Importar, netejar, unir

PID_00268324

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

Un analista de dades no sempre es troba la informació a punt per a ser explorada. La seva feina principal és sovint molt més feixuga del que sembla, ja que les dades poden estar desordenades, amb noms mal escrits difícils d'interpretar o separades en diferents arxius d'internet. És per aquest motiu que, en alguns àmbits, els científics de dades han de dedicar la major part del seu temps a aquestes tasques (Lohr, 2014). L'objectiu d'aquest mòdul és ajudar-vos a convertir dades disperses i desconnectades entre si en un sol marc de dades preparat per a ser analitzat.

En estudis internacionals, les dades provenen en la majoria d'ocasions de les principals organitzacions i centres d'estudis internacionals, de manera que ja han estat tractades i sistematitzades prèviament. Aquestes entitats s'encarreguen de recollir, netejar i ordenar les dades per tal de presentar-les al públic. Fins i tot algunes dades tenen disponible una llibreria específica en R per a facilitar-nos el tractament. Això no treu la importància d'aprendre a adquirir i preparar les dades per a poder-les estudiar a fons.

En aquest mòdul us ensenyarem tres passos bàsics per a la preparació de les dades.

1) El primer consisteix a importar les dades a RStudio. En aquest primer apartat us ensenyarem diferents maneres d'incorporar les dades al programa.

2) Un cop incorporades, un segon pas és netejar les dades. Dedicarem el segon apartat del mòdul a aquesta tasca, que consisteix a modificar quan sigui necessari l'estructura del marc de dades, el tipus de variables que tenim i a prendre decisions sobre els valors perduts i els casos extrems.

3) Finalment, el tercer pas consisteix a unir diferents marcs de dades. Aquest procés és extremadament útil per a la nostra feina, ja que ens permet analitzar informació que prové de fonts diferents. Per tant, el procés d'unir bases de dades serà un pas essencial previ al tractament.

1. Importar dades

Importar contingut és el primer pas que hem de fer per a treballar amb dades a RStudio. A internet tenim multitud de bases de dades relacionades amb estudis internacionals, com World Bank, Eurostat, Correlates of War, etc.[1] En aquest apartat aprendrem a importar a R algunes d'elles. Tenim dues vies principals per a importar-les: mitjançant un paquet d'R o bé important un arxiu localitzat al nostre ordinador o a la xarxa.

(1)Una de les llistes més extenses de bases de dades en estudis internacionals que trobem a la xarxa la tenim en aquest GitHub (https://github.com/erikgahner/PolData).

1.1. Importar un paquet

A dins dels paquets que ja tenim actualment carregats a R ja hi ha una gran quantitat de bases de dades que ens poden anar bé per a practicar amb el programa. Un dels paquets de base d'R anomenat *datasets* conté prop d'un centenar de marcs de dades que podeu consultar teclejant `help = "datasets"`. Altres paquets que ja coneixem, com *dplyr* o *ggplot2*, incorporen també, entre molts objectes i funcions, alguns marcs de dades al seu interior. Hi ha altres paquets que estan allotjats a la biblioteca central d'R, el CRAN, com és el cas de *gapminder*, que són fàcils d'utilitzar perquè quan carreguem el paquet a R se'ns carreguen també els marcs de dades del seu interior. Per a instal·lar un paquet utilitzarem la funció `install.packages()` amb el nom del paquet entre cometes i per a carregar-lo a R utilitzarem `library()` amb el nom del paquet sense les cometes. Un cop carregat, podem veure les funcions i els objectes del paquet si introduïm el seu nom seguit de dos punts (per ex., `gapminder::`). Ens apareixerà un desplegable amb la informació.

Marcs de dades en R

En els paquets de base d'R veureu que hi ha marcs de dades curiosos com el de `Titanic` (llista de passatgers supervivents al Titànic), `discoveries` (anys de descobriments importants) o `sleep` (dades d'un estudi que mesurava l'efecte del consum de drogues en les hores de son). També podeu instal·lar i descarregar paquets que tenen una gran quantitat de marcs de dades per a practicar, com `openintro`.

La majoria de paquets que tenen relació amb els estudis internacionals, però, són una mica més sofisticats, ja que incorporen diverses funcions que haurem d'aprendre a utilitzar per a poder treballar amb els marcs de dades que contenen al seu interior. Per exemple, *psData* conté diversos marcs de dades de ciència política, com la *Polity IV* o la *Democracy-Dictatorship (DD)dataset* . Si volem utilitzar-los hem de fer servir la funcions específiques del paquet. Per exemple, si volem descarregar la *Polity IV* hem d'introduir la funció `PolityGet()`.

Marcs de dades a *dplyr* i *ggplot2*

El paquet *dplyr*, per exemple, té el marc de dades `starwars`. El paquet *ggplot2* incorpora `presidential` (mandats dels presidents dels Estats Units des d'Eisenhower). Si voleu saber-ne més detalls, poseu el nom a la consola amb un interrogant al davant (exemple: `?Titanic`).

```
poliv <- PolityGet()
poliv <- PolityGet(vars = "polity2")
```

Si només volem descarregar una variable, haurem d'utilitzar l'argument *vars* i introduir el nom de la variable entre cometes. En el cas de la base de dades *DD*, podem fer el mateix procediment amb `DDGet()`. El paquet *psData* també porta incorporat un marc de dades anomenat `countrycode_data`, molt útil per a unir marcs de dades, ja que té enregistrats diversos noms i codis que prenen els països en diferents marcs de dades. Podeu activar-lo introduint-ne el nom i consultar l'estructura. A les files hi trobareu els diferents noms que pot adoptar

un país i a les columnes la nomenclatura que utilitza per a cada país una base de dades internacional diferent. Així, podem transformar fàcilment els noms de països per poder comparar dades que es troben en diferents marcs de dades.

Un paquet fàcil d'utilitzar és *unvotes*, que consta de tres marcs de dades i té registrades totes les votacions històriques a l'Assemblea General de Nacions Unides. El marc de dades principal del paquet és `un_votes`, que conté a la variable *rcid* un registre del número de votació i a la variable *vote* el sentit del vot de cada país: sí, no o abstenció. A `un_roll_calls` teniu la informació més ampliada de cada votació mentre que `un_roll_call_issues` ens ajuda a cercar per temàtica de cada votació.

Altres marcs de dades de *psData*

També es pot descarregar la Database of Political Institutions amb `DpiGet()`, la base de dades de crisis econòmiques de Carmen M. Reinhart amb `RRCrisisGet()`, una extensa base de dades del Fons Monetari Internacional i el Banc Mundial amb `IMF_WBGet()` i el `WinsetCreator()`.

El paquet de *World Development Indicators* (WDI) conté centenars d'indicadors que es troben allotjats a la base de dades del Banc Mundial. Un cop haguem carregat el paquet podrem remenar els indicadors amb el motor de cerca `WDIsearch()`, on introduirem la paraula clau que ens interessa. Si volem cercar per més d'una paraula clau les separarem amb `.*`. En l'exemple que veiem a continuació, la cerca per *military* ens retorna cinc entrades. Hem fet una segona cerca amb *military* i *gdp*, que ens ha retornat només una entrada. A la consola veurem com R ens retorna un marc de dades de dues columnes, el primer amb el codi de la variable i el segon amb la seva descripció. En la cerca que acabem de fer el codi de la variable és *MS.MIL.XPND.GD.ZS* i la descripció *Military expenditure (% of GDP)*. Per a importar aquesta informació utilitzarem la funció `WDI()`, on indicarem el codi de la variable i, opcionalment, un vector amb els països i els anys que volem que ens filtri.

Paquet *countrycode*

Per una versió més ampliada de codis de països en diverses bases de dades podeu instal·lar i carregar el paquet *countrycode*. Un cop carregat, imprimiu el `codelist_panel` per comprovar la gran quantitat de codis diferents que emmagatzema aquest paquet.

```
WDIsearch("military")
WDIsearch("military.*gdp")
mil_exp <- WDI(indicator = "MS.MIL.XPND.GD.ZS", country = c("GB"), start = 1970, end = 2005)
str(mil_exp)
names(mil_exp)[3] <- "mil_exp_uk"
plot(mil_exp$mil_exp_uk)
```

En l'exemple, hem creat l'objecte `mil_exp` amb les dades de la despesa militar del Regne Unit entre els anys 1970 i 2005. Després de comprovar que la tercera columna es refereix als valors amb una visualització de l'estructura, hem canviat el nom de la columna per un altre més fàcil de recordar i a continuació hem visualitzat les dades amb un diagrama de dispersió.

El paquet *eurostat* funciona d'una manera semblant al WDI, també amb un motor de cerca `search_eurostat()`. Aneu en compte perquè aquest motor és sensible a les majúscules. La cerca és una mica menys intuïtiva, per la qual cosa si us va millor podeu cercar les variables que us interessin al web d'Eurostat. En l'exemple següent hem cercat la paraula *pollution* i hem obtingut dues entrades. A continuació hem seleccionat només la primera columna de les dades per veure més clarament la descripció. Hem comprovat que la segona entrada *Environmental protection expenditure ...* era la que més ens interes-

sava i hem demanat la segona fila de la segona columna, que ens ha retornat el codi *env_ac_exp3*. A continuació, hem importat la variable d'interès amb la funció `get_eurostat()` i hem anomenat `pollution` al marc de dades resultant. Finalment hem demanat un sumari.

```
search_eurostat("pollution")
search_eurostat("pollution")[, 1]
search_eurostat("pollution")[2, 2]
pollution <- get_eurostat("env_ac_exp3", time_format = "num")
summary(pollution$values)
```

Hi ha més bases de dades que tenen paquet a R i funcionen de manera semblant als descrits anteriorment: o bé podem accedir directament al marc de dades o bé incorporen un motor de cerca amb la funció *search* i una funció *get* per a importar les dades. En la taula 1 mostrem una selecció de diversos paquets relacionats amb els estudis internacionals.

Paquets fora del CRAN

Hi ha altres paquets que no estan allotjats al CRAN, però per als quals algun desenvolupador ha creat algun procediment especial per a poder-los carregar. Vegeu, per exemple, els tutorials d'ús de *rqog* (Quality of Government) o de *wgi* (World Governance Indicators).

Taula 1. Paquets d'R de bases de dades

Base de dades	Codi	Descripció
Banc Mundial	`wbstats`	Paquet de dades del Banc Mundial.
Ciència política	`psData`	Recull de bases de dades de ciència política.
Comerç OEC	`oec`	Paquet de l'Observatory of Trade Complexity (OEC).
Comerç UN	`comtradr`	Dades de comerç de Nacions Unides.
Eurostat	`Eurostat`	Paquet de la base de dades Eurostat.
Estats	`states`	Dades d'estats independents al món des de 1816.
FMI (1)	`IMFData`	Paquet del Fons Monetari Internacional (FMI).
FMI (2)	`imfr`	Alternativa per accedir a l'FMI.
OCDE	`OECD`	Paquet per a extraure dades de l'OCDE.
Vots a l'ONU	`unvotes`	Vots de cada país a l'Assemblea General de l'ONU.
WDI	`WDI`	Indicadors de desenvolupament del Banc Mundial.

1.2. Importar un arxiu

Per a importar un arxiu, ho podem fer manualment per mitjà de File -> Import Dataset o mitjançant el codi. En aquest apartat expliquem com crear el codi per a importar els arxius. El primer que hem de saber és que les bases de dades poden estar emmagatzemades en diferents tipus d'arxiu. Els més comuns els descrivim en la taula 2:

Taula 2. Tipus d'arxiu per a emmagatzemar bases de dades[2]

Tipus	Extensió	Paquets i funcions
CSV	`.csv`	`utils::read.csv(), utils::read.csv2(), utils::read.table()`
Excel	`.xls / .xlsx`	`gdata::read.xls(),readxl::read_excel()`
R	`.Rdata`	`base::readRSD()`
SPSS	`.sav`	`haven::read_spss, foreign_read.spss()`
STATA	`.dta`	`haven::read_stata(), haven::read_data(), foreign::read.dta()`
TXT	`.txt, .tab, .tsv`	`utils::read.delim(), utils::read.delim2(), utils::read.table()`

[2]A la columna de paquets i funcions hem posat el nom del paquet que correspon a cada funció.

El segon que hem de saber és que el primer argument que introduirem dins de totes aquestes funcions és la localització de l'arxiu que conté la base de dades. L'arxiu pot estar localitzat en el nostre ordinador o a la xarxa. Si està localitzat en el nostre ordinador, recomanem tenir-lo ubicat a la carpeta que hem indicat a R com el nostre directori de treball i aplicar una de les funcions de la taula anterior. Si, pel contrari, l'arxiu està localitzat a la xarxa, podem actuar de diverses maneres:

Descàrrega des del directori de treball

Podem saber on tenim ubicat el directori de treball amb la funció `getwd()` mentre que amb `dir()` veurem quins arxius hi tenim guardats. Amb `setwd()` o manualment a la pestanya *Files* podrem canviar la ubicació del directori de treball. En aquest manual no explicarem com carregar arxius ubicats al nostre ordinador però fora del directori de treball.

1) La primera és descarregar l'arxiu manualment, posar-lo al nostre directori de treball i obrir-lo amb una de les funcions de la taula anterior com si estigués localitzat al nostre ordinador.

2) La segona és per mitjà de la funció `download.file()`. Amb aquesta funció indiquem la direcció web en el primer argument i el nom de l'arxiu que crearem en el segon argument. Aquesta operació ens guardarà l'arxiu en el directori de treball i després el podrem importar com en els procediments que acabem d'explicar.

3) La tercera opció és importar-lo directament d'internet amb una de les funcions indicades a la taula 2 fent servir com a primer argument la direcció web. L'opció triada dependrà de les vostres preferències i, a vegades, de com estigui ubicat l'arxiu a la xarxa.[3]

[3]A vegades trobarem bancs de dades en què només podem descarregar una base de dades si hi estem subscrits, per la qual cosa no podrem descarregar l'arxiu directament.

1.2.1. Arxius plans (.csv, .txt, .tab, .tsv)

Ens referim a arxius plans com els arxius de text que contenen els valors separats per algun caràcter no alfabètic, com un espai, un guionet, un punt i coma o un punt. En aquest apartat en distingirem de dos tipus:

- els arxius Comma Separated Values (CSV)
- els Tab-Separated Values (utilitzarem TXT com a abreviació)

Aquests arxius ocupen molt poc espai de memòria de manera que el procés per a importar-los o guardar-los los acostuma a ser molt ràpid. Normalment els CSV estan separats per comes, mentre que els TXT acostumen a estar separats per tabulacions o espais. Si obrim un CSV tindria més o menys l'aspecte següent:

```
pais,pibcap,habitants,deute,exportacions,importacions
Alemanya,44469,82.79,64.1,1.33,1.08
Àustria,47290,8.77,78.4,2.45,3.44
Bèlgica,43323,11.35,103.1,4.66,5.32
```

Podem veure com, a cada fila, la separació entre valors està marcada per la coma. El punt serveix per a marcar els decimals mentre que, en el cas de l'exemple, podem observar que la primera fila representa els títols de les columnes. Per a importar aquests tipus d'arxiu a R tenim diferents paquets. En aquesta assignatura ensenyarem les funcions més bàsiques, que es troben en el paquet *utils* que ve de sèrie amb R[4]. La funció per a importar arxius CSV `read.csv()` porta associats per defecte els arguments següents:

(4)Al CRAN podem trobar-hi altres paquets per a descarregar arxius com per exemple *readr* o *data.table*.

```
read.csv("arxiu.csv", header = TRUE, sep = ",", dec = ".", stringsAsFactors = TRUE)
```

En primer lloc, posarem el nom de l'arxiu que haurem situat abans al directori de treball o l'adreça web on es troba. És possible que amb el primer argument ja en tinguem prou per a importar la taula i no haguem d'especificar-ne cap més. Sovint, però, pot passar que la primera fila no porti el títol de les columnes, de manera que haurem d'especificar `header = FALSE`. Si és així, a dins de la funció hi indicarem un nou argument `col.names` on indicarem el títol de les columnes (per ex., `col.names = c("pais", "pibcap", "deute", "exportacions", "importacions")`). El vector haurà de tenir tants valors com columnes té el marc de dades.[5] Els arxius CSV tenen per defecte que el caràcter que separarà els arxius és una coma, però si és un altre caràcter l'haurem d'especificar. Per exemple, `sep = " "` indicarà que el separador és un espai. Els separadors dels arxius plans tenen molt a veure amb els sistemes d'emmagatzemar bases de dades. El sistema americà utilitza `read.csv()` i separa els decimals amb el punt i els valors amb la coma mentre que el sistema europeu utilitza `read.csv2()` i separa els decimals amb una coma i els valors amb un punt i coma. Finalment, tots els valors que ens detecti com a vectors de caràcter es convertiran en factors. Si volem canviar-ho, podem posar l'argument a `stringsAsFactors = FALSE`.

(5)De la mateixa manera, R ens crearà el tipus de vector associats a cada variable per defecte, però ho podem especificar manualment amb l'argument `colClasses` (per ex., `colClasses = c("character", "factor", "NULL", "numeric", "logical")`). R no ens importarà la columna indicada com a NULL.

Per a importar arxius TXT utilitzarem `read.delim()`, que fa exactament el mateix que la funció que acabem de veure però té per defecte el separador `sep = "\t"` (que significa tabulació). La funció `read.delim2()` ens importarà

(6)De fet, hem de pensar que en la pràctica totes aquestes funcions que hem vist són la mateixa funció però es distingeixen perquè tenen per defecte alguns arguments diferents.

els arxius en sistema europeu. La funció `read.table()` pot llegir tant CSV com TXT, però normalment li haurem d'especificar més arguments. El seu separador per defecte és `sep = "/"`[6].

[6]Fins i tot pot ser convenient preparar l'arxiu en Excel mateix, treure les primeres files i columnes buides, i guardar-lo directament en format *.csv*.

1.2.2. Excel

Excel és un programa molt útil per a treballar amb fulls de càlcul i bases de dades de petites dimensions. Sovint, algunes bases de dades es guarden en aquest format per la qual cosa necessitarem saber com importar arxius d'Excel a R. Abans de tot, hem de tenir en compte dues consideracions importants: les dades d'Excel acostumen a estar repartides en diversos fulls (sabreu que hi accedim a través de pestanyes visualitzables a la part inferior dels fulls de càlcul) i moltes vegades les dades no comencen a la primera fila i la primera columna del full de càlcul. És per això que pot valdre la pena primer fer una ullada prèvia al document en Excel per a fer-nos una idea de com estan distribuïdes les dades d'interès.[7] Els arxius d'Excel acostumen a tenir les extensions *.xls* o *.xlsx*. Per a importar-los a R utilitzarem els paquets *readlxl* o *gdata*.

El paquet XLConnect

Una altra opció per a descarregar Excel és l'XLConnect, que permet fer una importació més selectiva i escollir entre diferents pestanyes de dins d'un full de càlcul. Per exemple, si volem obtenir només les 10 primeres columnes de la segona pestanya del document "document.xlsx", primer carregarem l'arxiu `doc <- loadWorkBook ("document.xlsx")` i després el llegirem especificant el que volem importar: `readWorkSheet(doc, sheet = 2, startCol = 1, endCol = 10)`. La funció `getSheets()` fa una importació general de tot el document.

Dins del paquet *readxl*, necessitarem la funció `read_excel()` per a importar les dades. Si només introduïm a dins de la funció el nom del document, R ens importarà el primer full. Podem especificar el full que volem importar amb l'argument addicional *sheet* on indiquem el número de full o el nom del full entre cometes. L'exemple següent indica a R que ha de buscar l'arxiu `"docexcel.xls"` en el directori de treball, importar el primer full de càlcul del document, interpretar la primera fila com el títol de les columnes i no eliminar cap fila.

```
read_excel("docexcel.xls", sheet = 1, col_names = TRUE, skip = 0)
```

En l'argument `col_names` podem especificar TRUE si la primera fila representa el nom de les columnes i FALSE si no hi ha una fila amb el nom de les columnes, o bé podem crear un vector de caràcter indicant els noms de les columnes. Amb `skip` indiquem quantes files ens hem de saltar abans de començar a importar dades.

Importar diversos fulls

Si volem baixar diversos fulls de càlcul, haurem de crear diversos codis, cada un important un full de càlcul diferent. Per a saber el nom de cada full podem utilitzar la funció `excel_sheets()` i especificar el nom de l'arxiu del qual volem obtenir la llista.

El paquet *gdata* també llegeix documents d'Excel amb la funció `read.xls()`. Podem especificar el número de full de la mateixa manera que hem indicat anteriorment. La principal diferència que té aquesta funció és que utilitza la majoria d'arguments per defecte de `read.csv()`, de manera que tenim l'opció d'especificar si volem que ens converteixi els caràcters en factors o no (serà TRUE, per defecte).

1.2.3. Stata i SPSS

Unir diversos fulls de càlcul d'Excel

Un cop hem importat en diversos marcs de dades diferents fulls de càlcul d'Excel, tenim l'opció d'unir-los, sempre que els marcs de dades tinguin el mateix nombre de files. Amb la funció `cbind()` podem especificar el nom de cada un dels marcs de dades. Si les primeres columnes (per exemple, país i any) es repeteixen en tots els fulls de càlcul, podem eliminar-los indicant el número de la columna entre claudàtors (per ex., `cbind(full1, full2[-2], full3[-1])`).

També hi ha paquets pensats per a importar bases de dades de programari semblant a R, com Statistics and Data (Stata) o Statistical Package for Social Sciences (SPSS). El programa Stata és utilitzat principalment per l'econometria i guarda els arxius amb *.dta*, mentre que SPSS és utilitzat més aviat en ciències socials i guarda els arxius amb *.sav* o *.por*. Per a importar aquests arxius a R podem utilitzar dos paquets: *haven* o *foreign*.

1) El paquet *haven* utilitza `read_stata()` i `read_dta()` per a importar arxius STATA de les versions 8 a la 15 i `read_spss()` per a importar arxius SPSS. La complicació principal per a R amb els arxius provinents d'aquests programes estadístics és que els vectors de caràcter estan emmagatzemats com a numèrics i guarden els caràcters com a etiquetes. Amb *haven* no és possible convertir les dades directament a factors, de manera que ho haurem de fer manualment després de la importació. Per a fer-ho, utilitzarem la funció `as_factor()` de *haven* per a transformar el vector en qüestió a factor i, si cal, després el transformarem a caràcter amb `as.character()`.

2) Com a alternativa a descarregar STATA i SPSS tenim el paquet *foreign*, que suporta molts tipus de formats diferents a part dels esmentats. En el cas d'STATA utilitzarem `read.dta()` i en el cas d'SPSS utilitzarem `read.spss()`. El principal inconvenient que té la funció `read.dta()` és que no suporta arxius STATA superiors a la versió 12. No obstant això, aquesta funció sí que ens permet convertir directament les etiquetes de les variables a factors amb l'argument ja establert per defecte `convert.factors = TRUE`. La funció `read.spss()` també té predeterminat que converteixi les etiquetes d'SPSS a factors mitjançant l'argument `use.value.labels = TRUE`. Sí que és important especificar sempre `to.data.frame = TRUE`, ja que per defecte no ens crea un marc de dades sinó un altre tipus d'objecte.

1.3. Guardar el marc de dades

Finalment, després d'haver fet les transformacions pertinents en un marc de dades, podem tenir la necessitat de guardar-ho en un arxiu. El programa R té un format propi que té com a extensió *.RData* i utilitza la funció `saveRDS()`. El gran avantatge de guardar-ho en aquest format és que no perdrem informació sobre el tipus de variables, l'ordre dels factors, etc. A continuació, veiem com guardar el marc de dades `md` en format R, que anomenarem `"docR.xls"`. En segon lloc, veiem l'operació contrària: com importar l'arxiu cap al marc de dades `md` mitjançant la funció `readRDS()`:

```
saveRDS(md, "docR.xls")
md <- readRDS("docR.xls")
```

També podem guardar el marc de dades en un format més versàtil que pugui ser obert en altres programes com Excel, SPSS o STATA. Una bona opció és fer-ho en CSV per mitjà de `write.table()`. En el codi següent hem il·lustrat un exemple en què guardem el marc de dades de nom `md` a l'arxiu `"doc.csv"`. En el tercer argument especifiquem que volem que ens separi els valors per comes, que el punt marqui els decimals i que volem que ens guardi els noms de les columnes en la primera fila. Per defecte, `row.names` és `TRUE`, de manera que ens guardarà una primera columna addicional amb el número de cada fila. És convenient evitar-ho, per tant posarem l'argument a `FALSE`.

```
write.table(md, file = "docR.csv", sep = ",", dec =  ".", col.names = TRUE,  row.names = FALSE)
```

Els paquets que hem vist, com *readxl*, *haven* o *foreign* porten les seves pròpies funcions per a exportar les dades a arxius en format Excel, SPSS o STATA. També és una bona opció importar i exportar arxius amb el paquet *readr*, que no hem vist en aquest apartat però que haurem de conèixer si volem adquirir un domini més avançat en la importació i exportació de marcs de dades.

2. Netejar bases de dades

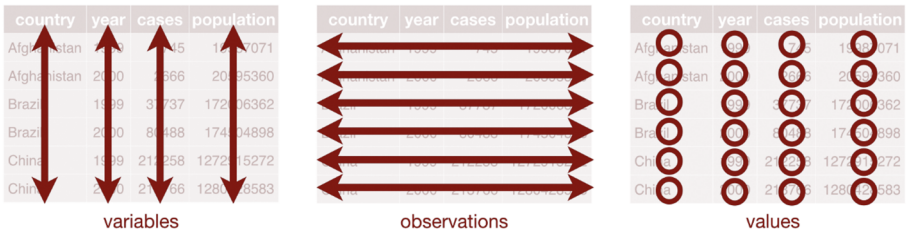
Quan importem un marc de dades a R, no necessàriament ens arriba net, amb les dades a punt per a ser explorades i analitzades. Considerem net un marc de dades si compleix quatre requisits (Wickham, 2014):

- cada tipus d'unitat observacional ha de ser un marc de dades diferent,
- les observacions han d'estar representades a les files,
- les variables han d'estar representades a les columnes i
- cada casella ens ha d'indicar un valor.

El primer requisit implica que només podem tenir una sola unitat d'anàlisi en cada marc de dades. Què observa el marc de dades en qüestió? Persones, països, dades de països, regions? El que no tindria sentit és que les dades en un mateix marc es referissin a vegades a persones i a vegades a països. En la majoria de casos treballarem amb el país com a unitat d'anàlisi.

Els altres tres requisits per a tenir un marc de dades net es refereixen a l'estructura d'un marc de dades, que resumim a la figura 1. Un marc de dades és una col·lecció de valors, cada un d'ells representats en una casella diferent. Cada valor pertany a una variable i observació determinada. Cada variable conté tots els valors que mesuren el mateix atribut en totes les unitats. Cada observació conté tots els valors que mesuren la mateixa unitat a través de tots els seus atributs. Si la taula compleix aquests requisits, vol dir que tenim una taula neta.

Figura 1. Requisits per a una taula neta



Font: Grolemund i Wickham (2017). *R for Data Science.* O'Reilly. CC BY-NC-ND 3.0 US.

El contrari d'una taula neta és una taula bruta. Podem trobar que els noms de les columnes siguin valors en lloc de variables, que algunes variables no estiguin representades pel tipus de vector que volem en cada cas, que hi hagi més files o columnes de les necessàries o que hi hagi una gran quantitat de dades perdudes o valors extrems que poden ser errors en les dades. Un dels símptomes de taula bruta (encara que no és un requisit necessari) és que tinguem moltes variables i poques observacions, ja que és possible que alguns valors categòrics estiguin fent la funció de falsa variable. El primer que farem abans

de començar la neteja és fer un anàlisi exploratori general, per fer-nos una idea de l'estructura del marc de dades amb funcions com `str()`, `glimpse()`, `names()`, `summary()`, `hist()` o `plot()`.

2.1. Netejar amb *tidyr*

Si ens trobem que un marc de dades no compleix els requisits d'una taula neta, el que necessitarem és el paquet *tidyr*. Haurem d'assegurar-nos que estigui instal·lat i carregat al nostre ordinador. Aquest paquet ens ajuda a transformar un marc de dades principalment de dues maneres:

- convertint noms de columna en valors d'una nova variable, o
- convertint valors d'una variable en noms de columnes.

Per a saber-ne més

Sobre el paquet *tidyr*, el científic de dades d'RStudio Garrett Grolemund ha preparat un GitHub amb una molt bona guia (http://garrettgman.github.io/tidying/) per a netejar marcs de dades.

En el primer cas allarguem el marc de dades, ja que reduïm el nombre de columnes i ampliem el nombre de files. Ho farem amb la funció `gather()`. En el segon cas eixamplem el marc de dades, ja que reduïm el nombre de files i ampliem el nombre de columnes. Ho farem amb la funció `spread()`. També aprendrem dues funcions més de *tidyr* que van molt lligades a aquestes dues: `separate()` i `unite()`. Trobem resumida la utilitat de les quatre funcions a la taula 3.

Taula 3. Funcions bàsiques amb *tidyr*

`gather()`	Allarga la taula transformant vàries columnes en valors d'una nova columna.
`spread()`	Eixampla la taula transformant els valors d'una columna en noms de noves columnes.
`separate()`	Separa els valors d'una columna en diverses columnes.
`unite()`	Uneix els valors de diverses columnes en una columna.

2.1.1. Les quatre funcions de *tidyr*

Comencem aprenent el funcionament de `gather()`. Fixeu-vos en el marc de dades següent, `taula_bruta`, que trobareu disponible en l'annex, en el qual veiem el nombre de protestes ciutadanes que hi ha hagut a diversos països, separades per l'any en què s'han produït a cada columna. Diríeu que és una taula neta?

```
> taula_bruta
     pais X2016 X2017 X2018
1 Espanya    64    83    95
2  França    43    34    33
3  Itàlia    56    53    67
4   Mèxic    35   101   120
5    Japó    12    19    18
```

Sembla que no. A les columnes veiem indicats els anys 2016, 2017 i 2018, que realment haurien de ser valors de la variable any.[8] Seguint els criteris del que considerem una taula neta, necessitaríem eliminar les columnes existents i crear dues columnes noves, la columna *any* i la columna *protesta*. Això ho podem fer fàcilment amb la funció `gather()`, on primer indicarem el marc de dades que volem transformar, seguit pels noms de les dues noves columnes: el primer indicarà el nom de la nova columna que representarà els valors que fins ara eren títols de columna (en direm *columna clau*), mentre que el segon ens indicarà el nom de la nova columna que representarà les dades que estaven a les observacions (en direm *columna valor*). En darrer lloc, indicarem els noms de les columnes que cal agrupar. Si són totes les columnes del marc de dades no cal que indiquem res. Si són diverses columnes ho haurem d'especificar amb un concatenat. És molt possible que ens sigui més còmode indicar amb el símbol negatiu les columnes que no s'han d'agrupar. Les dues possibilitats, que especifiquem a continuació, donaran el mateix resultat.

[8] Recordeu que els noms de columnes no poden començar amb un nombre, de manera que, en R, hi posem una X al davant. Més endavant ja la traurem.

Funcions *parse* del paquet *readr*

El paquet *readr* conté les funcions *parse*, que també són molt útils per a arreglar marcs de dades. Per exemple, en aquest cas, podem netejar un vector de caràcter i deixar només els nombres amb `parse_number()`. En el nostre cas, introduiríem `taula_neta$any <- parse_number(taula_neta $any)`.

```
gather(marc de dades, columna clau, columna valor, columnes que s'han d'agrupar)
taula_neta <- gather(taula_bruta, any, protestes, c(X2016, X2017, X2018))
taula_neta <- gather(taula_bruta, any, protestes, -pais)
```

Ara tenim el nou marc de dades `taula_neta`, que compleix els requisits de tenir les observacions a les files i les variables a les columnes. Per a acabar de netejar la taula correctament encara haurem de fer un pas més, ja que a la columna d'any tenim els valors amb una X al principi: X2016, X2017 i X2018. Una manera de treure aquesta X és amb la funció `separate()`, que separa els valors d'una columna pel caràcter que indiquem. A dins de la funció hem d'indicar el nom del marc de dades, seguit de la columna que conté els valors que volem separar. A continuació introduïm un vector amb el nom acompanyat de cometes que prendran les noves columnes. Finalment, amb l'atribut *sep* indicarem, també entre cometes, quin és el caràcter o caràcters que fan de separadors i que, per tant, seran eliminats.

Netejar vectors amb *stringr*

Una altra opció que permet eliminar part d'un vector de caràcter és amb la funció `string_replace()` del paquet *stringr*. En l'exemple, hauríem d'introduir l'operació següent: `str_replace(taula_neta $any, "X", "")`.

```
separate(marc de dades, columna, "nom noves columnes", sep = " ")
taula_sep <- taula_neta %>%
  separate(any, c("res", "any"), sep = "X") %>%
  select(-res) %>%
  head(8)
```

Si només utilitzem la funció `separate()`, R ens crearà dues columnes: la columna *any* amb l'any corresponent i la columna *res* que tindrà tots els valors en blanc, ja que no hi ha res a l'esquerra de la X que hem indicat com a separador.[9] Per a poder eliminar la columna *res*, en la funció anterior hem obert

[9] Una manera més ràpida de treure la X és amb `separate(taula_neta, any, into = "any", extra = "drop", sep = "X")`.

una *pipe* per a l'objecte `taula_neta`. Després de separar la columna d'interès, hem utilitzat la funció `select()` de *dplyr* per a eliminar la columna *res*. Finalment, hem imprès les vuit primeres files de la taula resultant.

Seleccionar el separador

Utilitzar l'argument *sep* és, en moltes ocasions, opcional. R interpretarà de manera automàtica que el separador serà un caràcter no alfabètic com un espai, un guionet, una barra baixa, etc. i l'eliminarà. En el cas que el símbol sigui un punt (`.`), ho haurem d'indicar així: `"\\."`.

El paquet *tidyr* també ens permet fer el procediment invers que acabem de veure. La funció `spread()` ens fa el contrari que `gather()`, és a dir, podem agafar els valors categòrics d'una columna i convertir-los en títols de columna. La funció que indiquem en el codi següent ens il·lustra aquest procediment, en el qual introduïm primer el nom del marc de dades i en segon lloc la columna clau del marc de dades original que conté les dades que volem que es converteixin en els títols de les noves columnes. En tercer lloc, introduirem el nom de la columna del marc de dades original que conté els valors de les columnes que crearem. Recordeu que, com en aquests processos ens referim a noms de variables existents, no ens cal especificar-ne el nom entre cometes.

```
spread(marc de dades, columna clau, columna valors)
spread(taula_neta, any, protestes)
```

De la mateixa manera que `spread()` fa el contrari que `gather()`, la funció `unite()` farà el contrari que `separate()`. És a dir, aquesta funció ens uneix les dades de dues columnes en una mateixa columna. Per defecte, `unite()` unirà els valors indicats amb un guió baix. Si volem que utilitzi un altre símbol per a unir les dades ho haurem d'indicar expressament. En l'exemple que veiem a continuació, hi ha un marc de dades creat anteriorment, `taula_sep`, que té els valors *any* i *mes* separats en dues variables diferents.

```
> taula_sep
     pais  any mes protestes
1 Espanya 2016   6        42
2 Espanya 2017   1        33
3 Espanya 2017   6        83
4  França 2016   6        42
5  França 2017   1        44
6  França 2017   6        34


unite(marc de dades, columna nova, columna1, columna2 ..., sep = "_")
unite(taula_sep, any_mes, any, mes, sep = "/")
```

Si volem ajuntar els valors de les columnes *any* i *mes* amb una barra inclinada, hem d'indicar a dins de `unite()` el marc de dades, seguit del nom de la columna que volem crear. A continuació, introduirem el nom de les dues columnes del marc de dades que volem unir. En aquest cas, no hem d'indicar el nom de les columnes entre cometes. Finalment, introduirem *sep* seguit del símbol que volem que uneixi els valors.

2.2. Convertir variables

El segon problema que podem tenir amb un marc de dades és que les variables no tinguin assignat el tipus de vector que volem. Per exemple, podem trobar que algunes de les variables categòriques que volem com a factors estiguin codificades com a vectors de caràcter o que algunes numèriques siguin dobles en lloc de nombres enters. Part del problema el podem solucionar quan descarreguem la base de dades amb atributs que tenen algunes funcions tipus `StringAsFactors = FALSE`, però és possible que un cop descarregades vulguem solucionar alguns casos.

Les funcions que hem d'utilitzar per a convertir variables ja les coneixem: `as.numeric()`, `as.character()`, `as.integer()`, `as.logical()`, etc. No obstant això, heu de tenir en compte algunes consideracions, sobretot quan convertim factors a altres tipus de variables. Com sabeu, els factors són vectors enters tractats com a variables categòriques. Com a tals, cada categoria és realment un nombre enter amb una etiqueta amb informació categòrica. Quan passem un factor a caràcter, convertirà les etiquetes a *strings* i eliminarà els nombres enters. Pel contrari, quan passem un factor a numèric convertirà els nombres enters a numèrics i eliminarà les etiquetes. En el cas que les etiquetes d'un factor siguin nombres i ens interessi guardar-los com a tal, haurem de fer una conversió doble: primer transformar el factor a caràcter per a obtenir els nombres i després transformar-lo a numèric: `as.numeric(as.character(factor))`.

Aquestes conversions no cal que les fem una per una, sinó que també les podem fer a gran escala. La funció `mutate_at()` del paquet *dplyr* permet agafar moltes variables d'un marc de dades i convertir-les de tipus en una sola operació. Suposem que volem canviar a numèric les columnes 6 a 50 del marc de dades `pol`. Introduirem `mutate_at(pol, vars(6:50), funs(as.numeric))`. Podem indicar indistintament el nom de la columna o el número de la columna.

Canviar el nom de les variables

Si volem canviar el nom de les variables, tenim dues opcions. El primer és canviar el nom de totes les variables del marc de dades. Per això hem de crear un vector amb els noms nous i incorporar-lo als noms de les columnes. En aquest exemple, modifiquem el nom de les cinc columnes de l'hipotètic marc de dades `md`:

```
colnames(md) <- c("pais", "any", "poblacio", "pib_cap", "democracia")
```

Si, al contrari, només volem canviar el nom d'una variable que es troba dins d'un marc de dades, podem indicar el número de columna entre claudàtors i inserir el nom nous dins un vector de caràcter. Per exemple, hem decidit:

```
names(md)[3] <- c("pop")
```

Hi ha algunes qüestions estilístiques que cal tenir en compte, com intentar posar tots els noms de columna en minúscules i utilitzar sempre la barra baixa (_) com a separador dins del mateix nom enlloc d'altres símbols. Per a posar tots els noms en minúscules utilitzarem `tolower(names(md))` i per a posar-los en majúscules farem servir `toupper(names(md))`.

Potser en algun moment durant la neteja del marc de dades també voleu crear una columna que sigui senzillament l'ordre de les files. Per a crear la hipotètica columna `num` en el marc de dades `md`, hauríeu d'utilitzar el codi següent: `md $num <- 1:nrow(md)`.

2.3. Eliminar files i columnes

Una part important de la neteja dels marcs de dades és eliminar les files i les columnes que sabem que no farem servir. Eliminar-les suposa més facilitat a l'hora de treballar i més velocitat per a R a l'hora de processar la informació. Com ja sabeu, *dplyr* té la funció `filter()` per a eliminar files i `select()` per a eliminar columnes. Alternativament, també podeu utilitzar els claudàtors per a suprimir les files i columnes que no ens interessin. Recordeu que per a fer una selecció d'un marc de dades, a dins del claudàtor hem d'introduir un vector amb la selecció de files i, separat per una coma, el vector amb la selecció de columnes. El símbol negatiu indica que en lloc de seleccionar-les, les volem eliminar. En el codi següent teniu diversos exemples de seleccions:

```
md <- md[-c(4, 23:50), ]
md <- md[ ,-c(14:ncol(md))]
md <- md[1000:nrow(md), -c(1, 5:7, 10:20)]
```

En el primer codi eliminem la fila 4 i les files de la 23 a la 50. En el segon exemple demanem suprimir de la columna 14 fins al final. En el tercer volem conservar de la fila 1.000 fins al final i eliminar les columnes 1, 5, 6, 7 i de la 10 a la 20. Les funcions `ncol()` i `nrow()` ens retornen el nombre total de files i columnes respectivament.

2.4. Anomalies en les dades

Quan carreguem una base de dades de la xarxa, també és habitual que tingui un contingut incomplet. No sempre totes les caselles tenen dades i de vegades aquestes dades no són correctes. En el cas de caselles sense dades (dades perdudes), haurem de detectar-les fent una observació ràpida al marc de dades un cop l'haguem descarregat. La funció més genèrica i útil per a detectar dades perdudes és la de `summary()`. En cas que hi hagi dades perdudes, R ens retornarà una fila extra a la descripció de la variable indicant el nombre de dades perdudes. Pel què fa els errors, no hi ha cap solució màgica per a trobar-los però amb un sumari podem detectar, algunes vegades, valors que s'escapen de tota lògica.

2.4.1. Dades perdudes

A R veurem les dades perdudes codificades com a NA (*Not Available*, no disponible).[10] Hi ha infinitat de raons per les quals podem tenir dades perdudes en una base de dades. En els estudis internacionals, la presència de dades està molt associada amb la capacitat de cada estat de recol·lectar-les (Stone, 2008). Així, si busquem dins del banc de dades del Banc Mundial, és molt probable que tinguem molta informació dels Estats Units però en canvi variables buides d'informació sobre països petits o en desenvolupament com Tonga o Angola. Les dades perdudes també poden ser aleatòries a causa d'errors a l'hora de codificar la informació.

[10]Cal distingir les dades perdudes (NA) dels valors infinits, senyalitzats amb *Inf*, i dels valors que no són nombres, senyalitzats amb *NaN* (*Not a Number*).

Per saber si tenim NA en el nostre marc de dades, el primer que podem fer és especificar el marc de dades en qüestió a dins de la combinació de funcions `any(is.na())`. R ens retornarà TRUE si tenim alguna dada perduda i FALSE si no en tenim cap. Si R ens diu que no en tenim cap, ja podem saltar al pas següent. Si, en canvi, R ens diu que sí que en tenim, podem demanar que compti la quantitat de dades perdudes del marc de dades amb la combinació de funcions `sum(is.na())`. R sumarà la quantitat de TRUE del marc de dades.

La funció *is.na*

Si preguntem simplement `is.na()`, R retornarà tot el mateix marc de dades però amb valors TRUE o FALSE a cada casella, de manera que tindrem marcades les dades perdudes com a TRUE i les altres com a FALSE. Aquest mètode no és recomanable amb marcs de dades grans.

Un cop ja sabem si tenim o no NA, el segon pas és localitzar on són. En això ens pot ajudar la funció `summary()`. Si apliquem aquesta funció al marc de dades obtindrem els estadístics descriptius de cada variable i, si alguna variable té dades perdudes, veurem el nombre d'NA al final del sumari de la variable. A continuació hem demanat un sumari hipotètic de la variable `md$gdp_cap`. R retorna el següent:

```
> summary(md$gdp_cap)
     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.   NA's
   -241.2   1413.1   4291.9   7215.3   9325.5 213623.1      4
```

Tal com veiem al final del sumari, aquesta variable té quatre dades perdudes. Per a identificar a quina fila es troba exactament cada un d'aquests NA, podem utilitzar la funció `which(is.na())`, que retornarà un vector numèric amb la posició de les files que tenen dades perdudes de la variable indicada.[11] Ara només ens cal visualitzar les files en qüestió seleccionant-les del marc de dades. Suposem que l'operació anterior ens ha retornat les files 34, 567, 1.234 i 1.520. Podem crear un vector amb aquests números i seleccionar-les com a files en el marc de dades.

[11]És millor fer aquesta operació amb vectors, no amb marcs de dades. Si indiquem tot el marc de dades, R retornarà el número que tenen les caselles on hi ha NA. Aquesta informació ens serà menys útil.

```
> na_md <- c(34, 567, 1234, 1520)
> md[na_md, ]
  país     continent   any   gdp_cap
1 Txad     Àfrica     1993        NA
2 Myanmar  Àsia       2005        NA
3 Senegal  Àfrica     2001        NA
```

```
4 Tonga    Oceania    1999       NA
```

Arribats a aquest punt, la gran pregunta és: què fem amb aquestes dades? Tenim dues opcions principals: eliminar la fila sencera o substituir les dades perdudes per unes altres. No hi ha una única resposta a aquesta pregunta i dependrà en bona part de la possibilitat d'esbrinar quines són les dades que falten o de la validesa dels mètodes per a imputar un valor als NA. El manual *Handbook on Constructing Composite Indicators* de l'OCDE (2008) ofereix diverses possibilitats a l'hora d'imputar nous valors. Si decidim eliminar les files, el paquet *tidyr* ens ofereix una solució ràpida amb la funció `drop_na()`, que retornarà un nou marc de dades amb les files que no tinguin valors perduts.[12]

[12]La funció `na.omit()` del paquet de base *stats* fa el mateix procediment.

Si el que volem és substituir les dades perdudes per uns valors concrets, tenim dues possibilitats:

- l'opció automàtica que ens ofereix *tidyr*,
- l'opció manual amb claudàtors.

Afegir NA a un marc de dades

És possible que trobem variables amb files buides que vulguem canviar a NA. Per a convertir-les a NA aplicarem `md $var[md$var == ""] <- NA`.

Si apliquem la funció `fill()` de *tidyr* a un vector omplirà la fila amb el valor més recent mentre que `replace_na()` farà una estimació a partir dels valors superior i inferior de la casella on tinguem l'NA.

L'opció manual consisteix a identificar la columna i el número o números de casella que tinguin dades perdudes i assignar individualment o conjuntament un nou valor. Com veiem a continuació, en el primer cas hem assignat el valor 2.045 a la fila 34 de la variable `md$gdp_cap`, mentre que en el segon cas hem aprofitat el vector `na_md` creat prèviament per a assignar el valor 2.045 a les quatre files amb dades perdudes.

```
md$gap_cap[34] <- 2045
md$gap_cap[na_md] <- 2045
```

2.4.2. Valors extrems (*outliers*)

Un altre tipus d'anomalia que podem trobar en un marc de dades són els valors extrems. Un valor extrem és un valor fora del normal en la nostra distribució. És possible que aquest valor sigui perfectament vàlid, com el cas del PIB per càpita de Kuwait en el marc de dades *gapminder*[13]. Per tant, que sigui extrem

[13]Si teniu carregat el paquet *dplyr* feu la prova: `arrange(filter(gapminder, year == 1952), desc(gdpPercap))`.

no vol dir que no sigui «real», però sempre serà motiu de sospita. Els valors extrems són normalment conseqüència d'errors en el procés de creació de les dades o fruit de codificacions especials.

Les codificacions de Polity IV

La base de dades Polity IV codifica els països en una escala de -10 a +10 segons si són més autocràtics o més democràtics. Els valors, però, prenen codificacions especials quan són règims en transició (codi -88), interrupció del règim (codi -66) o període *interregnum* (codi -77). Usar Polity IV comporta haver de prendre decisions sobre com recodificar aquests valors.

Com ja hem explicat, no hi ha cap fórmula màgica per a identificar valors extrems. El més important és fer una lectura atenta de les dades, explorar cada variable i pensar si té sentit que les dades tinguin el valor que tenen. Alguns errors són fàcils de detectar. Si sabem, per exemple, que l'esperança de vida mitjana oscil·la normalment entre 50 i 82 anys, ens sorprendrà molt si trobem un valor de 900. Difícilment, també, un país tindrà un PIB per càpita negatiu. Per tant, el més important és fer un esforç inicial per a conèixer a fons què mesuren les variables que volem treballar, quines unitats de mesura utilitzen i entre quins valors acostumen a oscil·lar aquestes unitats.

Fixem-nos bé en el sumari que hem fet en la secció anterior. El valor mínim de la variable `md$gdp_cap` és negatiu i, pel que sabem, els PIB per càpita no acostumen a ser negatius. També observem que el valor màxim supera els 200.000 dòlars per habitant i cap país acostuma a superar els 100.000. Novament podem utilitzar la funció `which()` per a identificar la fila o files que tenen valors poc habituals. En el primer cas hem seleccionat les files amb valors negatius de la variable `md$gdp_cap`, mentre que en el segon cas hem buscat els valors extrems de dues maneres: mirant quins valors són superiors a 100.000 o directament anant a buscar el nombre concret que havíem trobat a partir de la funció `summary()`.

```
md[which(md$gdp_cap < 0), ]
  pais       continent    any   gdp_cap
1 Singapur   Àsia        1991    -241.2


md[which(md$gdp_cap > 100000), ]
md[which(md$gdp_cap == 213623.1), ]
  pais      continent   any    gdp_cap
1 Canadà    Àfrica     2005   213623.1
```

Una manera alternativa d'identificar casos extrems és visualitzant la distribució d'una variable. Normalment amb `hist()` o `plot()` podrem comprovar si hi ha valors que se surten de la norma. Un cop els tenim identificats, el procediment torna a ser el mateix que per al cas dels NA.[14]

[14] Per exemple, hem trobat en una altra font que el PIB per càpita de Singapur l'any 1991 era de 11.463 dòlars. Podem utilitzar aquest valor en el marc de dades: `md$gap_cap[which(md $gdp_cap < 0)] <- 11463`.

3. Unir marcs de dades

Els analistes de dades es troben sovint que volen analitzar la relació entre dues variables, però que aquestes estan separades en dues bases de dades diferents. Per exemple, podria ser que en una base de dades tinguéssim informació sobre democràcia i en una altra tinguéssim informació sobre conflictes. Ens agradaria saber si hi ha alguna relació entre democràcia i conflicte, però per a fer-ho primer de tot hauríem d'unir aquestes dues bases de dades en una. Unir-les pot ser un procés complicat, no solament pel gran volum d'informació que pot contenir cada una d'elles, sinó per diversos entrebancs, com que la unitat d'anàlisi no coincideix o que el nom dels països que volem unir estan registrats en idiomes diferents. Normalment, davant d'aquest tipus de situació, la solució que s'acaba prenent és unir-les manualment, fila a fila, en un procés que pot tardar hores i hores. Per sort, programes com R tenen recursos per a unir taules amb certa facilitat. En aquest apartat descobrirem la segona gran virtut del paquet *dplyr*: a part de manipular dades, aquesta llibreria també té com a segona especialitat unir marcs de dades.

Mètodes per a unir marcs de dades

La funcions del paquet *dplyr* tenen una sintaxi molt intuïtiva i que pot ser aplicada a altres objectes a part d'un marc de dades. Podeu veure alguns exemples en aquest enllaç (https://dplyr.tidyverse.org/reference/join.html). A part de *dplyr*, hi ha una funció de base d'R, `merge()`, que també pot unir taules.

3.1. Ajuntar marcs de dades

Per a ajuntar dos marcs de dades amb *dplyr*, l'escenari ideal que ens agradaria trobar és que hi hagi una columna comuna de referència a les dues taules. És a dir, com veiem en l'exemple següent, el marc de dades `md_dem` (que indica si han tingut un règim democràtic en els últims 30 anys) i el marc de dades `md_war` (que indica hipotèticament el nombre de disputes militars en els darrers 30 anys) tenen una columna comuna, *country*, que té les categories amb el mateix nom. França és *France* en els dos marcs de dades, Espanya és *Spain*, i així successivament. En aquesta situació ideal, ajuntar marcs de dades és senzill amb *dplyr*.

Disponibles en l'annex

Els codis dels marcs de dades `md_dem` i `md_war` estan disponibles en l'annex.

```
> md_dem                > md_war
  country   dem           country war
1  France  TRUE         1  France  21
2   Spain  TRUE         2   Spain  13
3 Germany  TRUE         3 Germany   9
4   China FALSE         4   China  24
5   Yemen FALSE         5   Yemen  14
6   Haiti FALSE         6   Congo  42
```

A la columna que els dos marcs de dades tenen en comú l'anomenarem *columna clau* i serà la columna que ens servirà per a ajuntar les bases de dades. A l'hora d'unir-les, sempre entendrem que hi ha un marc de dades de referència (que rebrà les dades de l'altre marc de dades) i un marc de dades secundari. Suposem, en el cas que ens ocupa, que volem portar les dades de `md_war` cap

a `md_dem` i com a referència tenim la columna *country*. De la columna *country* de `md_dem` en direm *columna clau primària*, mentre que a la columna *country* de `md_war` l'anomenarem *columna clau secundària*. La distinció és important, ja que en algunes operacions R haurà de decidir si conserva alguns valors, i en aquests casos sempre conservarà els valors de la clau primària.

A la taula 4 veiem les sis funcions que estudiarem de *dplyr*. Totes elles tenen la mateixa sintaxi, formada per tres arguments dins de la funció: el nom del primer marc de dades, el nom del segon marc de dades i l'argument *by* seguit del nom de la columna clau entre parèntesi.

Taula 4. Unir marcs de dades amb *dplyr*

`left_join()`	Retorna totes les files del primer marc de dades i les files del segon marc de dades que coincideixin amb la columna o columnes clau.
`right_join()`	Retorna totes les files del segon marc de dades i les files del primer marc de dades que coincideixin amb la columna o columnes clau.
`inner_join()`	Retorna les files del primer i el segon marc de dades que coincideixin amb la columna o columnes clau.
`full_join()`	Retorna totes les files i columnes del primer i el segon marc de dades.
`semi_join()`	Retorna les files del primer marc de dades que coincideixin amb la columna o columnes clau del segon marc de dades.
`anti_join()`	Retorna les files del primer marc de dades que no coincideixin amb la columna o columnes clau del segon marc de dades.

L'única diferència entre `left_join()` i `right_join()` és que la primera utilitza el primer marc de dades com a primari mentre que la segona utilitza el segon. Amb `left_join()`, R retornarà totes les files del primer marc de dades i afegirà qualsevol fila del segon marc de dades que encaixi amb la columna clau. Amb `right_join()` R farà exactament el contrari: retornarà totes les files del segon marc de dades i afegirà qualsevol fila del primer marc de dades que encaixi amb la columna clau.

Fixeu-vos com, a continuació, unim els marcs de dades `md_dem` i `md_war` de dues maneres diferents. Amb `left_join()`, hem conservat Haití perquè era al primer marc de dades però, com que no tenim dades per a Haití de la columna *war*, ens ha retornat un NA. Amb `right_join()` conservem les dades del segon marc de dades i per tant tenim el Congo enlloc d'Haití. R ha buscat les dades de democràcia del Congo però com que no les ha trobat en la clau secundària, ens retorna un NA.

Més d'una columna clau

En la unió de dos marcs de dades podem identificar més d'una columna clau. Encara que en els exemples hem treballat amb una sola columna clau, moltes vegades n'utilitzarem dues. Treballar amb dues columnes clau és habitual, per exemple, quan tenim les dades agrupades per país i any. En aquest cas, la sintaxi serà: `xxxx_join(x, y, by = c("pais", "any"))`.

La sintaxi de les funcions *join*

La sintaxi en codi seria xxxx_join(x, y, by = "columna"). Si el nom de la columna clau no coincideix en els dos marcs de dades ho especificarem de la manera següent: `xxxx_join(x, y, by = c("nom_col1" = "nom_col2"))`. R mantindrà el nom del primer marc de dades. Si no indiquem l'argument *by* en la sintaxi, R mirarà els dos marcs de dades i combinarà les columnes que trobi amb el mateix nom. A la consola, R explicarà quin procediment ha seguit.

```
> left_join(md_dem, md_war)
  country   dem war
1  France  TRUE  21
2   Spain  TRUE  13
3 Germany  TRUE   9
4   China FALSE  24
```

```
5   Yemen FALSE  14
6   Haiti FALSE  NA

> right_join(md_dem, md_war)
  country   dem war
1  France  TRUE  21
2   Spain  TRUE  13
3 Germany  TRUE   9
4   China FALSE  24
5   Yemen FALSE  14
6   Congo    NA  42
```

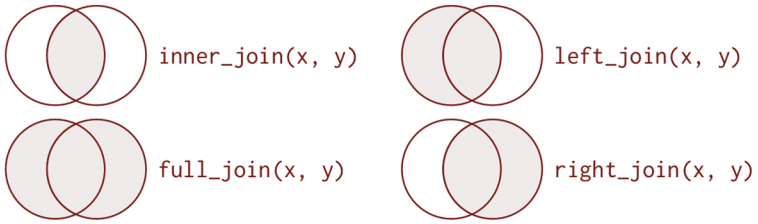
Les dues funcions que acabem de veure, `inner_join()` i `full_join()`, tenen una lògica molt semblant. En el cas de `inner_join()`, R només retornarà les files que apareguin en els dos marcs de dades, mentre que el en cas de `full_join()` obtindrem els valors que apareguin tant en un marc de dades com en l'altre. Lògicament, `full_join()` produirà molts NA. Com podem veure a continuació, el codi de l'esquerra ens uneix els marcs de dades de manera exclusiva, ja que eliminarà tot allò que no tingui informació en els dos marcs de dades, com és el cas d'Haiti i Congo. La segona funció és inclusiva, ja que conserva els valors encara que només apareguin en un marc de dades.

```
> inner_join(md_dem, md_war)
  country   dem war
1  France  TRUE  21
2   Spain  TRUE  13
3 Germany  TRUE   9
4   China FALSE  24
5   Yemen FALSE  14

> full_join(md_dem, md_war)
  country   dem war
1  France  TRUE  21
2   Spain  TRUE  13
3 Germany  TRUE   9
4   China FALSE  24
5   Yemen FALSE  14
6   Haiti FALSE  NA
7   Congo    NA  42
```

Aquestes quatre funcions de *dplyr* que acabem de veure estan resumides visualment a la figura 2. En aquesta imatge veiem la lògica de cada funció, on observem clarament com `inner_join()` és més exclusiva que `full_join()`.

Figura 2. Lògica d'unió de les funcions de *dplyr*



Font: Grolemund i Wickham (2017). *R for Data Science.* O'Reilly. CC BY-NC-ND 3.0 US.

El que hem vist fins ara amb aquestes quatre funcions de *dplyr* ha estat com unir valors de dos marcs de dades diferents en una o diverses columnes clau. A continuació, veurem dues funcions que fan una tasca semblant, però que no incorporen els valors del segon marc de dades. És a dir, el que fan és filtrar el primer marc de dades amb criteris del segon, però no incorporen els valors del segon marc de dades en el primer. Aquestes dues funcions són `semi_join()` i `anti_join()`. La funció `semi_join()` filtra el primer marc de dades segons els valors que coincideixen amb el segon marc de dades. La funció `anti_join()` fa tot el contrari que `semi_join()`: filtra el primer marc de dades segons els valors que no coincideixen amb el segon marc de dades. En el primer cas, l'únic valor que no coincidia amb el segon marc de dades és Haití, per la qual cosa l'ha filtrat del marc de dades original. En el segon cas, ha fet al revés. Com que Haití era el cas que no coincidia amb el segon marc de dades, l'ha mantingut.

```
> semi_join(md_dem, md_war)
  country   dem
1  France  TRUE
2   Spain  TRUE
3 Germany  TRUE
4   China FALSE
5   Yemen FALSE

> anti_join(md_dem, md_war)
  country   dem
1   Haiti FALSE
```

A simple vista, sembla que `semi_join()` i `anti_join()` no tinguin massa utilitat pràctica, però la realitat és més aviat el contrari. La funció `semi_join()` va bé per a conservar unes dades que compleixin uns requisits determinats. La funció `anti_join()` acostuma a ser molt útil per a diagnosticar problemes. Suposem que tenim dos marcs de dades que volem unir però que hi ha files que no ens uneix correctament. Segurament això es deu al fet que aquests marcs de dades no són exactament iguals. Les files en qüestió seran fàcils d'identificar amb un *anti join*.

Exemple per *semi_join*

Paquets de dades com el d'Eurostat donen, per una banda, la possibilitat de crear marcs de dades amb les dades que volem i, de l'altra, marcs de dades amb una relació de països segons l'organització a què pertanyen. Si, per exemple, volem dades només de l'EFTA,

podem primer generar unes dades determinades i a continuació fer un *semi_join* amb el marc de dades de l'EFTA. Així ens preservarà només les dades d'aquests països.

3.2. Combinar marcs de dades

L'agrupació de marcs de dades pot seguir una lògica diferent a la que hem explicat fins ara. Pensem en aquest supòsit: un grup de dos investigadors ha passat una enquesta a diferents estudiants universitaris. Els dos han fet les mateixes preguntes però com que no s'han coordinat a l'hora de passar l'enquesta, saben que hi pot haver estudiants que han respost tant en una enquesta com en l'altra. Cada investigador s'ha encarregat de construir el marc de dades des del seu ordinador i ara volen combinar-ne els marcs. En aquest cas tenim marcs de dades amb les mateixes variables però amb files que poden estar repetides. Vegem els marcs de dades `enquesta1` i `enquesta2`:

```
> enquesta1
  nom                 edat  P1  P2  P3  P4
1 Josep Claramunt       21   8   3   9   5
2 Josefina Curtiella    24   4   5   4   4
3 Andreu López          23   9   1   3   2
4 Matilde Llauradó      21  10   6   8   7
5 Rafa Sopena           19   1   5   3   4

> enquesta2
  nom                 edat  P1  P2  P3  P4
1 Anna Abelló           23   4   5   6   3
2 Lluís Busquets        22   5   6   7   1
3 Josep Claramunt       21   8   3   9   5
4 Albert Cusidó         23   9   1   3   2
5 Jofre Monés           20   7   6   8   4
```

La sintaxi per a combinar aquests dos marcs de dades en un de sol és molt senzilla, ja que només hem d'introduir la funció i el nom dels dos marcs de dades com a arguments. La funció `union()` ens agrupa les files dels dos marcs de dades i, en cas que hi hagi files idèntiques, en deixarà només una. La funció `intersect()` ens ajuda a trobar les files que apareixen en els dos marcs de dades mentre que la funció `setdiff()` ens retorna els valors que apareixen en el primer marc de dades però no en el segon.[15] A continuació veiem el marc de dades que retorna R en cadascuna de les operacions:

[15] Seguint l'àlgebra Booleana, la primera funció equivaldria a OR, la segona a AND i la tercera a OR menys NOT: `(enquesta1 | enquesta2) - (enquesta1 & enquesta2)`.

```
> union(enquesta1, enquesta2)
  nom                 edat  P1  P2  P3  P4
1 Anna Abelló           23   4   5   6   3
2 Lluís Busquets        22   5   6   7   1
3 Josep Claramunt       21   8   3   9   5
4 Josefina Curtiella    24   4   5   4   4
5 Andreu López          23   9   1   3   2
```

```
6 Matilde Llauradó      21  10   6   8   7
7 Rafa Sopena           19   1   5   3   4

> intersect(enquesta1, enquesta2)
  nom                 edat  P1  P2  P3  P4
1 Josep Claramunt       21   8   3   9   5

> setdiff(enquesta1, enquesta2)
  nom                 edat  P1  P2  P3  P4
1 Josefina Curtiella    24   4   5   4   4
2 Andreu López          23   9   1   3   2
3 Matilde Llauradó      21  10   6   8   7
4 Rafa Sopena           19   1   5   3   4
```

Una segona opció per a combinar marcs de dades és amb les funcions del paquet *dplyr* `bind_rows()` i `bind_cols()`, que permeten lligar marcs de dades que contenen la mateixa quantitat de files o la mateixa quantitat de columnes. La sintaxi és simple i només es tracta de posar el primer marc de dades com a primer argument i el segon marc de dades com a segon argument. La funció `bind_rows()` té un tercer argument molt útil, `.id`, que crea una columna nova que permet identificar el marc d'on provenen les dades. En l'exemple següent, hem demanat que creï una nova variable que es digui enquesta. En aquesta variable, els valors prendran el nom d'Enquesta 1 si provenen del marc de dades *enquesta1* i el nom d'Enquesta 2 si provenen del marc de dades *enquesta2*.

```
> bind_rows(Enquesta 1 = enquesta1, Enquesta 2 = enquesta2, .id = "enquesta")
   enquesta    nom                 edat  P1  P2  P3  P4
1  Enquesta 1  Josep Claramunt       21   8   3   9   5
2  Enquesta 1  Josefina Curtiella    24   4   5   4   4
3  Enquesta 1  Andreu López          23   9   1   3   2
4  Enquesta 1  Matilde Llauradó      21  10   6   8   7
5  Enquesta 1  Rafa Sopena           19   1   5   3   4
6  Enquesta 2  Anna Abelló           23   4   5   6   3
7  Enquesta 2  Lluís Busquets        22   5   6   7   1
8  Enquesta 2  Josep Claramunt       21   8   3   9   5
9  Enquesta 2  Albert Cusidó         23   9   1   3   2
10 Enquesta 2  Jofre Monés           20   7   6   8   4
```

En el paquet de base d'R, `rbind()` i `cbind()` fan una tasca molt semblant a les funcions que acabem de veure. Aquestes funcions, però, són més lentes i no retornen un *tibble*.

Resum

Aquest és un mòdul més aviat complementari per a l'aprenentatge d'RStudio, ja que no mostra directament com analitzar dades en el sentit de visualitzar variables o transformar informació per a generar estadístics descriptius útils. No obstant això, els tres apartats d'aquestes pàgines són imprescindibles per tal de dominar l'anàlisi de dades de manera autònoma. Principalment, el contingut d'aquest mòdul ens permet descarregar qualsevol base de dades disponible en organitzacions i centres d'estudis internacionals i preparar les dades per a generar el contingut que volem.

És per això que, si volem analitzar dades de forma autònoma, haurem de dominar amb agilitat els tres passos que hem après en aquest mòdul:

1) La importació de marcs de dades ens permet incorporar a R qualsevol base de dades, estigui en el format que estigui, per poder-hi treballar.

2) La neteja de marcs de dades ens permet preparar les dades per a la seva anàlisi, un pas molt important quan no obtenim les dades de fonts oficials i que, per tant, no han estat preparades prèviament. En aquest mòdul hem après a prendre decisions importants sobre quina estructura han de tenir les dades, com hem de tipificar les variables per a poder-les treballar amb més comoditat i què hem de fer amb els valors perduts i els casos extrems.

3) Finalment, un dels grans valors afegits de programes com R és la facilitat amb què poden unir diferents marcs de dades que procedeixen de fonts diferents. Aquest procés és extremadament laboriós amb altres programes i, en canvi, amb R és possible amb una senzilla línia de codi.

Exercicis d'autoavaluació

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

1. Crea el marc de dades `const_pol` demanant la variable següent de Polity IV.

```
exconst
```

2. Busca la paraula següent a la base de dades de WDI i demana els set primers resultats.

```
women
```

3. Tria la millor funció per a descarregar l'arxiu següent en format europeu.

```
preguntesexamen.csv
```

4. Descarrega el segon full d'aquest document. Tingues en compte que no hi ha una fila amb els noms de columna.

```
solucionsPAC.xls
```

5. Transforma en columnes els valors de la primera variable amb paràmetres de la segona variable.

```
wbtrade$regions, wbtrade$gdp
```

6. El marc de dades següent conté la variable *date* amb els mesos i els anys separats per un guió baix. Separa les variables.

```
wbtrade
```

7. Converteix els noms de columna del marc de dades següent a majúscula.

```
un_votes
```

8. Detecta quin valor de la variable *age* pot ser un error.

```
35, 40, NA, 42, 15, 17, 49, 65, 37, -2, 47, 21
```

9. Uneix de forma exclusiva els marcs de dades següents.

```
un_votes / un_roll_call_issues
```

10. Volem ajuntar els marcs de dades següents per la columna *country* de manera que ens filtri les files del primer marc de dades segons els valors del segon.

```
europe, efta
```

Solucionari

1. `const_pol <- PolityGet(vars = "exconst")`

2. `WDIsearch("women")[1:7,]`

3. `read.csv2("preguntesexamen.csv")`

4. `read_excel("solucionsPAC.xls", sheet = 2, col_names = FALSE)`

5. `spread(wbtrade, regions, gdp)`

6. `separate(wbtrade, date, c("month", "year"), sep = "_")`

7. `toupper(names(un_votes))`

8. `-2`

9. `inner_join(un_votes, un_roll_call_issues)`

10. `semi_join(europe, efta)`

Bibliografia

Lohr, S. (2014). *For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights*. *The New York Times*. Disponible en línia.

Stone, R. W. (2008). *The Scope of IMF Conditionality*. *International Organization* (vol. 62, núm. 4, pàg. 589–620).

Wickham, H. (2014). *Tidy Data*. *Journal of Statistical Software* (vol. 50, núm. 10, pàg. 1-23).

Annex del mòdul

Codi de l'apartat 2.1.1

```
taula_bruta <- data.frame(pais = c("Espanya", "França", "Itàlia", "Mèxic", "Japó"),
'2016' = c(64, 43, 56, 35, 12), '2017' = c(83, 34, 53, 101, 19), '2018' = c(95, 33, 67, 120, 18))
```

Codi de l'apartat 3.1

```
md_dem <- data.frame(country = c("France", "Spain", "Germany",
                                 "China", "Yemen", "Haiti"),
                     dem = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE))

md_war <- data.frame(country = c("France", "Spain", "Germany",
                                 "China", "Yemen", "Congo"),
                     war = c(21, 13, 9, 24, 14, 42))
```

Codi de l'apartat 3.2

```
enquesta1 <- tribble(~nom, ~edat, ~P1, ~P2, ~P3, ~P4,
                     "Josep Claramunt", 21, 8, 3, 9, 5,
                     "Josefina Curtiella", 24, 4, 5,  4, 4,
                     "Andreu López", 23, 9, 1, 3, 2,
                     "Matilde Llauradó", 21, 10, 6, 8, 7,
                     "Rafa Sopena", 19, 1, 5, 3, 4)

enquesta2 <- tribble(~nom, ~edat, ~P1, ~P2, ~P3, ~P4,
                     "Anna Abelló", 23, 4, 5, 6, 3,
                     "Lluís Busquets", 22, 5, 6, 7, 1,
                     "Josep Claramunt", 21, 8, 3, 9, 5,
                     "Albert Cusidó", 23, 9, 1, 3, 2,
                     "Jofre Monés", 20, 7, 6, 8, 4)
```