans, ja que demanem que la cerca inclogui una paraula però no una altra o bé volem que inclogui o bé una paraula o bé una altra. En el nostre cas, suposem que volem saber quines observacions del marc de dades `pov` són o bé Àustria o bé tenen una pobresa inferior al 15 per cent. Aquesta funció ens la permet fer l'operador OR (símbol | a R), que ens retorna TRUE si almenys un dels requisits que hem indicat es compleix. En el primer cas, Armènia no és Àustria però té una pobresa baixa, per la qual cosa ens retorna un TRUE perquè compleix un dels requisits. En el tercer valor, Benín no compleix cap dels dos requisits i per tant ens retorna FALSE.

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

L'operador AND (símbol `&` a R) és més restrictiu i només ens retornarà TRUE en el cas que es compleixin tots els requisits indicats. En l'exemple següent, només el segon valor ens el retorna com a TRUE perquè és l'únic cas que és Àustria i té una pobresa inferior a 15.

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

Finalment, tenim l'operador NOT (símbol `!` a R) que ens indica que no ha de complir el requisit que indiquem. Les dues maneres d'utilitzar-lo que indiquem a continuació retornen el mateix vector. Demanem dos requisits: no pot ser Àustria i (AND) ha de tenir una pobresa inferior al 15 per cent. Com veiem, el símbol de negació es pot situar, en el primer cas, al principi de la fórmula o, en el segon cas, canviant l'igual pel símbol `!=`.

```
> !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 categòriques ordinals

Les **variables ordinals** són variables categòriques que es poden ordenar de manera lògica.

Les variables ordinals es diferencien de les variables categòriques nominals perquè són ordenables i de les numèriques discretes perquè els seus valors representen categories en lloc de nombres. Això vol dir que les operacions que podem fer amb els valors és saber si són iguals, diferents, més grans o més petits.

Operacions amb variables categòriques ordinals

Igual: ==
No igual: !=
Més gran: >
Més gran o igual: >=
Més petit: <
Més petit o igual: <=

Una variable ordinal ha de ser necessàriament tractada amb R mitjançant un vector lògic o un factor. Podem codificar la variable com a vector lògic sempre que sigui una variable binària i la categoria que sigui superior a l'altra codificada com a TRUE. En aquest cas, podem seguir les instruccions de l'apartat anterior sobre vectors lògics. En la majoria dels casos, però, tractarem les variables ordinals com a factors.

Exemple

Una enquesta pot preguntar «valori el seu grau de satisfacció amb la democràcia» i donar com a possibles respostes baix, mitjà o alt. És evident que alt és més gran que mitjà i mitjà és més gran que baix. Per tant, a la variable hi ha tres categories possibles i aquestes es poden ordenar de manera lògica.

Diferència entre una variable ordinal i una variable numèrica discreta

Si podem assignar un nombre a cada categoria d'una variable ordinal, què diferencia una variable ordinal d'una de discreta? Podríem assignar, per exemple, el número 1 a baix, 2 a mitjà i 3 a alt. Però aquests nombres no tindrien significat propi, a diferència de les variables numèriques. És a dir, l'assignació seria igual de vàlida si per indicar baix, mitjà i alt utilitzéssim 2, 4 i 6; 10, 11 i 12; 1.050, 1.100 i 1.150... En altres paraules, en les variables ordinals sabem l'ordre dels valors però desconeixem el valor de cada categoria i la distància exacta entre categories.

2.2.1. Factors

Els factors són uns dels instruments principals que tenim per a emmagatzemar variables categòriques, però el seu gran potencial és la manera com poden guardar variables ordinals, ja que tenen la capacitat d'ordenar les categories seguint l'ordre que indiquem. Per a il·lustrar-ho, hem replicat una classificació ordinal del Banc Mundial que categoritza els països segons el seu nivell d'ingressos, que pot ser *low*, *lower-middle*, *upper-middle* i *high*. Per a recrear aquesta classificació, hem construït el marc de dades `wb`, on la primera variable `country` mostra quinze països del Carib i la segona variable `income` mostra el nivell d'ingressos corresponent.[9] Fixeu-vos amb l'estructura de les variables, el tipus de vector de la variable `country`, els atributs de la variable `income` i també com es veuria aquesta darrera variable com a vector enter.

[9] Indicant `stringsAsFactors = FALSE` i aplicant `factor()` a la segona variable hem aconseguit que la variable *country* sigui un vector de caràcter i la variable *income* 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 ens hi fixem, podem comprovar que els nivells del factor `wb$income` estan ordenats alfabèticament, començant per "high", que seria el primer per ordre alfabètic. Aquesta observació la podem fer també amb `levels(wb$income)`. R encara no sap quin nivell és superior a l'altre i per això ens ordena els nivells alfabèticament. Per a convertir aquesta variable en ordinal, hem d'introduir la funció `factor()` amb el nom de la variable, un segon argument `order = TRUE` i un tercer argument amb `levels,` i un vector que indiqui, de més gran a més petit, com s'ordenaran els nivells. Proveu de fer l'operació següent:

```
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 després de transformar el vector consultem la classe de l'objecte, veurem que ens indica que és un "ordered" "factor". Si imprimim el vector `wb$income` podem comprovar que ens apareixerà l'ordre dels nivells, que no ens apareixia anteriorment. Ara R té clar que "low" és inferior a "lower-middle", que és inferior a "upper-middle", que és inferior a "high".

De la mateixa manera que podem convertir un vector de caràcter a factor amb `factor()` o `as.factor()`, podem fer el procediment a la inversa mitjançant `as.character()`. Si transformem la variable a caràcter (`wb$country <- as.character(wb$country)`), hem de tenir en compte que R eliminarà tant l'ordinalitat dels factors com els nombres enters que tenien associats i transformarà els nivells en caràcters. De la mateixa manera, `as.logical()` transformarà un vector qualsevol en vector en lògic i `as.numeric()` el transformarà en numèric.

2.3. Variables numèriques

Les variables numèriques, anomenades també cardinals o quantitatives, són variables que representen nombres en els seus valors.

Operacions amb variables numèriques

Igual: ==
No igual: !=
Més gran: >
Més gran o igual: >=
Més petit: <
Més petit o igual: <=
Sumar: +
Restar: –
Multiplicar: *
Dividir: /
Exponencial: ^
Logaritme: log()

Els valors de les variables numèriques es poden representar amb nombres que tenen significat per ells mateixos. Que tinguin significat numèric vol dir, en primer lloc, que sempre podrem ordenar els seus valors, ja que, per exemple, un nombre negatiu serà inferior a un nombre positiu i el número 500 serà superior al 450.

Que tinguin significat numèric també vol dir que, a més de ser ordenables, la distància entre els valors d'una variable numèrica és coneguda. En una variable ordinal no podem conèixer la distància entre un valor "Mig" i un valor "Alt". En canvi, en una variable numèrica podem conèixer i calcular la distància entre valors. Això ens permetrà fer operacions aritmètiques com sumar, restar, multiplicar, dividir, treure l'arrel quadrada o buscar el logaritme neperià.

Exemples de variables numèriques

La temperatura, la latitud, el percentatge de terra conreable, la quantitat d'ajuda oficial al desenvolupament, la migració neta, la taxa de creixement natural de la població o les emissions de diòxid de carboni a l'atmosfera.

Per a emmagatzemar dades numèriques, R utilitza dos tipus de vectors: el vector enter i el vector numèric o doble. Amb el vector enter emmagatzemarem normalment les variables numèriques discretes, que no poden tenir decimals, mentre que amb el vector numèric emmagatzemarem les variables contínues, que sí que poden contenir decimals. Encara que és important conèixer la distinció entre discretes i contínues, en la pràctica, amb R, aquesta distinció ens serà poc útil, ja que podrem fer les mateixes operacions tant amb un tipus de variable com amb l'altre. Per tant, la distinció té més sentit teòric que pràctic. A més, per emmagatzemar un vector enter ho haurem d'especificar amb una `L` al final de cada valor, cosa que no facilita la tasca de crear vectors enters.[10] A continuació mostrem dues maneres de crear un vector enter i mirar-ne el tipus de vector que ha creat.

[10] La lletra `L` no apareixerà en el vector quan el visualitzem, però és la manera que R sàpiga que l'ha d'emmagatzemar com a vector enter.

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

Com a norma general, doncs, i per ser pràctics, tractarem amb variables numèriques amb el vector doble o numèric, encara que hem de saber que també podem utilitzar un vector enter per a tractar amb les variables numèriques discretes. En un pla teòric, les variables numèriques discretes contenen valors numèrics específics que poden ser comptats o enumerats. Per tant aquests va-

lors són limitats i no accepten decimals, com per exemple el nombre de fills, la població d'un país, el nombre de dones als parlaments nacionals, el nombre d'habitants d'un país o el nombre d'homicidis.

Discretitzar una variable numèrica (1): El mètode *if_else()*

A vegades podem voler transformar una variable numèrica en una variable discreta, normalment categòrica. A aquest procés l'anomenarem *discretitzar*. Un dels mètodes existents és mitjançant la funció `if_else()`, disponible en el paquet *dplyr*. En aquesta funció hem de donar tres arguments:

- En primer lloc, establirem una condició lògica: que pugui ser veritat o fals.
- En segon lloc, assignarem el valor que tindrà la nova variable si la condició és veritat.
- I en tercer lloc, assignarem el valor de la nova variable si la condició és fals.

En l'exemple següent, indiquem que si el PIB per càpita és superior o igual a 30.000, el valor del nou vector serà 1. Si en canvi, el PIB per càpita és inferior a 30.000, el valor del nou vector serà 0.

```
if_else(data$gdp >= 30000, 1, 0)
```

El vector que creem no cal que sigui numèric. Si indiquem "Ric" i "Pobre" en el segon i el tercer argument ens retornarà un vector de caràcter mentre que si indiquéssim TRUE i FALSE ens retornaria un vector lògic. En el primer argument, amb la funció `between()` podrem indicar un interval de dades. Per exemple, en el codi següent, assignarem el valor "Medium" a tots els valors que estiguin entre 20.000 i 30.000 (inclosos) i el valor "Other" als que estiguin fora d'aquest interval.

```
if_else(between(data$gdp, 20000, 30000), "Medium", "Other")
```

Aquesta operació es pot fer també amb la funció del paquet base d'R `ifelse()`. Una altra funció molt similar que té *dplyr* és `recode()`.

Les variables numèriques contínues, per la seva banda, ens permeten adoptar un número infinit de valors. La infinitat s'ha d'entendre com l'espai que separa dos valors. Penseu en el percentatge de superfície forestal d'un país. La xifra podria ser, suposem, un 15,69 per cent. És un valor numèric continu, ja que pot acceptar decimals, però què el fa infinit en comparació amb un valor numèric discret? Trobareu la resposta en el quadre de la dreta. Són variables numèriques contínues el PIB per càpita, l'esperança de vida, el temps mitjà d'ús d'internet, l'ajuda oficial al desenvolupament rebuda o el percentatge de població que té accés a aigua potable.

Infinitat

Els valors numèrics, al cap i a la fi, es mouen en uns intervals determinats. El PIB per càpita no pot ser negatiu i és impensable que l'esperança de vida d'una població sigui de 300 anys. Què fa, doncs, que un valor sigui infinit? Per entendre perquè les variables numèriques contínues són infinites, pensem en l'espai que hi ha entre el 15 per cent i el 16 per cent. Si ho pensem bé, aquest espai és infinit ja que sempre podrem incloure xifres i xifres en la part dels decimals. En canvi, en les variables numèriques discretes aquest espai no existeix. Entre 15 habitants i 16 habitants, no hi ha cap possible valor entremig.

Discretitzar una variable numèrica (2): El mètode *case_when()*

Una segona manera de discretitzar una variable és amb la funció del paquet *dplyr* `case_when()`, especialment útil quan volem crear una nova variable amb més de dues categories. Agafant d'exemple el marc de dades `pov`, utilitzat anteriorment en aquest mòdul, volem crear una nova variable `f_poverty` amb tres categories a partir de la variable numèrica `poverty`. Als nombres inferiors a 2 els anomenarem "Low", als nombres entre 2 i 10 els anomenarem "Medium" i a la resta de valors els anomenarem "High".

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

Cada argument de la funció `case_when()` és una seqüència de dues fórmules. En la primera indiquem a quins valors aplica la transformació i, separat pel símbol `~`, a la segona indiquem el nou valor. Els nous valors no han de ser necessàriament de caràcter. Poden ser lògics o numèrics. Com veiem en el codi, hem utilitzat la funció `between()` per a indicar un interval de valors. Si en el darrer argument de la sèrie indiquem TRUE, ens agruparà la resta de casos que no haguem especificat en cap condició.

L'operació ens ha convertit la nova variable `f_poverty` en un vector de caràcter. Per a ferla ordinal, l'hem hagut de convertir en factor i assignar els nivells i l'ordre corresponent.

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

És important saber que podem fer dos tipus d'operacions amb les variables numèriques segons la longitud dels vectors:

1) En primer lloc, si fem una operació d'un sol valor amb un vector d'una determinada longitud, R aplicarà l'operació del valor a cada un dels valors del vector. En l'exemple següent, R multiplicarà per tres cada un dels valors del vector.

2) En segon lloc, podem fer operacions entre vectors d'igual longitud. En aquest cas, R retornarà un vector de la mateixa longitud i aplicarà l'operació entre vectors. En l'exemple, R multiplicarà el primer valor del primer vector pel primer valor del segon vector i el col·locarà en el primer valor del vector resultant. En el segon valor del vector resultant R haurà multiplicat el segon valor del primer vector pel segon valor del segon vector, i així successivament.

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

En aquest codi, en el primer cas ens retornarà el vector `c(3, 9, 18)` i en el segon cas el vector `c(1, 9, 36)`.

Resum

Aquest mòdul introductori d'R ha tingut la voluntat de començar com una classe d'idiomes. Per a parlar amb R hem de conèixer la gramàtica bàsica i els elements necessaris per a construir frases i aconseguir que el programa entengui el que li comuniquem. Per això hem conegut els dos elements imprescindibles per a treballar amb R i com es relacionen entre ells: els objectes i les funcions. Hem conegut i après a crear els dos objectes principals que utilitza R en l'anàlisi de dades en ciències socials: els vectors i els marcs de dades. Com a analistes de dades, és evident que la nostra funció no serà crear aquests objectes sinó que en la majoria dels casos només els haurem de descarregar de la seva font, normalment a internet. No obstant això, és important saber com es creen per a comprendre'n la seva estructura.

L'altre element imprescindible són les funcions. Al principi quedarem aclaparats per la gran quantitat de funcions que hi ha i la dificultat d'aprendre les tasques que fa cada una i els seus arguments. A base de pràctica i de consultar aquest o altres documents anirem ampliant de mica en mica el nostre vocabulari de funcions.

Finalment, hem après com traslladar característiques dels fenòmens que volem estudiar, les variables, a un objecte d'R, els vectors. En la taula 8 trobareu un resum del que hem vist en la tercera secció d'aquest mòdul.

R tracta diferent als vectors diferents

Per a veure una primera diferència de com R tracta una variable segons si està emmagatzemada com a vector, podeu provar d'utilitzar la funció `summary()` per a demanar un resum de la variable. Demaneu primer un sumari d'un vector de caràcter dels que hem tractat en aquest mòdul, després d'un vector i després d'una variable numèrica. Com podreu comprovar, la informació que veurem canviarà segons el tipus de vector que vulguem resumir.

Taula 8. Sumari de tipus de variables

Variable	Vectors	Exemple	Operacions
Categòrica nominal	Caràcter, lògic, factor	`"Japan"`	`==, !=`
Categòrica ordinal	Factor, lògic	`"Medium"`	`==, !=, >, >=, <, <=`
Numèrica discreta	Enter, doble	`108`	`==, !=, >, >=, <, <=` `+, -, *, /, ^, etc.`
Numèrica contínua	Doble	`203445.39`	`==, !=, >, >=, <, <=` `+, -, *, /, ^, etc.`

La nostra feina és ajudar R per tal que tingui associada cada variable amb el seu vector corresponent, ja que no sempre és així. Quan importem bases de dades podem trobar-nos amb variables numèriques a dins de vectors de caràcter, amb variables ordinals que no estan tractades com a factor ordinal o bé amb variables categòriques nominals que s'han guardat com a factors i les vulguem tenir com a vectors de caràcter. Tenir les variables associades amb el vector que correspon ens facilitarà molt la feina en l'anàlisi de les dades.

Exercicis d'autoavaluació

Per a un millor l'aprenentatge, intenteu fer mentalment el màxim d'exercicis possible, sense utilitzar R.

1) Quin és el valor de l'objecte nou?

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

2) Què ens retorna l'operació final d'aquesta seqüència?

```
cinc <- 5
cinc <- 4
cinc * cinc
```

3) Què ens retorna l'operació final d'aquesta seqüència?

```
Cinc <- 5
cinc <- 4
Cinc + cinc
```

4) Què ens retorna aquesta operació?

```
4:8
```

5) Quin nombre retorna aquesta operació?

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

6) Quants decimals retorna aquesta operació?

```
round(2.718)
```

7) Quin tipus de vector ens generarà aquest valor?

```
"TRUE"
```

8) Quin tipus de vector ens generarà aquest valor?

```
"324"
```

9) Què ens retorna l'operació final d'aquesta seqüència?

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

10) De quina classe ens apareixerà aquest vector?

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

11) Quin vector ens retorna aquesta operació?

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

12) Quin vector ens retorna aquesta operació?

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

13) Indica el resultat de les operacions següents:

```
md42 <- data.frame(p = c("França", "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 resultat del codi següent.

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

15) Quin vector obtindrem en el codi següent?

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

16) Indica el resultat lògic de l'operació següent.

```
TRUE > FALSE
```

17) Indica el resultat lògic de l'operació següent.

```
TRUE == FALSE
```

18) Indica el resultat lògic de l'operació següent.

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

19) Indica el resultat lògic de l'operació següent.

```
TRUE & FALSE
```

20) Indica el resultat lògic de l'operació següent.

```
TRUE | FALSE
```

Solucionari

1) `20`

2) `16`

3) `9`

4) `4 5 6 7 8`

5) `25`

6) `Zero. Amb round(2.718) ens retornarà el valor 3.`

7) `caràcter (perquè va entre cometes)`

8) `caràcter (perquè va entre cometes)`

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 segons l'atzar
d) 37500
```

14) `abq, lax, jfk, bcn`

15) `1 2 1 2 3 1`

16) `TRUE`

17) `FALSE`

18) `FALSE`

19) `FALSE`

20) `TRUE`

Solució de l'exercici 1.

```
numèric
caràcter
caràcter
caràcter
```

Solució de l'exercici 2.

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

```
round(pi)
```

Solució de l'exercici 4.

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

Glossari

<- Crea un objecte

: Selecciona una cadena de valors segons la seva posició

:: Especifica la funció d'una llibreria determinada

? Obre el quadre d'ajuda d'una funció o objecte

?? Fa una cerca per paraula clau

[ ] Retorna els valors seleccionats d'un vector

[,] Retorna files i columnes d'un marc de dades

$ Estableix la separació entre el marc de dades i la variable

| Booleà OR, retorna TRUE si es compleix un dels requisits

& Booleà AND, retorna TRUE si es compleixen tots els requisits

! Booleà NOT, nega la condició lògica

== Igual, usat en operacions lògiques

!= No és igual que

>= Més gran o igual que

< Més petit que

<= Més petit o igual que

> Més gran que

+ Suma

- Resta

***** Multiplica

/ Divideix

^ Eleva exponencialment un valor

`args()` Mostra els arguments d'una funció

`as.character()` Converteix una variable a caràcter

`as.data.frame()` Transforma un objecte en un marc de dades

`as_data_frame()` Transforma un objecte en un marc de dades (*dplyr*)

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

`as.logical()` Converteix una variable en lògica

`as.numeric()` Converteix una variable en numèrica

`as_tibble()` Crea un *tibble*, un tipus de marc de dades (*dplyr*)

`attributes()` Mostra els atributs d'un objecte

`c()` Crea un vector

`case_when()` Discretització múltiple d'una variable numèrica (*dplyr*)

`class()` Mostra la classe d'objecte

`clean_names()` Neteja els noms d'un vector de caràcter (*janitor*)

`data.frame()` Crea un marc de dades

`factor()` Crea un factor

`help()` Obre el quadre d'ajuda d'una funció o objecte

`if_else()` Discretització binària d'una variable numèrica (*dplyr*)

`length()` Retorna la longitud d'un objecte

`levels()` Mostra els nivells d'un factor

`ls()` Mostra els objectes creats

`max()` Retorna el valor màxim d'un vector

`recode()` Discretitza una variable numèrica (*dplyr*)

`rep()` Repeteix una cadena de valors

`round()` Arrodoneix un valor numèric

`sample()` Retorna la mostra aleatòria d'un objecte determinat

`seq()` Retorna una seqüència de valors

`sort()` Ordena els valors d'un vector

`sum()` Sumatori dels valors d'un objecte

`str_detect()` Retorna un vector lògic que indica si ha trobat el conjunt de caràcters que hem especificat (*stringr*)

`str_replace()` Canvia en un vector un conjunt de caràcters per un altre (*stringr*)

`str_remove()` Elimina el conjunt de caràcters que indiquem (*stringr*)

`str_to_lower()` Converteix tots els caràcters en minúscules (*stringr*)

`str_to_upper()` Converteix tots els caràcters en majúscules (*stringr*)

`tbl_df()` Crea un *tibble*, un tipus de marc de dades (*dplyr*)

`tibble()` Crea un *tibble*, un tipus de marc de dades (*dplyr*)

`tribble()` Crea un *tibble*, un tipus de marc de dades (*dplyr*)

`typeof()` Retorna el tipus d'objecte

`unclass()` Elimina la classe d'un objecte

`View()` Mostra l'objecte en el visor

Bibliografia

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