# Explorar, transformar i visualitzar

PID_00268325

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 familiaritzar-nos amb l'entorn R d'una manera àgil i sense massa càrrega teòrica per tal de poder aplicar ràpidament els coneixements apresos a l'ús del programari. En les pàgines següents aprendrem en pocs passos a transformar un marc de dades de dimensions considerables en informació útil i visualment atractiva. Per a aquest propòsit haurem d'aprendre les funcions bàsiques de dos dels paquets essencials d'R: *dplyr* i *ggplot2*. La llibreria *dplyr* inclou principalment funcions orientades a manipular marcs de dades. Quan diem *manipular*, no ens hi referim amb un mal sentit, sinó en el sentit d'adaptar i transformar les dades en informació útil que ens ajudi a respondre a preguntes concretes que ens vulguem formular. Manipular vol dir, per exemple, canviar l'ordre de les files d'un marc de dades sobre la base d'un criteri determinat, seleccionar una part de les files o de les columnes o bé crear columnes amb informació nova a partir de dades ja existents. Un cop haguem transformat les dades al nostre gust amb *dplyr*, la llibreria *ggplot2* ens permetrà crear visualitzacions gràfiques per a poder comunicar de manera atractiva els nostres resultats.

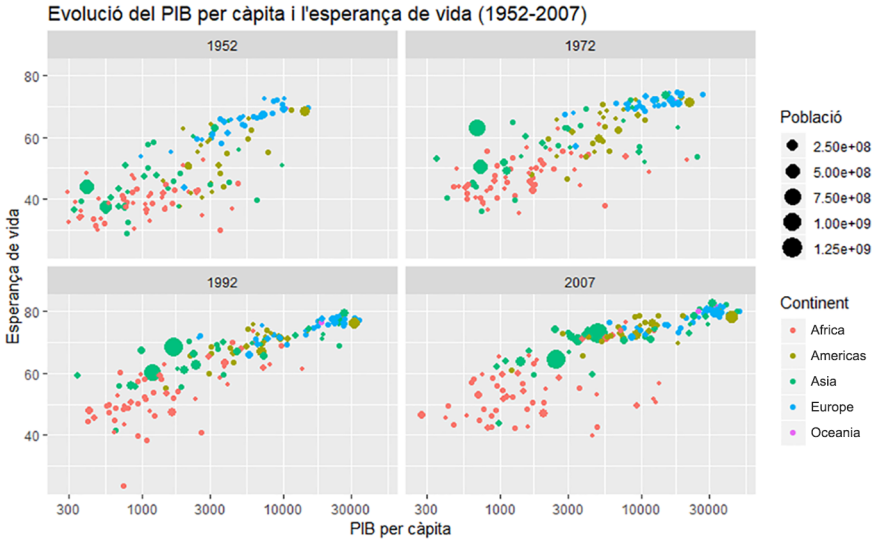
Per a aquest procés, haurem de tenir instal·lats i carregats a R *dplyr* i *ggplot2* i també el paquet *gapminder*, que utilitzarem per a fer els exercicis en aquest mòdul. El paquet *gapminder* inclou un marc de dades amb indicadors socioeconòmics com el PIB per càpita o l'esperança de vida en 142 països diferents. Un cop tingueu els paquets instal·lats i carregats, teclejant la funció `search()` podeu comprovar que efectivament *dplyr*, *ggplot2* i *gapminder* estan carregats al Global Environment d'R. Instal·larem els paquets amb la funció `install.packages()` i els carregarem, per separat, amb `library()`.

**Per saber-ne més**

No us perdeu la pàgina web de *gapminder* (www.gapminder.org ), un portal educatiu on podreu analitzar en línia una gran quantitat de dades. Tingueu en compte que quan carregueu el paquet a R no el visualitzareu a la finestra principal d'Environment com si fos un objecte que hàgim creat nosaltres. No és necessari, però si el voleu visualitzar a Environment, l'haureu de crear mitjançant `gapminder <- gapminder`.

Aquest és probablement el mòdul que canviarà definitivament la nostra percepció sobre R. Un cop finalitzem la seva lectura i hàgim treballat els exercicis, dominarem els processos bàsics de l'anàlisi de dades. En les pàgines següents utilitzarem el llenguatge d'R de manera aplicada, dominarem els instruments bàsics per a transformar i visualitzar dades i serem capaços de comunicar-nos amb el programa de manera fluida. Sense anar més lluny, al final del mòdul haurem après a construir visualitzacions com la figura 1, on serem capaços d'agrupar en un sol gràfic fins a cinc variables. Sou capaços d'identificar quines són i com està representada cada una?

Figura 1. Visualització d'un marc de dades tractat




**Les cinc variables del gràfic**. El PIB per càpita de cada país és la variable que està representada en l'eix horitzontal. En l'eix vertical, hi trobem l'esperança de vida. La tercera variable és el continent, on cada categoria té assignat un color diferent. La població de cada país està representada per la mida de cada un dels punts, de manera que països amb poca població tindran un punt petit i països grans tindran un punt més gran. La cinquena variable és l'any. Hem construït una graella de quatre gràfics que representen quatre anys diferents.

Fixeu-vos quantes preguntes podem arribar a respondre en aquesta figura:

**1)** Hi ha una relació positiva entre el PIB per càpita i l'esperança de vida?
**2)** A quin continent són més baixos l'esperança de vida i el PIB per càpita?
**3)** A quin continent hi ha els països amb més població?
**4)** A quin continent pertanyen la majoria dels països amb PIB per càpita i esperança de vida alta?

Si us fixeu, aquestes preguntes les podem contestar sense les dades numèriques, només amb una visualització ràpida de la figura anterior. També podem apreciar altres detalls, com que el 1992 tenim un cas extrem: un país africà amb una esperança de vida extremadament baixa. També veiem alguns casos, especialment a l'Àfrica però també a Àsia el 1952 i el 1972, de països relativament rics però amb una esperança de vida molt baixa. En canvi, no observem que hi hagi casos en el sentit contrari: països molt pobres però amb una esperança de vida alta.

En aquest mòdul, aprendrem a fer aquest procés: observar i explorar un marc de dades, fer-nos preguntes sobre les dades i com es relacionen entre elles, tractar-les per a poder respondre a aquestes preguntes, triar la millor visualització per a poder respondre a aquestes preguntes i finalment comunicar-les. La visualització ens pot ajudar a formular noves preguntes, que ens poden portar a investigar nous aspectes de les dades.

**Interpretar nombres com e+08**

És bastant habitual en R trobar nombres amb notació científica que acostumen a tenir el format e+. Interpretar-los és més fàcil del que sembla, ja que simplement hem de moure els decimals tantes vegades a la dreta com ens indiqui l'últim nombre, si és positiu, o tantes vegades a l'esquerra si és negatiu. Per exemple, 2.50+e08 es traduirà com a 250.000.000.

# 1. Explorar

L'exploració inicial de les dades és el primer pas que hem de fer quan tenim al davant un marc de dades (Babbie, 2013, pàg. 90; King i altres, 1994). L'objectiu principal consisteix a entendre, a grans trets, quines són la composició i l'estructura de les nostres dades. Ens interessa saber les dimensions del marc de dades, el nombre de variables que hi ha, la tipologia d'aquestes variables i també ens pot interessar observar una petita mostra de les dades. Un cop hàgim fet aquesta exploració general, el segon pas consistirà a veure les característiques més substantives de cada una de les variables d'interès de manera més específica.

**Consultar el manual d'ajuda**

En l'exploració inicial sempre és convenient fer una ullada a les instruccions, el llibre de codis o el material associat a les bases de dades. En el cas dels marcs de dades que provenen de paquets d'R podem consultar l'ajuda per mitjà, indistintament, de `?gapminder` o bé de `help(gapminder)`. En aquest petit manual, *gapminder*, hi ha una descripció de les variables que ens permet, per exemple, veure una descripció de què és i què mesura cada variable.

## 1.1. Exploració general

Com que ja hem incorporat a R el paquet *gapminder*, el primer que farem és una exploració inicial de les dades. Directament, podem imprimir el marc de dades `gapminder` per a veure a la consola una idea inicial.

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

Què observem en aquesta taula? En primer lloc, veiem que és un marc de dades en format *tibble* que conté 1.704 observacions i 6 variables. Les variables, situades a les columnes, són el país, el continent, l'any, l'esperança de vida, la població i el PIB per càpita. Aquestes variables estan formades per tipus de vectors diferents. Podem comprovar que les variables *país* i *continent* estan registrades com a factors <*fctr*>, les variables *any* i *població* estan registrades com a vector enter <*int*> mentre que les dues restants són un vector numèric o doble <*dbl*>.

En la dimensió vertical, a les files del marc de dades, hi trobem les observacions. Les observacions ens donen informació de cada un dels casos recollits en el marc de dades. Fixem-nos que, en aquest marc de dades, cada fila representa una combinació de país i any, de manera que tota la resta de variables ens mostren dades en relació amb un país determinat en un any determinat. Ens hem de fixar molt bé en quina és la **unitat d'anàlisi** del marc de dades. En el nostre cas la unitat d'anàlisi és el país, però també podria ser que cada observació fos un individu, com és el cas de les enquestes d'opinió, o bé que observéssim regions, documents legislatius, etc.

**Veure els noms de cada columna**

La funció `names()` ens permet veure els noms que prenen les variables d'un marc de dades. Només cal que introduïm el nom del marc de dades dins del parèntesi. En marcs de dades molt amples, amb moltes columnes, podem utilitzar els claudàtors per a seleccionar una part de les columnes que vulguem visualitzar. Per exemple, `names(gapminder)[2:4]` ens retornarà els noms de la columna 2 a la 4.

També podem explorar el marc de dades d'una manera més ajustada amb les funcions `head()` o `tail()`. La primera funció mostra per defecte les sis primeres files del marc de dades, mentre que la darrera ens mostra les sis últimes. Si, enlloc de sis, volem veure una quantitat diferent d'observacions, podem indicar-ho afegint dins de la funció un nou argument amb el nombre d'observacions que volem visualitzar.

**Exemple**

Si volem explorar les primeres 15 observacions, introduirem el codi `head(gapminder, 15)`. Si volem explorar les 10 últimes, escriurem `tail(gapminder, 10)`.

Altres funcions que podem utilitzar per a explorar de manera general les dades són `dim()`, per a veure el nombre de files i de columnes, o `str()` i `glimpse()`, que fan tasques exploratòries semblants.[1] Aquestes dues funcions són molt útils per a explorar marcs de dades amb moltes variables, ja que ens mostren les columnes verticalment. En lloc de veure, com és habitual, les variables en l'eix horitzontal i les observacions en l'eix vertical, amb aquestes funcions trobarem les variables distribuïdes en vertical i encapçalades pel símbol de dòlar (`$`). La raó per la qual se'ns mostra així és perquè en l'exploració inicial ens interessarà més aviat observar totes les variables, però només algunes observacions. Aquesta operació ens serà més fàcil si ens desplacem en vertical per la consola, que no té límits per a mostrar-nos la informació en format vertical però sí que ens hauria de tallar les dades si ens les mostra en horitzontal.

[1] La funció `glimpse()` pertany al paquet *dplyr*, mentre que `str()` és una funció de base d'R. Podríem considerar `glimpse()` com una versió avançada de `str()`, ja que ens retorna una taula més neta i que s'ajusta automàticament a l'amplada de la nostra consola. Proveu d'imprimir `gapminder` amb les dues funcions per veure'n les diferències visuals.

**Les dades al cap, no a la pantalla!**

La manera d'explorar marcs de dades amb R funciona diferent en comparació amb altres programes com Excel o SPSS. En lloc de visualitzar sempre el marc de dades en pantalla i desplaçar-nos de manera horitzontal i vertical, R assumeix que quan treballem amb grans bases de dades no necessitem visualitzar la taula sencera, sinó tenir una idea general del seu contingut. És per això que ens haurem d'acostumar a tenir l'estructura de les dades al cap, no a la pantalla!

A continuació, hem cridat amb la funció `str()` l'estructura de *gapminder*. Si el marc de dades tingués 20 variables més seria més còmode explorar les dades d'aquesta manera que no pas en horitzontal. Podem observar, per exemple, quantes categories hi ha a les variables definides com a factors. La variable *country* té 142 països diferents (ens indica que és un factor amb 142 nivells), mentre que la variable *continent* té cinc continents diferents (cinc nivells).

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

```
$ pop      : int  8425333 9240934 10267083 11537966 ...
$ gdpPercap: num  779 821 853 836 740 ...
```

Ara ja tenim una idea general de què conté el marc de dades *gapminder*. Durant l'exploració general hi haurà variables que ens hauran cridat més l'atenció que altres i que voldrem treballar. O bé pensem que hi pot haver observacions (països i anys concrets) que ens interessarà més analitzar.

### 1.2. Exploració específica

L'exploració específica permet obtenir més informació del contingut de les variables del marc de dades que més ens han cridat l'atenció. Com ja sabeu, per cridar una variable que es troba dins un marc de dades hem d'escriure el nom del marc de dades, seguit del símbol `$` i a continuació el nom del vector corresponent. Si ho fem així, R ens retornarà tot el vector. Quan treballem amb grans bases de dades els vectors acostumen a ser llarguíssims, de manera que si observem tot el vector no podrem treure informació rellevant. Proveu, per exemple, el codi següent:

```
gapminder$country
```

En lloc de cridar la variable directament, el que farem és demanar informació més concreta mitjançant algunes funcions. En l'exploració general hem vist que la variable *country* tenia 142 països. Podríem, per exemple, preguntar-nos quins són aquests països. O bé quins són els diferents anys en què les dades han estat capturades. La funció `unique()` ens ajuda a obtenir els valors únics d'una variable, de manera que ens serà molt útil per a variables categòriques i fins i tot per a numèriques discretes. Així, `unique(gapminder$country)` ens retornarà un vector amb els països únics i `unique(gapminder$year)` ens retornarà tots els anys de què tenim dades:

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

R ens retorna un vector en el qual observem que el primer any del qual tenim dades és 1952 i l'últim any 2007. Entre aquests dos períodes, tenim dades cada cinc anys. Podríem combinar aquesta funció amb altres funcions que ja hem vist anteriorment, com per exemple `head(unique(gapminder$year), 5)` per veure només els primers cinc anys que apareixen a la base de dades. En el cas que vulguem saber només els anys més actuals, una opció seria afegir la funció `sort()` i demanar que ens ordeni els anys en ordre descendent i a continuació que ens ensenyi els cinc valors més alts: `head(sort(unique(gapminder$year), decreasing = TRUE), 5)`. Si eliminem l'argument *decreasing* o bé el passem a FALSE veurem els cinc valors més baixos de la variable.

**Comptar casos**

Si ens fa mandra comptar quants anys únics tenim, podem demanar-li a R que ens els compti per nosaltres amb la funció `length()`. Si introduïm `length(unique(gapminder$year))` veiem que tenim 12 anys diferents.

Una funció molt utilitzada i que ens resultarà molt útil per a l'exploració específica és `summary()`. Aquesta ordre ens mostrarà un resum de l'objecte d'interès. Això vol dir que podem tant demanar el sumari sencer del marc de dades *gapminder* (ens retornarà un sumari de cada una de les seves variables) com demanar el sumari d'una variable específica. R ens ofereix un sumari diferent segons el tipus de vector. En el codi següent hem demanat un sumari de *continent*, *lifeExp* i *gdpPercap*.

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

> summary(gapminder$lifeExp)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
23.60   48.20   60.71   59.47   70.85   82.60

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

El `summary()` ens permet fer-nos una idea de com estan distribuïts els valors de la variable segons el tipus de vector amb què l'haguem codificat. Fixeu-vos que d'aquesta manera podem respondre a preguntes més específiques com per exemple:

«A quin continent tenim més dades registrades?»
«Quina és l'esperança de vida màxima d'un país?»
«I el PIB per càpita mínim?»

**Sumari diferent segons el tipus de vector**

Quan demanem un sumari, R ens retornarà un resum o un altre segons el tipus de vector. Si és un factor ens tornarà el nombre de freqüències que té cada categoria. Si és un vector de caràcter ens retornarà la longitud total de la columna i el tipus de vector. Si és una variable numèrica o entera ens retornarà estadístics descriptius, com el valor mínim, la mediana, la mitjana o el valor màxim. En el cas que hi hagi dades perdudes, la funció `summary()` ens retornarà una darrera columna amb el nombre de dades perdudes que té el vector. En aquest cas, veiem que no hi ha dades perdudes.

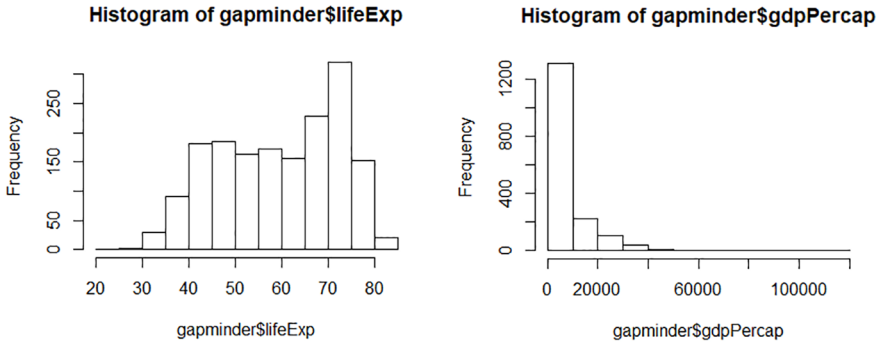
També podem fer algunes apreciacions interessants amb les dades que tenim, com que pràcticament no hi ha diferència entre la mediana i la mitjana en l'esperança de vida mentre que sí que n'hi ha en el PIB per càpita.

En darrer lloc, en l'exploració específica també podem demanar la visualització gràfica de les dades mitjançant algunes funcions dels paquets de base d'R. Segons el tipus de variable que vulguem visualitzar, utilitzarem una funció diferent. Per a una variable numèrica, utilitzarem la funció `hist()` per a visualitzar un histograma. Per les variables categòriques utilitzarem `plot()`, funció que també ens anirà bé per a visualitzar la interacció entre dues dades numèriques. Per a visualitzar la interacció entre una variable categòrica i una numèrica farem servir `boxplot()`. Proveu de reproduir els codis següents:

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

En els dos primers codis hem demanat un histograma perquè és la manera de visualitzar com estan distribuïts els valors de variables numèriques, com l'esperança de vida i el PIB per càpita. Fixem-nos en la columna més alta de la figura 2, que indica l'interval que té més observacions. Aquesta columna està situada entre 70 i 75 anys en l'esperança de vida i per sota els 10.000 dòlars en el PIB per càpita (cal tenir en compte que les observacions comencen a partir de 1952 i per tant ens trobarem molts països amb un PIB per càpita molt baix).

Figura 2. Visualització simple d'un histograma




També observem que la distribució dels valors en un histograma i l'altre és completament diferent. Mentre els casos estan distribuïts de manera prou uniforme al llarg de l'histograma d'esperança de vida, la majoria de casos del PIB per càpita estan situats a un extrem. Això explica el motiu pel qual la mitjana i la mediana són tant diferents en el PIB per càpita i tant similars en l'esperança de vida. Veurem més detalls sobre aquests estadístics descriptius més endavant, la qual cosa ens ajudarà a entendre millor la diferència entre la mitjana i la mediana. Podeu provar de fer una exploració específica dels altres gràfics que teniu en el codi anterior.

# 2. Transformar

A hores d'ara, gràcies a l'exploració inicial, ja tenim una imatge mental prou clara de l'estructura de les nostres dades. Ara que sabem quines variables tenim, de quin tipus són i com estan distribuïts els valors en cada una d'elles, és el moment de fer-nos preguntes encara més concretes sobre aquestes dades. I per això haurem d'aprendre a transformar i manipular les nostres dades. Una pregunta podria ser, per exemple, com ha evolucionat l'esperança de vida d'Espanya al llarg dels anys. També ens podríem preguntar quines diferències hi ha entre l'evolució del PIB per càpita entre Espanya i França. O bé fins i tot podríem crear dades noves a partir de les dades existents.

En aquest apartat aprendrem a transformar les dades amb les funcions del paquet *dplyr* a partir d'una sintaxi basada principalment en sis verbs: filtrar, arranjar, seleccionar, mutar, resumir i agrupar, que resumim a la taula 1.

Taula 1. Sis funcions bàsiques del paquet *dplyr*

| Objectiu | Funció | Acció |
|---|---|---|
| Manipular files | `filter()` | Elimina files. |
| | `arrange()` | Reordena les files. |
| Manipular columnes | `select()` | Elimina o reordena les columnes. |
| | `mutate()` | Crea noves columnes o en transforma d'existents amb valors construïts a partir de dades. |
| Manipular grups d'observacions | `group_by()` | Agrupa observacions. |
| | `summarize()` | Resumeix diverses observacions en una de sola mitjançant una operació. |

En les properes pàgines expliquem, una per una, cada una d'aquestes funcions. Però també hem de saber que en traurem molt més partit si les apliquem de manera simultània. Combinar funcions és una tasca visualment poc agradable, ja que hem de posar funcions dins d'altres funcions, i és complicat llegir un codi entre tants parèntesis. Si, seguint la taula anterior, volem filtrar unes dades, després crear una nova columna i finalment arranjar les dades, podem o bé crear un codi diferent per a cada verb o bé intentar afegir capes i capes de parèntesis en un sol codi.

**Obtenir les dades de PIB total**

Si ens fixem bé en les dades que tenim, veurem que podem obtenir el PIB total de cada país ja que disposem d'informació del PIB per càpita i la població. Com sabeu, el PIB d'un país és el conjunt de béns i serveis produïts per tots els habitants que viuen en un país durant un any. Com que tenim les dades del nombre d'habitants que viu en un país en un any determinat (`pop`) i el que produeixen de mitjana (`gdpPercap`), multiplicant aquestes dues dades podrem crear una nova columna on tindrem el PIB total.

**Més sobre el paquet *dplyr***

Podeu ampliar els coneixements sobre aquestes funcions consultant la secció *5. Data Transformation* que trobareu al llibre en línia (https://r4ds.had.co.nz/transform.html) *R for Data Science*.

Per sort, hi ha una altra opció per a estalviar-nos tots aquests procediments. A dins del paquet *dplyr* tenim la *pipe* (símbol `%>%`), eina que ens permet aplicar sobre el mateix objecte diverses funcions d'una manera molt més intuïtiva i ordenada. El que fa la *pipe* és canviar la gramàtica de les funcions, de manera que ens transforma una funció de diversos arguments `f(x, y)` en l'estructura `x %>% f(y)`. A primer cop d'ull costa comprendre la seva utilitat, però a través de diferents exemples n'anirem comprovant la funcionalitat. Per a il·lustrar una primera idea, en la taula 2 observem com les funcions `arrange()` i `filter()` s'agrupen a la manera clàssica i amb la *pipe*. A la manera clàssica, comencem per la funció i anem posant els arguments a dins el parèntesi. Amb la *pipe*, posem el primer argument abans d'indicar la funció. Això ens permet encadenar funcions sempre que tinguin el primer argument en comú. D'aquesta manera, totes les funcions de dins la *pipe* començaran pel segon argument.

**Més sobre la *pipe***

La *pipe* està dissenyada originalment en el paquet aquest *magrittr*. Podeu trobar més informació en aquest enllaç (https://magrittr.tidyverse.org/).

Taula 2. La funció *pipe* (`%>%`)

| | |
|---|---|
| **Clàssica i *pipe*** | `f(x, y) = x %>% f(y)` |
| **Agrupació clàssica** | `arrange(filter(gapminder, continent == "Europe"))` |
| **Agrupació amb *pipe*** | `gapminder %>%`<br>`filter(continent == "Europe") %>%`<br>`arrange(lifeExp)` |

**Crear objectes per a guardar les transformacions**

És important saber que quan transformem un objecte amb una funció, aquesta no guarda els canvis en qüestió a menys que ho indiquem expressament. A la consola veurem com quedarien aquests canvis, però R en cap cas ens modifica l'objecte. Si volem guardar les operacions haurem de sobreescriure l'objecte o crear-ne un de nou amb el símbol `<-`. Per exemple, si volem només les dades d'Alemanya, podem introduir `gapminder_germany <- filter(gapminder, country == "Germany")`. Per ara, recomanem no sobreescriure els canvis sobre el mateix marc de dades.

Podem afegir tantes funcions com vulguem a la *pipe*. Si en l'exemple volguéssim afegir una funció nova, col·locaríem una altra *pipe* després de la funció *arrange* i obriríem una línia nova amb la funció nova corresponent. Durant aquest mòdul ens anirem adonant de mica en mica de la utilitat de la *pipe*. Per ara, només intenteu retenir que és una nova manera d'estructurar gramaticalment un codi en R.

## 2.1. Filtrar

La funció `filter()` ens permet filtrar un subconjunt de les files d'un marc de dades sobre la base d'una o diverses condicions lògiques.[2] Suposem que només volem veure les observacions d'un sol país, en aquest cas d'Alemanya. Escollim "Germany", filtrem per aquest país i demanem observar només les primeres sis files amb `head()`.

[(2)]Cal diferenciar aquesta funció, que redueix o filtra les files, de la funció `select()`, que redueix o selecciona les columnes.

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

```
  5 Germany    Europe  1972    71.0 78717088 18016.180
  6 Germany    Europe  1977    72.5 78160773 20512.921
```

En aquest exemple hem filtrat només per una condició lògica, però la funció `filter()` permet fer filtres amb diverses condicions. En la majoria dels casos només caldrà que posem una coma i introduïm un nou argument amb una altra condició lògica. La coma fa de condició lògica AND, en què demanem que ens retorni només els casos que compleixen les condicions que hem assenyalat. En la primera funció del codi següent demanem a R que ens retorni els casos que compleixin a la vegada dos requisits: que siguin de l'any 1987 i que el PIB per càpita sigui superior als 25.000 dòlars (aconseguirem el mateix resultat si en lloc d'una coma hi posem el símbol `&`).

**Igual senzill (=) i igual doble (==)**

Com veieu, estem treballant amb dos tipus d'igualtat. Per a diferenciar-los, hem de pensar que l'igual doble (==) només l'utilitzarem quan estiguem comprovant les condicions d'un objecte, de manera que ens crearà, implícitament o explícitament, un vector lògic. En canvi, l'igual senzill (=) ens fa assignacions de valors o resultats com veurem més endavant.

També podem filtrar les dades d'altres maneres, com amb la condició lògica OR (símbol `|`). Com veiem, en la segona funció del codi estem filtrant per les dades que compleixin qualsevol dels dos requisits: només que les dades siguin igual o superiors a 1982 o bé que el PIB per càpita sigui superior a 25.000 dòlars ja s'inclourà en la selecció. Aquest filtre retornarà més casos que l'anterior. En el tercer codi hem separat els requisits pel símbol `&`, però aquesta vegada demanem que ens ensenyi tots els anys excepte 1982 (amb el símbol de negació `!=`) per a aquells països amb un PIB per càpita inferior o igual a 25.000 dòlars. En la darrera funció hem inclòs el símbol `%in%` per a indicar que volem filtrar per un determinat nombre de categories, que especifiquem dins un vector.

**Filtrar les dades perdudes**

R ens filtra automàticament les dades perdudes que hi ha en el vector que indiquem a dins de la funció `filter()`. En el cas que vulguem conservar les dades perdudes, ho podem indicar a dins mateix de la funció amb `is.na()`, per exemple `filter(gdpPercap > 30000 | is.na(gdpPercap)`.

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


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


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


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

Hem de tenir en compte, a l'hora de filtrar, que els factors requereixen una atenció especial. Això és degut al fet que quan filtrem les dades per mitjà d'una variable que és un factor, R elimina els valors que hem indicat però no elimina cap nivell de la variable. És a dir, si a *gapminder* filtrem pel continent Àfrica, ens quedarà un marc de dades amb només una sola categoria a la variable continent, però en canvi si demanem els nivells de la variable veurem que continua tenint cinc nivells.

```
gap_afr <- gapminder %>%
             filter(continent == "Africa")
```

```
> unique(gap_afr$continent)
Africa
Levels: Africa Americas Asia Europe Oceania
```

Per a eliminar els nivells buits haurem de crear una altra *pipe* després de la funció `filter()` i afegir la funció `droplevels()`. Cal remarcar, finalment, que podem afegir tantes condicions com vulguem sempre i quan no filtrem més del compte, ja que en aquest cas R ens retornarà un marc de dades sense cap observació.

## 2.2. Arranjar

La funció `arrange()` simplement ordena les files de manera ascendent o descendent a partir dels valors d'una columna determinada. És, doncs, probablement, la funció que té menys misteri de tots els verbs que estudiarem de la llibreria *dplyr*. Abans de tot, sabríeu intuir com està ordenat el marc de dades *gapminder*?

**Exercici 1. Arranjar dades amb *gapminder***

Imprimiu el marc de dades *gapminder* i intenteu deduir per quina columna està ordenat en primer lloc. Els valors estan ordenats en ordre ascendent o descendent? I quina és la columna que està ordenada en segon lloc?

Si hem fet l'intent, haurem pogut comprovar que *gapminder* està ordenat primer per països per ordre alfabètic descendent, ja que comença per totes les observacions que té d'Afganistan, segueix per Albània, etc. En segon lloc, el marc de dades està ordenat per any de manera ascendent, ja que comença amb dades de 1952 i va pujant fins a 2007. Si volem canviar com està ordenat *gapminder* i, per exemple, ens agradaria veure en primer lloc els països amb l'esperança de vida més baixa, teclejarem el codi següent:

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

Veiem que la majoria de casos registrats amb l'esperança de vida més baixa són de l'inici de la recollida de les dades els anys 1952 i 1957. Les dades, però, d'esperança de vida més baixes que trobem en tota la base de dades són les de Rwanda el 1992, en l'època de la guerra civil i el genocidi. Com seria si mirem les dades d'esperança de vida pel cantó dels països que la tenen més alta? Doncs hem d'especificar que volem que ens retorni la taula en ordre descendent amb la funció `desc()`:

**Arranjar i les dades perdudes**

Tant sigui en ordre ascendent com descendent, la funció `arrange()` sempre situarà els valors perduts de la variable que volem ordenar al final del marc de dades.

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

El país que ha registrat una esperança de vida més alta és el Japó, amb 82,6 anys el 2007, seguit de Hong Kong. Com veiem, però, el tercer país de la taula torna a ser Japó, de manera que el fet que tinguem barrejats esperances de vida d'anys diferents fa que tinguem duplicats països a la taula.

En aquest punt arriba la part més creativa de *dplyr* i on podrem començar a explotar realment les virtuts de la *pipe*, ja que gràcies al símbol `%>%` podrem treballar amb diverses funcions a la vegada. Suposem que només ens interessa veure les dades més recents de països que tenen l'esperança de vida alta:

**1)** Primer filtrem les observacions que compleixin els dos requisits següents: que siguin de l'any 2007 i que tinguin una esperança de vida superior als 78 anys.

**2)** A continuació, creem una nova *pipe* i demanem que ens ordeni els resultats per esperança de vida en ordre descendent.

**3)** Com que podem arranjar per tantes columnes com vulguem, demanarem que, en cas de tenir dos països amb la mateixa esperança de vida, primer ens mostri el que té un PIB per càpita superior.

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

Com veieu, mitjançant la *pipe* hem encadenat dues funcions:

**1)** Primer hem indicat l'argument *x*, en aquest cas l'objecte *gapminder*, seguit de la primera *pipe*.

**2)** En segon lloc hem introduït la primera funció amb els arguments corresponents seguit d'una altra *pipe* i la segona funció amb els seus arguments corresponents.

### 2.3. Seleccionar

Fins ara hem vist dues funcions del paquet *dplyr* que transformen les files (les filtren o les ordenen). La propera funció és `select()`, una funció semblant a `filter()` però que en lloc de reduir el nombre de files el que fa és reduir el nombre de columnes. Aquesta funció és especialment útil quan tenim marcs de dades amb un nombre enorme de variables. Per a poder treballar millor el marc de dades, ens interessarà reduir les columnes a una quantitat més manejable.

**Distingir entre filtrar i seleccionar**

Per a distingir millor una funció de l'altra, direm que filtrem per files mentre que seleccionem per columnes. Quan transformem dades, veurem que utilitzarem molt més sovint la funció `filter()` que no pas `select()`.

El marc de dades *gapminder* no té un nombre excessiu de columnes, però el seu nombre és suficient per a poder practicar els diferents usos de la funció `select()`. Fixeu-vos en els quatre codis del següent quadre:

**1)** El primer codi que veiem a continuació ens retorna un marc de dades amb només tres columnes: *country*, *year* i *lifeExp*.

**2)** En el segon codi, el símbol negatiu ens serveix per a excloure una variable i per tant ens tornarà tot el marc de dades excepte la columna *pop*.

**3)** En el tercer cas seleccionem des de la variable *country* fins a la variable *year* i, a més, també seleccionem la variable *gdpPercap*.

**4)** En el darrer cas, hem donat una utilitat una mica diferent a `select()`, ja que l'únic que hem fet és indicar un ordre de columnes diferent a l'ordre original.[3]

[3] També podem fer la selecció per nombres en lloc de per noms de variables. Per exemple, `select(1:3)` ens seleccionarà de la primera a la tercera columna.

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

gapminder %>%
  select(-pop)

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

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

A dins de `select()` podem incloure funcions, algunes resumides en la taula 3, que ens ajudaran a la selecció de les variables que volem retenir. Per exemple, si volem canviar d'ordre algunes columnes i situar-les al principi, però mantenir la resta, podem utilitzar `everything()` (per ex., `select(gapminder, continent, year, everything())`).

Taula 3. Funcions addicionals amb `select()`

| | |
|---|---|
| `everything()` | Comença amb uns caràcters determinats. |
| `starts_with()` | Comença amb uns caràcters determinats. |
| `ends_with()` | Acaba amb uns caràcters determinats. |
| `contains()` | Conté uns caràcters determinats. |

## 2.4. Mutar

El paquet *dplyr* també incorpora la funció `mutate()`, que permet transformar els valors d'una variable o crear noves variables a partir de dades ja existents. Quan mutem una columna, podem sobreescriure els canvis a la mateixa columna o bé crear una columna nova:

**1)** si el primer que especifiquem a dins la funció és el nom de la columna que volem transformar, R ens sobreescriurà els valors a la mateixa columna;

**2)** si, en canvi, indiquem el nom d'una columna nova que encara no existeix en el marc de dades, R ens mantindrà la columna original i ens crearà una columna nova amb la transformació que hem indicat.

Vegem un exemple de cada i comencem per la primera opció, que implica transformar les dades d'una columna. Si ens fixem amb la variable *pop* de *gapminder*, les dades contenen molts nombres. Suposem que ens agradaria visualitzar les dades d'una manera més simple i tenir les xifres de la variable en milions. En altres paraules, que si un país té 7.500.000 d'habitants passéssim a veure aquest valor a 7,5 (en R el separador serà un punt, no una coma). El que haurem de fer és el següent:

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

Com que, en aquest primer cas, indiquem en la funció que *pop* = *pop*, R entén que ha de substituir les dades originals de la columna *pop* per les noves dades que hem creat amb l'operació. Amb la taula que genera el codi anterior veureu com R ha dividit entre un milió tots els valors de la columna *pop*, de manera que ara veiem les dades en milions. En aquest procés, per tant, R no ha creat cap variable.

Si el que volem és mantenir la columna original i crear una variable nova a partir de la transformació mitjançant `mutate()`, indicarem com a primer argument un nom diferent a qualsevol de les columnes que conformen el marc de dades. En aquest exemple, crearem la variable `gdp`, que indica PIB total de cada país. Com que en el marc de dades tenim les dades de població i les de PIB per càpita, per saber el PIB només caldrà que multipliquem les variables *gdpPercap* i *pop*. Com que cap altra columna del marc de dades té el nom de `gdp`, R interpretarà que ha d'incorporar una nova variable en el marc de dades que fins ara no existia.

```
> gapminder %>%
    mutate(gdp = gdpPercap * pop) %>%
    head()
# A tibble: 6 x 7
  country     continent  year lifeExp      pop gdpPercap         gdp
```

```
  <chr>       <fct>     <int>   <dbl>    <int>     <dbl>        <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.  6567086330.
2 Afghanistan Asia       1957    30.3  9240934      821.  7585448670.
3 Afghanistan Asia       1962    32.0 10267083      853.  8758855797.
4 Afghanistan Asia       1967    34.0 11537966      836.  9648014150.
5 Afghanistan Asia       1972    36.1 13079460      740.  9678553274.
6 Afghanistan Asia       1977    38.4 14880372      786. 11697659231.
```

Espremem una mica més les funcions de *dplyr* i les *pipe* amb tot el que hem après fins ara. En el codi següent, hem filtrat per l'any 1952 i el continent asiàtic. Com que només ens quedarà una sola categoria en les variables *any* i *continent*, hem eliminat aquestes columnes amb la funció `select()`. Seguidament, hem mutat les dades, primer creant la variable `gdp`, i després hem calculat el percentatge sobre el PIB total de les files filtrades amb `gdp / sum(gdp)`. Finalment, ordenarem els resultats per la columna *gdp*, acabada de crear, en ordre descendent de manera que els valors més alts ens quedin al capdamunt de la taula. En aquest codi hem creat l'objecte `gap_asia`. Un cop l'haguem creat, l'imprimirem per veure'n el resultat.

**Calcular percentatges**

Anoteu bé la fórmula `x / sum(x)`, ja que ens pot ser de molta utilitat més endavant per a calcular proporcions amb les nostres dades.

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

Tingueu en compte que l'ordre en què establim les diferents funcions pot determinar-ne la taula resultant. Si haguéssim primer mutat i després filtrat les dades, la nova columna de percentatges ens hagués calculat els percentatges sobre el total de la base de dades, no sobre el total de les columnes filtrades. En canvi, com que hem mutat després de filtrar, la mutació ens l'ha fet només per als països del continent asiàtic l'any 1952. També podeu comprovar com podem cridar objectes acabats de crear. Per exemple, podem arranjar pel PIB perquè just abans hem creat aquesta columna amb `mutate()`.

## 2.5. Resumir

La funció `summarize()` ens resumeix en un sol valor les dades d'una columna a partir d'una determinada operació matemàtica. La seva funcionalitat és semblant a `mutate()`, però en lloc de treballar en horitzontal creant noves columnes, treballa en vertical, transformant diverses dades d'una mateixa columna en una de sola. A l'objecte `gap_asia` tenim una taula amb l'esperança de vida, la població, el PIB per càpita, el PIB i el percentatge sobre el PIB del continent per als països del continent asiàtic l'any 1952. Ara provarem de demanar resums d'aquest marc de dades. Calcularem:

**Calcular proporcions i casos**

Les funcions `mean()` i `sum()`, a part de treure la mitjana i sumar, també són molt útils per altres coses. Si demanem un vector lògic a dins de `mean()`, la funció ens farà la mitjana dels valors que són veritat i els que són falsos, retornant-nos, al cap i a la fi, una proporció de quin percentatge d'observacions són veritat. Si demanem un vector lògic a dins de `sum()`, sumarem els casos en què l'operació lògica sigui veritat.

**1)** el total de població al continent,
**2)** la mitjana del PIB per càpita,
**3)** la proporció d'observacions amb un PIB per càpita superior a la mitjana,
**4)** el nombre de països amb una esperança de vida superior als 60 anys i
**5)** el nombre d'observacions que tenim en la mostra.

Dins de la funció `summarize()`, separats per comes, hem creat cinc sumaris:

**1)** *tpop* ens suma la població de tots els països filtrats,
**2)** *mean_gdpcap* ens fa la mitjana,
**3)** *prop* ens calcula la proporció de països que es troben per sobre de la mitjana,
**4)** *esp* ens retorna el nombre de països amb esperança de vida superior als 60 anys i
**5)** *count* recompta les observacions filtrades.

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

R ens ha retornat un sumari per a cada operació que hem demanat. La població total era d'uns 1.400 milions d'habitants l'any 1952 a Àsia. La riquesa per habitant mitjana era d'uns 5.200 dòlars i sabem que només un 9 per cent dels països del continent estaven per sobre d'aquesta mitjana. La funció `n()`, sense res a dins del parèntesi, ha fet el recompte de 33 observacions (en aquest cas, països asiàtics l'any 1952), dels quals només quatre tenien una esperança de vida superior als 60 anys.

**Treure dades perdudes del sumari**

Si ens fixem en el codi de la mitjana, veurem que hi hem introduït l'argument `na.rm = TRUE` (significa *NA remove*, eliminar valors perduts). És important conèixer aquest argument i utilitzar-lo quan demanem estadístics descriptius, ja que, en qualsevol operació, exclourà les dades perdudes quan vulguem obtenir un sumari. Al contrari, per defecte, aquest argument és fals (`na.rm = FALSE`) de manera que en el cas d'haver-hi una sola dada perduda a la columna, R ens hagués retornat a la taula el valor *NaN (Not a Number)*.

## 2.6. Agrupar

Les dades que ens ofereix `summarize()` són molt genèriques i per a nosaltres només tenen un valor prou significatiu si podem comparar-les amb altres dades. Suposem que volem resumir les dades del codi anterior per a cada continent. Una opció, lenta i farragosa, seria aplicar la funció `filter()` per a cada continent i generar un sumari diferent cada vegada. Primer filtraríem per Àsia i faríem un sumari, després per Àfrica, després per Europa, etc. Així, després d'una bona estona, acabaríem tenint els resums per cada categoria. Una opció més ràpida és mitjançant `group_by()`, que agrupa els resums de `summarize()` segons els valors d'una variable categòrica o una numèrica discreta. Si agrupem, per exemple, per la variable *any*, R ens retornarà un sumari per a cada valor de la variable *any*.

La funció `group_by()` per ella mateixa no té cap efecte sobre la taula, sinó que s'ha d'aplicar en combinació amb una altra funció. Normalment la combinarem amb `summarize()`, però també es pot combinar amb les altres funcions de *dplyr*.

En el codi següent aprofitarem parts dels codis que hem demanat anteriorment, però en lloc de filtrar per un continent determinat, demanarem a R que ens agrupi els sumaris per continent. El que farem serà posar la funció `group_by()` amb el nom de la variable que volem que ens agrupi els sumaris. R ens retorna un sumari per a cada continent.

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

```
4 Europe     418120846       5661. 0.433     23    30
5 Oceania     10686006      10298. 0.5        2     2
```

La funció `group_by()` ens fa una agrupació prèvia de les dades abans que `summarize()` apliqui els càlculs pertinents. R ens ha retornat un marc de dades en què cada observació, tal com hem indicat, és un continent. Veiem que Àsia era el continent amb més població l'any 1952 mentre que el PIB per càpita a Oceania era el més alt (encara que només hi ha dos països en el recompte). Europa era el continent amb més països amb una esperança de vida superior als 60 anys.

**Exercici 2. Agrupar per diverses variables**

Podem agrupar amb la funció `group_by()` per diverses variables. Proveu de treure el filtre de l'any i afegiu-lo a l'agrupament amb `group_by(continent, year)` per tal que també agrupi la taula a la vegada per continent i per any. Així podreu veure una evolució temporal per a cada continent. Proveu també de posar les dades al revés per a veure com canvia la distribució de la taula: `group_by(year, continent)`.

Hem de tenir en compte que la funció `group_by()` s'aplica només en variables discretes que continguin un nombre suficient d'observacions en cada grup (com país o continent, que són categòriques, però també any, que també és discreta i té suficients observacions per a cada valor de la variable). Si apliquem la funció a variables numèriques contínues el més probable és que ens acabi retornant el mateix marc de dades.

## 2.7. Recapitulant *dplyr*

Si tenim una pregunta sobre unes determinades dades, el més probable és que amb R tinguem les eines per a contestar-la. El més complicat és saber quines eines hem d'utilitzar d'entre l'enorme quantitat de possibilitats que ofereix R. En aquesta secció hem après les funcions més essencials del paquet *dplyr*, que ens ajuden a transformar les dades de múltiples maneres per tal de respondre la majoria de preguntes que ens formulem. També hem conegut com agrupar aquestes funcions mitjançant la *pipe*, una manera més intuïtiva de fer operacions complexes amb diverses capes de funcions.

Per a tancar aquesta secció us oferim un exemple de com les sis funcions diferents de *dplyr* que hem estudiat es poden combinar en un sol codi. En primer lloc, informem R de quina és la nostra *x* i obrim una *pipe*. A continuació, filtrem per les dades de 2007 i mutem les dades per a crear la nova columna *gdp*. Seguidament, agrupem les dades per continent i demanem un sumari del país amb més població, el país amb menys població, la diferència entre el país amb més població i el país amb menys població i un recompte. Per a acabar, arrangem les dades en ordre descendent per la nova columna *diff* que ens apareixerà amb el sumari i seleccionem un ordre diferent de les columnes, de manera que ens aparegui primer el continent, després la variable *diff* i a continuació la resta de variables.

```
gapminder %>%
```

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

Evidentment, un codi amb més operacions no és necessàriament un codi millor. Un bon codi és simplement aquell que ens ajuda a transformar les dades de la manera que els resultats responguin a les nostres preguntes. I per a ferho, normalment no ens caldrà cada vegada les sis funcions de *dplyr*.

**Per a saber més sobre *dplyr***

D'ara en endavant ens serà de gran utilitat el resum (https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf) per a transformar dades amb *dplyr* que han preparat els programadors d'RStudio. També ens serà molt útil la funció d'ajuda que podem activar en qualsevol moment a través de la consola (e.g. `?dplyr`, `?filter()`, `?mutate()`, etc.).

# 3. Visualitzar

La visualització és una part importantíssima en la feina d'un analista de dades, ja que tot el treball dut a terme en el procés d'anàlisi també s'ha de poder comunicar per mitjà de gràfics atractius i fàcils d'entendre. Això vol dir que part de la informació que hem aconseguit generar mitjançant diverses funcions del paquet *dplyr* es pot quedar en un no-res si no som capaços de transmetre-la al gran públic. Massa informació en un gràfic, o un gràfic que no explica prou bé les dades amb què estem treballant, pot dificultar la llegibilitat i la comprensió de tota la nostra feina. És per això que hem de ser especialment curosos a l'hora de mostrar el producte final.

Segurament ja coneixem algunes funcions de base d'R com `plot()` o `boxplot()`, que ens permeten visualitzar les dades d'una manera ràpida i exploratòria. En aquesta secció aprendrem a crear visualitzacions de les dades més atractives, que ens serveixin no solament per a explorar sinó també per a comunicar i presentar les nostres troballes. El paquet que utilitzarem és *ggplot2*, un paquet de gràfics molt potent pensat per a poder representar estèticament les dades a partir d'una gramàtica implícita formada per diverses capes de sintaxi.[4]

[4] Aquesta gramàtica de capes està construïda a partir dels fonaments teòrics del llibre de Leland Wilkinson (1999) *The Grammar of Graphics*, que descriu i sistematitza les estructures que operen en la construcció de gràfics en l'anàlisi quantitativa.

L'estructura basada en diferents capes del paquet *ggplot2* comporta un gran avantatge per a treballar amb codi, ja que ens permetrà aplicar sempre la mateixa estructura independentment del gràfic que vulguem visualitzar. A continuació descriurem set capes, encara que n'hi ha algunes més. Per a crear un gràfic mitjançant *ggplot2* necessitem sempre, com a mínim, les tres primeres capes. El primer que haurem de fer és indicar la funció `ggplot()`. A dins d'aquesta funció indicarem les dues primeres capes.

1) En la primera capa indicarem el marc de dades.

2) En la segona capa els estètics (com es disposaran les variables en el gràfic).

3) Seguit del signe + indicarem la tercera capa, que és la geometria (quina figura geomètrica representarà les dades).

L'estructura bàsica de qualsevol visualització amb *ggplot2* la trobem resumida en la taula 4. Les primeres tres capes (base de dades, estètics i geometria) seran sempre necessàries per a produir una visualització, mentre que les altres quatre capes que explicarem en aquest mòdul (*facet*, escales, coordenades i temes) són opcionals. Hem de tenir en compte que a mesura que vulguem crear gràfics més sofisticats ens serà més necessari dominar totes les capes de *ggplot2*.

Taula 4. Estructura de capes de *ggplot2*

| |
|---|
| `ggplot(marc de dades, aes(x, y, altres estètics)) +` |
| `geometria() +` |
| `facet() +` |
| `escales() +` |
| `coordenades() +` |
| `temes()` |

**Per a saber més sobre *ggplot2***

Podeu veure un exemple de les possibilitats de *ggplot2* en aquest enllaç (http://www.ggplot2-exts.org/gallery/) o bé en la pàgina web oficial de Tidyverse (https://ggplot2.tidyverse.org/reference/). També podeu consultar els llibres de Chang (2012) i Grolemund i Wickham (2016). Per a treballar amb el paquet, ens serà molt útil aquest resum (https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf) preparat pels creadors d'RStudio.

**Errors comuns amb *ggplot2***

Quan treballem amb *ggplot2*, també és molt freqüent l'error de descriure la *x* i la *y* del gràfic sense posar-les al final de la funció d'estètics (per ex., `aes(x = var1, y=var2)`), com també el de no tancar el segon parèntesi després d'estètics. Recordeu començar amb `aes()` l'argument dels estètics i que sempre hi haurà dos parèntesis que cal tancar al final d'aquesta línia, el de `ggplot()` i el d'`aes()`.

Abans de començar a repassar l'estructura de capes, és molt important remarcar quatre aspectes fonamentals de *ggplot2*:

**1)** Fixeu-vos que el paquet es diu *ggplot2*, però la funció per crear gràfics és `ggplot()` sense el '2' final.

**2)** La funció `aes()` obre sempre la segona capa i els estètics se situen a dins del parèntesi.

**3)** A partir de la segona capa, *ggplot2* uneix les altres capes mitjançant el símbol +. Descuidar-nos-el és força freqüent i ens portarà a errors en la consola.

**4)** Finalment, tingueu també en compte que en moltes ocasions cridarem una geometria determinada però no haurem de posar res a dins del parèntesi, com `geom_point()` o `geom_abline()`, ja que estarem utilitzant implícitament els arguments per defecte associats a la geometria.

## 3.1. Les primeres capes

### 3.1.1. Marc de dades

En la primera capa simplement introduïm el nom del marc de dades que volem utilitzar. Cap misteri. Només cal tenir en compte dues consideracions:

**1)** Abans de visualitzar dades amb *ggplot2*, normalment tractarem prèviament el marc de dades amb paquets com *dplyr*.

**2)** Les funcions de *ggplot2* es poden enllaçar amb la lògica de *pipes*. Per tant, tal com mostrem al principi de la secció anterior, podem posar el marc de dades al principi del codi, fer les transformacions pertinents amb *dplyr* i tot seguit enllaçar una *pipe* amb la funció `ggplot()`. Com veiem en l'exemple següent, com que ja hem posat el marc de dades al principi del codi, situarem els estètics com a primer argument. Recordeu que a partir d'aquí les capes aniran connectades mitjançant el símbol +.

```
gapminder %>%
  filter(country == "Germany") %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_line()
```

### 3.1.2. Estètics

La segona capa de sintaxi són els estètics, representats per la funció `aes()`, que indica la manera com *ggplot2* disposa estèticament cada una de les variables que volem representar. La variable que en aquest apartat definim com a *x* estarà representada en l'eix horitzontal i la que definim com a *y* en l'eix vertical. També tenim l'opció d'incloure variables addicionals amb altres tipus d'estètics, com el color, la forma o la mida. Per exemple, fixeu-vos en el codi següent:

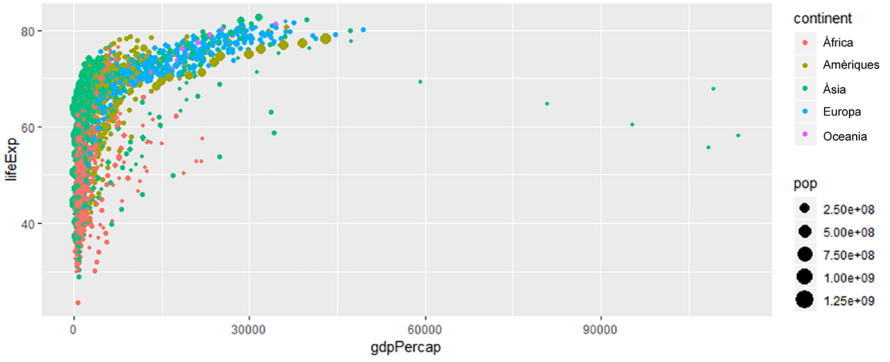
```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  geom_point()
```

Hem representat aquest codi en la figura 3. Com veieu, hem representat el PIB per càpita en l'eix horitzontal de les *x* i l'esperança de vida en l'eix vertical de les *y*. A més d'aquesta relació entre *x* i *y*, també especifiquem que volem visualitzar dues dimensions més: el color ens representarà la variable continent (cada continent serà un color) mentre que la mida ens representarà la variable de població (la mida de la geometria variarà segons el valor de la població en cada cas). Tancarem amb un doble parèntesi i introduirem el símbol + per a donar pas a la geometria, on especificarem a R que utilitzi un diagrama de dispersió per representar aquestes dades amb la funció `geom_point()`.

**Variables dependent i independent**

Normalment posarem la variable que considerem independent o explicativa en l'eix horitzontal i la variable que considerem dependent o explicada en l'eix vertical.

Figura 3. Visualització amb quatre estètics




En aquesta figura hem utilitzat un tipus de gràfic anomenat diagrama de dispersió, que és una bona manera de visualitzar una relació quan tenim variables numèriques en els eixos horitzontal i vertical. Si bé els estètics *x* i *y* seran sempre el que ens marcaran el tipus de gràfic que utilitzarem per a representar les dades, hem de saber que la utilització d'altres estètics com el color o la forma estaran molt determinats pel tipus de variable que vulguem representar. El color, per exemple, funciona molt bé amb variables categòriques com és el cas de la variable *continent*. En el gràfic podem veure que l'estètic *color* assigna automàticament un color diferent a cada continent i disposa una llegenda a la part dreta del gràfic. La mida (*size*) és més útil per a representar variables contínues, de manera que un punt més gran ens representarà un valor més alt de la variable. En la taula 5 teniu un resum dels diferents estètics que podem utilitzar segons el tipus de variable.

Taula 5. Estètics en funció del tipus de variable

| Estètic | Codi | Descripció |
|---|---|---|
| X | `x` | Eix horitzontal. |
| Y | `y` | Eix vertical. |
| Color | `col / color` | Si la variable és categòrica, ens donarà un color diferent per a cada categoria. Si és numèrica, ens farà una gradació de color segons el valor de la variable. |
| Mida | `size` | En variables numèriques, ens farà la geometria més gran o més petita (diàmetres dels punts, mida del text o gruix de les línies) segons cada valor associat. |
| Forma | `shape` | Restringit a variables categòriques. La representació per defecte és un punt, però es pot representar amb altres figures com un triangle, un quadrat, una rodona, etc. S'ha d'introduir un valor de l'1 al 25 (vegeu enllaç (https://ggplot2.tidyverse.org/reference/scale_shape.html)). Als valors de l'1 al 20 només se'ls pot canviar color. Del 21 al 25 se'ls pot canviar el color i omplir-los. |
| Omplir | `fill` | Omple les geometries de forma 21 a 25, polígons o altres figures geomètriques. |
| Transparència | `alpha` | Modifica la transparència de la geometria en un valor que varia de 0 a 1. |
| Tipus de línia | `linetype / lty` | Modifica el tipus de línia. La que correspon al valor 1 és contínua i fins a 12 són tipus de línies discontínues. |

**Quin estètic utilitzar**

En general, els estètics que funcionen millor per a variables categòriques són el color (pot representar cada punt o línia geomètrica amb un color diferent), la forma o el tipus de línia. En el cas de les variables numèriques, la mida i també el color en escala cromàtica (per exemple, blau intens indica un valor més gran i blau menys intens un valor més petit) ens representaran bé els diferents valors.

| Estètic | Codi | Descripció |
|---|---|---|
| Etiquetes | `labels` | Restringit a variables categòriques. Ens permet posar etiquetes al gràfic. |

Per a utilitzar els estètics, posarem el codi de l'estètic seguit d'un símbol igual = i el nom de la variable. Afegiu, per a fer la prova, `shape = continent` a dins dels estètics del codi anterior.

### 3.1.3. Geometria

La geometria és la tercera capa de la funció `ggplot()`, separada de les dues primeres capes amb el símbol +. Aquí indicarem l'element visual que volem fer servir per a representar les dades: un histograma, un diagrama de caixes o, com en l'exemple anterior on tenim dues variables numèriques, un diagrama de dispersió. La geometria està normalment representada per *geom* seguit d'una barra baixa i de l'objecte geomètric en qüestió: `geom_point()` pel diagrama de dispersió, `geom_histogram()` per l'histograma, `geom_bar()` pel diagrama de barres, etc.

**Geometries diferents a *ggplot2***

El paquet *ggplot2* té fins a 37 geometries diferents. En aquest mòdul, per a introduir *ggplot2*, només veurem més a fons la geometria del diagrama de dispersió. En altres mòduls d'estadística descriptiva podrem aprendre altres representacions gràfiques com el diagrama de línia, el diagrama de barres o l'histograma.

Totes les geometries estan associades amb unes característiques per defecte. Per exemple, si volem representar un punt en el gràfic, per defecte serà de color negre i tindrà una mida determinada. És per això que moltes vegades no ens caldrà especificar res a dins del parèntesi d'una geometria, ja que les característiques per defecte associades a la geometria ja ens estaran bé. Si, en canvi, volem canviar les característiques per defecte d'un determinat gràfic, com el color d'una línia o el gruix d'un punt, ho haurem d'indicar expressament a dins del parèntesi. Si volem, per exemple, un diagrama de dispersió amb quadrats blaus, més grans que la mida per defecte, i amb toc transparent, haurem de canviar la forma (*shape*), el color, la mida (*size*) i la transparència (*alpha*).

**Exercici 3. Provar combinacions**

Reproduïu el codi següent provant diverses combinacions de color ("red", "light blue", "green", etc.), mida (0.1, 1, 4 o 10), forma (0, 8, 15 o 24) i transparència (0.1, 0.3 o 0.8).

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

Hem de saber diferenciar entre els estètics que representen una variable, que indicarem normalment a la segona capa a dins de la funció `aes()` i els estètics que representen un atribut del gràfic, que indicarem a la tercera capa a dins de la geometria. En l'exemple anterior, *fill* és un estètic que ens dona informació sobre la variable *continent*. Podem fer que *fill* passi a ser un atribut de la geometria si ho posem a dins de la geometria i indiquem, per exemple, amb `fill = "yellow"`. En aquest cas, *fill* ja no ens dona informació de cap variable, sinó que passa a ser una manera de representar la geometria.

**Noms de colors**

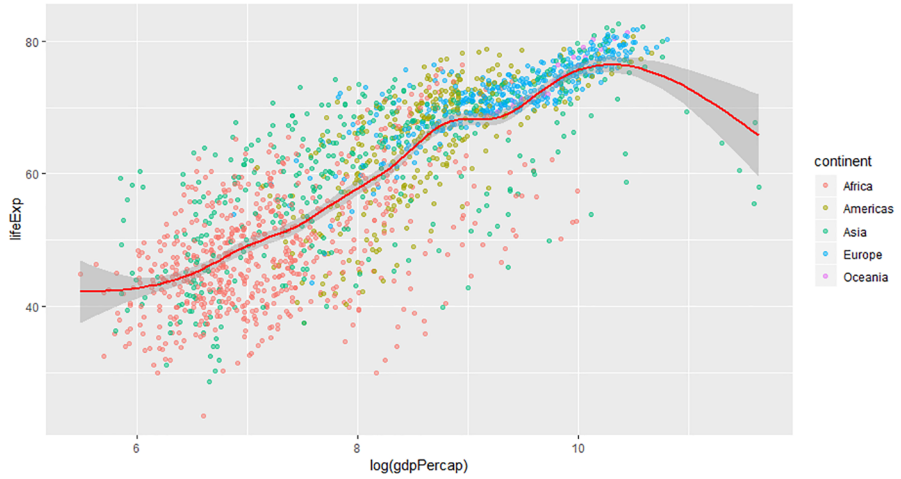
Podem anomenar els colors de centenars de maneres diferents. En aquest web (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) tenim centenars de noms de colors que podem assignar a *col* o *fill*.

També és important retenir que els estètics poden anar tant en la funció de `ggplot()` com a dins de la geometria. Posar els estètics dins de la geometria ens serà útil quan tinguem diverses geometries que tenen tots els estètics coincidents. Per a intentar aclarir una mica aquesta distinció entre la capa d'estètics i la de geometria, i entre estètics i atributs, fixeu-vos en el codi següent.

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

A dins de `ggplot()` hem assignat els estètics generals del gràfic: totes les geometries representaran a l'eix de les *x* el logaritme del PIB per càpita i a l'eix de les *y* l'esperança de vida. Volem visualitzar aquest gràfic amb dues geometries: un diagrama de dispersió `geom_point()` i una línia de regressió local `geom_smooth()`. Hi ha, però, un estètic, el color, que només estarà representat en la primera geometria i no afectarà la segona geometria. Això vol dir que veurem punts de color diferent segons el continent que representin, però no veurem cinc línies de color diferent segons cada continent. L'estètic del color, doncs, l'haurem de posar a dins de la funció `geom_point()` amb la funció `aes()`. Fora dels estètics demanarem un atribut, que no representa cap variable, sinó que és una característica de la geometria: tots els punts tindran un 50 per cent de transparència. Seguidament, amb el símbol +, afegim `geom_smooth()`, que agafarà els estètics generals *x* i *y*, però no l'específic de color per a cada continent. Sí que demanarem, en canvi, veure una línia de color vermell en lloc de la blava que ens mostraria per defecte. En la figura 4 en veiem el resultat.

Figura 4. Diagrama de dispersió i línia de regressió local



Fins ara només hem utilitzat les tres primeres capes de *ggplot2*, suficients per a crear gràfics de tota mena. Hi ha, però, altres capes que ens poden ajudar a ampliar el nombre de variables que volem representar, definir els títols que donarem a cada una de les variables o canviar altres aspectes visuals, com el tipus de lletra o el color de fons.

## 3.2. Altres capes

### 3.2.1. Facet

El nom de la quarta capa, *facet*, es podria traduir com aspecte o dimensió d'un objecte. És precisament el que fa aquesta funció. La capa *facet* permet construir múltiples gràfics a partir de les diverses dimensions d'una variable categòrica (Tufte, 1983). Per exemple, podem veure la relació entre el PIB per càpita i l'esperança de vida segons el continent.

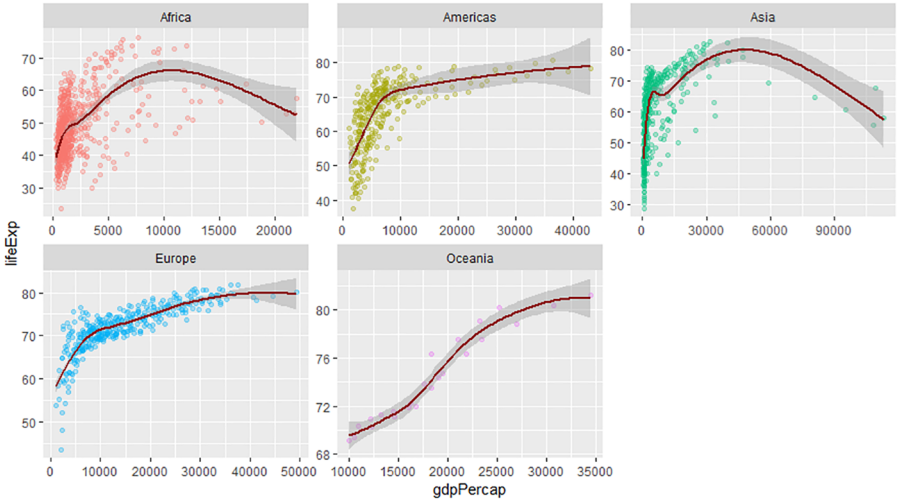
```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point(aes(col = continent), alpha = 0.3, show.legend = FALSE) +
  geom_smooth(col = "dark red") +
  facet_wrap(. ~ continent, scales = "free")
```

A dins la funció de *facet* sempre utilitzarem la titlla, el símbol ~, on indicarem si dividim els gràfics per columnes o per files. Si, com en l'exemple, la variable categòrica està situada a la dreta de la titlla, construirem els *facets* per columnes. Si està a l'esquerra, construirem els *facets* per files. Al cantó contrari, hi posarem un punt. Fixeu-vos que hem mantingut l'estètic de color per al diagrama de dispersió, de manera que els punts de cada *facet* surten pintats d'un color diferent. Hem decidit amagar la llegenda dels estètics del diagrama amb `show.legend = FALSE` perquè era redundant. Per defecte, els eixos horitzontal i vertical amb la funció *facet* ens apareixen amb la mateixa escala però en el nostre cas hem indicat que les escales siguin lliures amb `scales = "free"` de manera que cada gràfic té una escala diferent segons els seus valors (per exemple a Oceania el PIB per càpita arriba a 35.000 mentre que a Àsia sobrepassa els 90.000).

**La titlla (símbol ~)**

El símbol ~ s'anomena titlla i és una marca gràfica que en R significa: «explicat per». Trobarem aquest símbol en altres funcions d'R com `lm()` o `case_when()`.

Figura 5. PIB per càpita i esperança de vida en *facets* per continents




Una altra variant de *facet* és `facet_grid()`. A diferència de `facet_wrap()`, en aquest cas tenim l'opció de construir una graella de *facets* que comparteixin etiquetes i eixos de coordenades. Aquesta funció és molt útil quan volem afegir una variable categòrica al *facet* de les files i una variable categòrica al *facet* de columnes. Com veieu en el gràfic següent, hem creat un `facet_grid()` i hem construït una graella amb els anys a les files i el continent a les columnes.

```
gapminder %>%
  filter(year %in% c(2002, 2007)) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point(aes(col = continent), alpha = 0.3, show.legend = FALSE) +
  geom_smooth(col = "dark red") +
  facet_grid(year ~ continent)
```

Si compareu les dues *facets*, veureu que `facet_wrap()` utilitza etiquetes i coordenades en cada un dels gràfics, mentre que `facet_grid()` construeix una graella amb un nombre molt reduït d'etiquetes i coordenades.

**Estils de *facet***

En aquest enllaç (https://ggplot2.tidyverse.org/reference/facet_grid.html) podeu veure altres exemples del seu funcionament.

### 3.2.2. Escales

Per escala (*scale*) ens referim a la manera de representar, modular i mostrar informació dels diferents estètics del gràfic, en particular dels eixos horitzontal i vertical. Sempre introduirem les escales amb el mateix ordre: primer posarem el nom *scale*, seguit de guió baix, l'estètic, un altre guió baix i el nom de l'escala. Per exemple, en el codi següent modulem i mostrem nova informació sobre els eixos *x* i *y*. En primer lloc, indiquem el codi amb l'eix i el tipus de variable (*continuous* si és numèrica i *discrete* si és categòrica). A dins de la funció tenim diversos arguments per a introduir informació o modular el gràfic.

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point(aes(col = continent), alpha = 0.3) +
```

```
    geom_smooth(col = "dark red", se = FALSE) +
    scale_x_continuous(name = "PIB per càpita",
                       expand = c(0,0)) +
    scale_y_continuous(name = "Esperança de vida",
                         limits = c(20, 80),
                         breaks = c(20, 40, 60, 80),
                         label = c("Joves", "Adults",
                                 "Grans", "Vells")) +
    ggtitle("PIB per càpita i esperança de vida")
```

En aquest codi hem modificat primer les escales de l'eix de les *x* indicant que és una variable numèrica amb `scale_x_continuous()`. A dins de la funció, hem especificat els límits inferior i superior del gràfic així com "PIB per càpita" com a títol d'eix amb *name*. També amb *name*, hem anomenat l'eix vertical "Esperança de vida" i hem establert els límits superior i inferior a 20 i 80 anys amb l'argument límits. A continuació hem tallat l'eix en quatre parts amb *breaks*, i amb *label* hem assignat el nom de les etiquetes a cada tall. Finalment, hem assignat un títol general al gràfic amb la funció `ggtitle()`.

Hi ha moltes maneres de modificar les escales del gràfic. Aquí mostrarem com establir els límits del gràfic i com indicar les etiquetes:

**1)** Establir els límits del gràfic: Amb l'argument `expand = c(0,0)` aconseguim que els límits d'un eix s'ajustin el màxim possible a les dades. També dins de les funcions *scale* podem utilitzar l'argument *limits*, on indicarem els límits inferior i superior. Fora de les funcions *scale*, també podem establir els límits amb `xlim()` i `ylim()`, per exemple `xlim(20, 85)` per a l'esperança de vida. Aquestes dues funcions són una manera més ràpida de marcar els límits del gràfic, però només funcionen quan no utilitzem les funcions *scale*. Hem d'anar en compte amb les conseqüències de canviar els límits del gràfic, ja que R ens pot eliminar part de la informació que mostrarà visualment.

**2)** Establir les etiquetes del gràfic: Podem indicar les etiquetes del gràfic tant a dins com a fora de les funcions *scale*. Una manera més ràpida de definir el nom de les escales és amb la funció `labs()`. Per exemple, `labs(x = "Nom x", y = "Nom y", col = "Nom Color")`. De la mateixa manera, també podem eliminar els noms de les escales amb `labs(x = NULL, y = NULL)`.

**Per a saber-ne més**

Trobareu més informació en aquest enllaç (https://ggplot2.tidyverse.org/reference/scale_continuous.html) per a variables numèriques i en aquest enllaç (https://ggplot2.tidyverse.org/reference/scale_discrete.html) per a variables categòriques. També podem definir les escales de la resta d'estètics com el color, la mida o la forma.

#### Paletes de colors

A l'escala és on també indiquem els colors que ens representaran els estètics. Proveu d'afegir l'escala següent en el gràfic anterior:

```
scale_color_brewer(palette = 1, direction = -1, type = "div")
```

En aquesta funció estem dient que mostri els colors de la paleta 1. Podeu anar canviant el número i veureu diverses paletes, que la direcció dels colors sigui a l'inversa (podeu indicar 1 o -1) i que mostri colors divergents. Si les categories són ordinals és millor utilitzar diferents tons d'un mateix color, que aconseguirem amb `type = "div"`. És possible que vulguem colors completament diferents, que indicarem amb `type = "seq"`. Una altra opció que tenim és eliminar *direction* i *type* i indicar a la paleta una de les combinacions que R té disponibles, com `"PiYG"`, `"Spectral"`, `"Set2"` o `"Purples"`.

Podeu consultar més colors a `?scale_fill_brewer` o `?brewer.pal`, o bé al web de Color Brewer (http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3).

A les escales també podem modular les unitats amb les quals visualitzem la informació. Això ens pot ser útil en el cas de tenir variables numèriques amb casos extrems o valors esbiaixats cap a un cantó de la distribució. Una deformació típica de les dades és l'escala logarítmica, que transforma un eix de manera que cada canvi d'unitat en l'eix representa un canvi de *x* vegades la unitat inicial. Per exemple, si afegim `scale_x_log10()` estarem multiplicant per 10 cada canvi d'unitat en les dades originals.

**Altres opcions d'escala**

A part de l'escala logarítmica, també podem convertir l'escala en l'arrel quadrada de la variable amb `scale_x_sqrt()` o capgirar la variable amb `scale_x_reverse()`.

### 3.2.3. Coordenades

Aquesta capa controla les dimensions del gràfic. Fins ara, tots els gràfics que hem visualitzat tenen dues dimensions i s'anomenen coordenades cartesianes. Una dimensió és horitzontal, representada per l'eix de les *x*, i l'altra la vertical, representada per l'eix de les *y*. Hi ha, però, altres tipus de coordenades com per exemple les coordenades polars, que ens possibiliten visualitzar diagrames circulars com el del codi següent:

**Girar les coordenades**

Aquesta capa la farem servir molt poc. Potser la funció més útil és `coord_flip()`, que intercanvia les dimensions cartesianes del gràfic de manera que *x* passa a ser representada en l'eix vertical i *y* passa a ser representada en l'eix horitzontal.

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

Cal dir que el diagrama circular no és molt recomanable, ja que és difícil avaluar i comparar la mida relativa de cada segment. És a dir, entre dos segments de mida semblant però diferent serà complicat identificar quin és més gran. Si mirem la visualització del codi anterior, no podem saber si hi ha més població a Àfrica o Amèrica. En canvi, amb un diagrama de barres són més fàcils d'apreciar aquestes petites diferències.

### 3.2.4. Tema

Tota la part visual que no té a veure amb les dades s'introdueix a la capa de tema. Aquí és on podem canviar el color de fons, el tipus de lletra de cada element, la mida de les lletres o qualsevol altre element del gràfic.[5] En aquest manual us suggerim possibles temes que R té predissenyats. Per defecte, R sempre us mostrarà `theme_gray()`, però podeu canviar-lo si afegiu un nova capa a la funció `ggplot()`, per exemple, `theme_classic()`, `theme_bw()`, o `theme_dark()`. Podeu baixar-vos nous temes com el que utilitzen *The Economist* o *The Wall Street Journal* mitjançant el paquet *ggthemes* carregant el paquet `library(ggthemes)`.

[5] Trobareu més informació de com canviar cada un dels elements a la pàgina de la funció theme (https://www.rdocumentation.org/packages/ggplot2/versions/3.1.0/topics/theme).

## Resum

Amb tot el que hem après en aquest mòdul ja estem preparats per a poder-nos enfrontar a qualsevol base de dades. Primer hem après les eines que tenim per a fer una exploració inicial en un marc de dades, que ens han ajudat a prescindir de la visualització clàssica de les bases de dades. Ara ja no necessitarem tenir les dades visualment disponibles a la pantalla, sinó que amb l'exploració general hem après a fer-nos una idea al nostre cap de l'estructura de les dades i les variables que són del nostre interès. També ens hem fet una idea de les característiques de les variables més rellevants del marc de dades per mitjà d'una exploració més específica i una visualització ràpida de les dades.

El més important, però, per a un analista de dades és dominar la transformació de les dades amb paquets com *dplyr*. Mitjançant les sis funcions principals d'aquest paquet hem après a fer diverses manipulacions en el marc de dades *gapminder* per a preparar les dades per una visualització posterior. Finalment, també hem vist una petita part de les enormes possibilitats que ens ofereix el paquet *ggplot2*, que ens permet crear una gran quantitat de gràfics basats en un sistema de capes.

Amb tot, ara ja estem preparats per a crear el codi que ens permet construir el gràfic que hem vist a la figura 1 d'aquest mòdul.

```
gapminder %>%
  filter(country != "Kuwait",
         year %in% c(1952, 1972, 1992, 2007)) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, col = continent,
             size = pop)) +
  geom_point() +
  scale_x_log10() +
  facet_wrap(~ year) +
  labs(x = "PIB per càpita", y = "Esperança de vida",
       col = "Continent", size = "Població") +
  ggtitle("Evolució del PIB per càpita i l'esperança de vida (1952-2007)")
```

Fixeu-vos que aquí tenim fins a cinc dimensions de les dades:

**1)** el PIB per càpita,
**2)** l'esperança de vida,
**3)** el continent,
**4)** la població i
**5)** l'any.

Fixeu-vos que hem tret el país Kuwait perquè es tractava d'un cas extrem que ens distorsionava la visualització de les dades. Amb el símbol `%in%` hem filtrat quatre valors de la variable any, que ens serviran per a fer un *facet* amb *ggplot2*. Per a una millor visualització de les dades hem escalat la *x* a logaritme. Una bona manera de practicar és crear gràfics nous a partir de modificacions de codi que podeu fer en els exemples d'aquest mòdul.

## Exercicis d'autoavaluació

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

**1.** Tecleja el codi per a imprimir el marc de dades següent.

```
worldbankdata
```

**2.** Volem veure les últimes set files i les primeres set columnes del marc de dades següent.

```
imfdata
```

**3.** Quina és la funció de *dplyr* que retorna un resultat semblant a aquesta funció?

```
str()
```

**4.** Tecleja el codi per a visualitzar la variable següent del marc de dades `imfdata`.

```
regions
```

**5.** Volem veure els noms únics, ordenats de la Z a la A, de la variable següent del marc de dades `imfdata`.

```
regions
```

**6.** Visualitza un histograma de la variable següent del marc de dades `eurostat`.

```
social_expenditure
```

**7.** Transforma la funció següent en una *pipe*.

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

**8.** Detecta l'error de la funció següent.

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

**9.** Filtra la variable següent per als anys 1980, 1985 i 1990.

```
wbdata$any
```

**10.** Filtra per països que tinguin bombes nuclears (vector lògic) o despesa militar superior al quatre per cent del PIB (vector numèric, escala de 0 a 1).

```
militarydata$nuclear / militarydata$despesa_pib
```

**11.** Arranja el vector següent en ordre descendent.

```
militarydata$despesa_pib
```

**12.** Selecciona les primeres tres columnes del marc de dades següent i totes les columnes que acabin amb la paraula "europe".

```
tradedatabase
```

**13.** La columna següent està en escala de 0 a 1. Muta-la per tal de veure les dades en tant per cent.

```
militarydata$despesa_pib
```

**14.** Quines funcions calen per a resumir el marc de dades següent segons els valors d'una variable categòrica?

```
eurostat
```

**15.** Agrupa les dades per a la primera variable i resumeix la suma dels valors de la segona.

```
militarydata$nuclear / militarydata$despesa_pib
```

**16.** Demana un diagrama de dispersió amb les variables següents com a *x* i *y*.

```
militarydata$num_guerres / militarydata$despesa_pib
```

**17.** Canvia el color a vermell de la geometria següent.

```
geom_smooth()
```

**18.** Posa transparència del 50 per cent en la geometria següent.

```
geom_point()
```

**19.** Indica quin seria l'argument per a omplir la geometria de color groc següent.

```
geom_bar()
```

**20.** Construeix un `facet_wrap` per files amb la variable categòrica següent.

```
militarydata$nuclear
```

# Solucionari

**1.** `worldbankdata`

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

**3.** `glimpse()`

**4.** `imfdata$regions`

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

**6.** `hist(eurostat$social_expenditure)`

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

**8.** L'igual (=) hauria de ser doble (==)

**9.** `wbdata %>% filter(any %in% c(1980, 1985, 1990))`

**10.** `militarydata %>% filter(nuclear == TRUE | despesa_pib > 0.04)`

**11.** `militarydata %>% arrange(desc(despesa_pib))`

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

**13.** `militarydata %>% mutate(despesa_pib = despesa_pib * 100)`

**14** .`group_by()` i `summarize()`

**15.** `militarydata %>% group_by(nuclear) %>% summarize(mean(despesa_pib))`

**16.** `militarydata %>% ggplot(aes(x = num_guerres, y = despesa_pib) + geom_point()`

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

**18.** `geom_point(alpha = 0.5)`

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

**20.** `facet_wrap(nuclear ~ .)`

# Glossari

**`%>%`** Símbol *pipe* que replica el primer argument d'una funció com a argument de les funcions següents (*dplyr*).

**`%in%`** Condició lògica que indica una selecció de categories.

**`arrange()`** Reordena les files d'un marc de dades (*dplyr*).

**`boxplot()`** Visualitza en un diagrama de caixes la relació entre una variable categòrica i una variable numèrica.

**`coord_polar()`** Canvia les coordenades cartesianes per coordenades polars (*ggplot2*).

**`dim()`** Veiem les dimensions del marc de dades, primer les files i després columnes.

**`droplevels()`** Elimina els nivells buits d'un factor.

**`ggtitle()`** Mostra el títol principal del gràfic (*ggplot2*).

**`glimpse()`** Mostra una estructura de les dades més neta que `str()` i adaptada a les dimensions de la consola (*dplyr*).

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

**`facet_wrap()`** Crea una graella de *facets* reduint els eixos i les coordenades (*ggplot2*).

**`filter()`** Elimina les files d'un marc de dades (*dplyr*).

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

**`ggplot()`** Crea una visualització gràfica (*ggplot2*).

**`group_by()`** Agrupa observacions d'un marc de dades (*dplyr*).

**`head()`** Retorna les sis primeres files del marc de dades.

**`hist()`** Permet visualitzar la distribució d'una variable numèrica.

**`install.packages()`** Instal·la llibreries a R.

**`labs()`** Mostra les etiquetes del gràfic (*ggplot2*).

**`library()`** Mostra les llibreries o carrega una llibreria determinada.

**`mutate()`** Crea noves columnes o en transforma d'existents amb valors construïts a partir de dades (*dplyr*).

**`names()`** Mostra el nom de les columnes d'un marc de dades.

**`plot()`** Permet visualitzar les freqüències d'una variable categòrica o bé visualitzar la relació entre dues variables numèriques.

**`unique()`** Retorna valors únics d'un vector.

**`scale_*_****()`** Prepara les escales d'un eix (*ggplot2*).

**`search()`** Mostra els paquets carregats a R.

**`select()`** Elimina o reordena les columnes d'un marc de dades (*dplyr*).

**`summarize()`** Resumeix diverses observacions en una de sola mitjançant una operació (*dplyr*).

**`summary()`** Resumeix una variable i retorna un sumari específic segons cada tipus de vector.

**`str()`** Mostra l'estructura de les dades, amb les variables a les files, el tipus de variable i les primeres observacions de cada variable.

**`tail()`** Retorna les sis darreres files del marc de dades.

# Bibliografia

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

**Chang, W.** (2012). *R Graphics Cookbook*. Canadà: O'Reilly. http://www.cookbook-r.com/Graphs/

**Grolemund, G.; Wikcham, H.** (2016). *R for Data Science*. Canadà: O'Reilly. https://r4ds.had.co.nz/

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

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

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