Contents

2º paso: preparar variables e limpar datos

Unha vez que os datos están cargados na memoria do ordenador e necesario organizalos en función do que pretendamos facer con eles, e incluso limpalos ou modificalos, posto que poden aparecer diferentes tipos de situacións ou problemas neles que debemos de resolver.

Para unha persoa que comece pode ser máis rápido realizar as manipulacións previas nunha folla de cálculo e deixar para R o máis simple do proceso. Aínda así, cando se vai adquirindo habilidade, compensa o esforzo de facelo con código, posto que **permite recordar nun futuro os pasos aplicados**, e **automatizar o proceso para novos grupos de datos** de características similares, algo difícil de facer cunha folla de cálculo.

```
#ler datos
datos=read.csv2("csv/leite.csv")
```

Os datos que se van manexar nesta parte son unha submostra dos microdatos de **2014** da **EPF, base (2006)**, procedentes do INE.

Un primeiro paso será **coñecer que temos**, que variables e de que tipo son as que aparecen dentro do data.frame que se formou a partir do ficheiro.

Maneiras rápidas e simples son ver os nomes das variables (***names()***), observar as 6 primeiras filas (***head()***) ou as 6 derradeiras (***tail()***)

```
names(datos)
```

```
##  [1] "leite.enteiro" "leite.non"     "NUMERO"        "CCAA"
##  [5] "TAMAMU"        "FACTOR"        "NMIEMB"        "NMIEM7"
##  [9] "NMIEM8"        "NUMESTU"       "UC1"           "TIPHOGAR8"
## [13] "EDADSP"        "SEXOSP"        "IMPEXAC"       "GASTMON"
```

```
head(datos)
```

```
##   leite.enteiro    leite.non NUMERO CCAA TAMAMU   FACTOR NMIEMB NMIEM7
## 1    26.5224364 120.67929979    298   14      1 559.8471      2      0
## 2     0.0000000   0.00000000    768    1      5 598.1511      4      0
## 3     1.1054263   1.55906600    903   12      3 793.9561      2      0
## 4    85.1987253 246.63805448   1017    9      1 827.3677      6      0
```

```
## 5     0.4249617   1.08457226   1130   12      1 579.8876      2      0
## 6   111.3457321   0.02346163   1349   15      5 447.9655      1      0
##   NMIEM8 NUMESTU UC1 TIPHOGAR8 EDADSP SEXOSP IMPEXAC  GASTMON
## 1      0       0  17         1     74      1   26796 18674.15
## 2      2       0  29         3     46      6   18000 11981.74
## 3      0       1  17         4     58      6   16500 36530.47
## 4      2       0  41         4     62      1   35880 19394.78
## 5      0       0  17         1     78      1   14520 15665.73
## 6      0       0  10         2     48      1   14124 19905.42
```

```
tail(datos)
```

```
##     leite.enteiro  leite.non NUMERO CCAA TAMAMU    FACTOR NMIEMB NMIEM7
## 95        0.00000   0.000000  20520   10      2 1834.7996      1      0
## 96        0.00000   0.000000  20639   12      1  700.0412      2      0
## 97        0.00000 222.289964  20948   12      5  594.8689      2      0
## 98       48.97095  38.763542  21308    9      1  862.4924      3      0
## 99      276.85480   5.631425  21815    1      1 1067.6498      3      1
## 100      37.02139 198.658856  21821   12      2 1806.2414      2      0
##     NMIEM8 NUMESTU UC1 TIPHOGAR8 EDADSP SEXOSP IMPEXAC   GASTMON
## 95       0       0  10         2     43      1   13200  8391.678
## 96       0       0  17         1     85      1    9744  2974.116
## 97       0       0  17         1     81      1   14460  9612.696
## 98       0       1  24         4     55      1   67272 80251.644
## 99       0       0  22         3     38      6   14232 19012.003
## 100      0       0  17         2     39      1   20988 16488.182
```

Máis información obténse co comando ***str()***, que describe a estructura dun obxecto:

```
#como son as variables?
str(datos)
```

```
## 'data.frame':    100 obs. of  16 variables:
##  $ leite.enteiro: num  26.522 0 1.105 85.199 0.425 ...
##  $ leite.non    : num  120.68 0 1.56 246.64 1.08 ...
##  $ NUMERO       : int  298 768 903 1017 1130 1349 1476 1742 1790 1929 ...
##  $ CCAA         : int  14 1 12 9 12 15 13 12 2 6 ...
##  $ TAMAMU       : int  1 5 3 1 1 5 1 5 1 5 ...
##  $ FACTOR       : num  560 598 794 827 580 ...
##  $ NMIEMB       : int  2 4 2 6 2 1 3 3 1 2 ...
##  $ NMIEM7       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ NMIEM8       : int  0 2 0 2 0 0 1 1 0 0 ...
##  $ NUMESTU      : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ UC1          : int  17 29 17 41 17 10 22 22 10 17 ...
##  $ TIPHOGAR8    : int  1 3 4 4 1 2 3 3 2 1 ...
##  $ EDADSP       : int  74 46 58 62 78 48 39 41 61 65 ...
##  $ SEXOSP       : int  1 6 6 1 1 1 6 1 6 1 ...
##  $ IMPEXAC      : int  26796 18000 16500 35880 14520 14124 32472 32640 10020 15552 ...
##  $ GASTMON      : num  18674 11982 36530 19395 15666 ...
```

Neste caso indícanos que son 100 observacións de 16 variables. Unha parte delas son do tipo *integer* (equivalente aos números enteiros en matemáticas) e outra de tipo *numeric* (equivalente aos números reais).

A información sobre as variables está incluida no anexo.

Revisando esa información vemos que 4 das variables *integer* son en realidade **variables categóricas**, e os seus valores só representan diferentes categorías.

Neste caso o apropiado para R é manexar datos de tipo **factor**, polo que imos facer o cambio.

```
#transformar en factores
datos$CCAA=factor(datos$CCAA)
datos$TAMAMU=factor(datos$TAMAMU)
datos$TIPHOGAR8=factor(datos$TIPHOGAR8)
datos$SEXOSP=factor(datos$SEXOSP)
```

A función ***factor()*** transforma vectores numéricos ou de caracteres en factores.

Cal é a diferencia entre un factor e un vector de caracteres?

A diferencia está na forma que R ten de almacenalos. **Nun vector de caracteres R almacena o vector "como está"**, se ten palabras almacena cada palabra enteira.

O factor en cambio asignaría un número a cada palabra diferente e almacenaría ese número e a lista indicando a que palabra corresponde cada un. Cada palabra desa lista será un **nivel** do factor

Un exemplo sería un factor que, a diferencia do noso exemplo, almacenase **as autonomías usando os nomes** e non un número de referencia. Nese caso o número de referencia asignaríase igual, pero almacenaría unha copia dos nomes (**niveis**) para que R identificase cada un deles.

Esta estructura afecta ao seu manexo, xa que considera que **os niveis son unha lista pechada** na que non entran nin desaparecen novos niveis, a non ser que se lle pida directamente.

Esta primeira análise vaise facer sobre o **consumo total de leite** e a porcentaxe que iso representa do gasto do fogar.

Esas variables non existen, polo que se van crear a partir das existentes:

```
#crear novas variables
#operando con vectores
leite=datos$leite.enteiro+datos$leite.non #gasto total en leite
porcentaxe=round(100*leite/datos$GASTMON,2)  #% do gasto total en leite respecto do gasto total
```

Para crear estas dúas variables estamos utilizando operacións con vectores. En R manexa unha **aritmética de vectores**, que, simplificadamente, pódese describir dicindo que permite facerlle as mesmas operacións que se fosen números, operando posición a posición (1º con 1º, 2º con 2º,...) e obtendo como resultado un vector co mesmo tamaño contendo os resultados das operacións realizadas.

No caso de que os tamaños sexan diferentes R actúa "reciclando" as posicións, e nos vectores máis pequenos sigue operando pero collendo de novo os valores do principio. Unha consecuencia é que se pode sumar un vector cunha constante e o resultado será o mesmo vector con todos os elementos aumentados nesa constante

```
z=c(2,2,1,3)
z+2
```

```
## [1] 4 4 3 5
```

Outras transformacións prodúcense extraendo subconxuntos de elementos dun vector ou dun data frame.

Para o exemplo imos considerar unicamente os fogares que gastaron ***1 euro ou máis en leite***. Iso implica transformar as variables creadas e as incluidas no data frame, extraendo (ou eliminando) os fogares que non cumpran esa condición.

Para un vector sería crear unha nova versión (ou un novo vector) indicándolle como índice a condición que deben cumprir:

```
vectornovo=vectorvello[condición]
```

Para o noso exemplo a condición ***gastar 1 euro ou máis en leite*** escríbese ***leite>=1**, xa que o obxecto **leite** contén o gasto total en leite de cada fogar.

Aplicaríase esa condición aos obxectos **porcentaxe** e **leite** creando novas versións:

```
#usar unicamente fogares que consumen
porcentaxe=porcentaxe[leite>=1] #significa: gardar en porcentaxe, aqueles valores de
#porcentaxe que ocupen as mesmas posicións que ocupan os valores de leite maiores ou iguais a 1
```

Para data frames hai que ter en conta que se manexan 2 dimensións: filas e columnas.

Para crear unha nova versión do data.frame gastos hai que escoller fogares con gasto de leite >= 1, significa seleccionar as filas (1ª dimensión) do data frame que coincidan coas filas de leite que verifican esa condición.

```
#usar unicamente fogares que consumen
gastan=datos[leite>=1,]
```

A segunda dimensión,as columnas, déixase en branco para indicar que queremos todas as variables (todas as columnas)

Podemos comprobar agora cantos fogares quedan preguntándolle a R a dimensión do data frame gastan:

```
dim(gastan)
```

```
## [1] 85 16
```

Obtense un vector de tamaño 2, o primeiro elemento será o número de filas (nº fogares) e o segundo o número de columnas (nº de variables)

O data.frame **gastan** será semellante ao data.frame **datos** pero unicamente con fogares que gastan en leite 1 euro ou máis

De xeito semellante podemos construir un data.frame cos fogares que non gastan, cambiando unicamente a condición:

```
#separar non gastan
non.gastan=datos[leite<1,]
```

Queda por modificar o obxecto **leite**

```
#obxecto leite para os que si gastan:
leite=leite[leite>=1]
```

O seguinte obxecto creado vai ser un factor, que indique cales fogares son galegos e cales non. Pódese facer creando un obxecto novo, co longo igual ao número de fogares que gastan, e despois completalo cunha etiqueta para os fogares galegos e outra para os demais:

```
tam=length(leite) #numero de fogares que gastan en leite

#Creando un novo factor que indique os fogares galegos e non galegos
lugar=rep(NA,tam)

lugar[gastan$CCAA==12]="gal"
lugar[gastan$CCAA!=12]="outro"
lugar=factor(lugar)
```

O comando

rep(algo,nveces)

crea un obxecto novo repetindo *algo nveces*; deste xeito creouse o obxecto **lugar**, que repite **NA** (lugares vacios) tantas veces como o tamaño do obxecto *leite* (gardado en *tam*)

Nos seguintes comandos vaise enchendo posicións, primeiro aquelas nas que o fogar é galego (CCAA=12) > lugar[gastan$CCAA==12]=“gal”

e despois aquelas que non o é.

lugar[gastan$CCAA!=12]=“outro”

Finalmente dóuselle a **lugar** unha estructura de factor.

lugar=factor(lugar)

ANEXO

VARIABLES incluidas no ficheiro leite.csv

- **leite.enteiro**: Importe total del gasto en leche entera
  - b,1-9999
- **leite.non**: Importe total del gasto en leche semi y desnatada
  - b,1-9999
- **NUMERO**: Número secuencial que indica el orden del hogar en el fichero
  - 00001- 22146
- **CCAA**: Comunidad autónoma de residencia
  - 1 Andalucía
  - 2 Aragón
  - 3 Asturias, Principado de
  - 4 Balears, Illes
  - 5 Canarias
  - 6 Cantabria
  - 7 Castilla y León
  - 8 Castilla – La Mancha
  - 9 Cataluña
  - 10 Comunitat Valenciana
  - 11 Extremadura
  - 12 Galicia
  - 13 Madrid, Comunidad de
  - 14 Murcia, Región de
  - 15 Navarra, Comunidad Foral de
  - 16 País Vasco
  - 17 Rioja, La
  - 18 Ceuta
  - 19 Melilla
- **TAMAMU**: Tamaño del municipio
  - 1 Municipio de 100.000 habitantes o más
  - 2 Municipio con 50.000 o más y menos 100.000 habitantes
  - 3 Municipio con 20.000 o más y menos de 50.000 habitantes
  - 4 Municipio con 10.000 o más y menos de 20.000 habitantes
  - 5 Municipio con menos de 10.000 habitantes
- **FACTOR**: Factor poblacional
  - Cualquier valor, distinto de b y de 0
- **NMIEMB**: Número de miembros del hogar
  - 1-20
- **NMIEM7**: Número de miembros del hogar menores de 4 años
  - 0-19
- **NMIEM8**: Número de miembros del hogar entre 5 y 16 años
  - 0-19

- **NUMESTU**: Número de estudiantes hogar
  - 0-20
- **UC1**: Tamaño equivalente del hogar. Escala OCDE 1 + 0,7 * (NMIEM1 - 1) + 0,5 * NMIEM2
  - 1-150
- **TIPOHOGAR8**: Tipo de hogar (octava clasificación)
  - 1 Persona o pareja (al menos uno de los miembros) de 65 o más años
  - 2 Otros hogares con una persona o pareja sin hijos
  - 3 Pareja con hijos menores de 16 años o adulto con niños menores de 16 años
  - 4 Otros hogares
- **EDADSP**: Edad (calculada a fecha de cumplimentación de la ficha de hogar) Nota: Ampliado el intervalo en 2011
  - 85 Personas de 85 ó más años
  - 16-84 Resto de personas
- **SEXOSP**: Sexo
  - 1 Hombre
  - 6 Mujer
  - -9 No consta
- **IMPEXAC**: Importe exacto de los ingresos mensuales netos totales del hogar
  - 0-99999 Importe
- **GASTMON**: Importe total del gasto monetario anual del hogar elevado temporal y poblacionalmente. (para el salario en especie se contabiliza sólo el importe del pago realizado por el hogar)
  - b,1-9999999999999999

ANEXO

Vectores lóxicos

Os vectores poden conter distintos tipos de cousas, sempre e cando en cada vector haxa un único tipo.

Un tipo de vectores útil o formado por valores loxicos: **TRUE, FALSE, e NA** ("non dispoñible""). Ou as súas abreviaturas **T e F**, respectivamente.

Os valores lóxicos **formanse a partir de condicions**. Por exemplo:

```
18 > 13      #isto é unha condición que compara se 18 é maior ca 13, e en caso
```

```
## [1] TRUE
```

```
#de ser certo responde TRUE; se fose falso respondería FALSE

18<13
```

```
## [1] FALSE
```

Para realizar condicións en R existen operadores lóxicos **<, <=, >, >=, == (igual) e != (diferente)**.

```
A=6
B=12

#agora operar con A e B:
A<B
```

```
## [1] TRUE
```

```
A<=B
```

```
## [1] TRUE
```

```
A>B
```

```
## [1] FALSE
```

```
A>=B
```

```
## [1] FALSE
```

```
A==B # A e B son iguais?
```

```
## [1] FALSE
```

```
A!=B # A e B son diferentes?
```

```
## [1] TRUE
```

Ademais existen operacións para expresions lóxicas: a **intersección** e a **unión**. Se c1 e c2 son expresións lóxicas podemos ***intersecalas* &** (***and***) (**c1 & c2** equivale a **c1 e c2**) , ***unilas* |** (**or**) (**c1|c2** equivale a **c1 ou c2**) ou ***negalas* !** (**negation**) (**!c1** equivale a **o contrario de c1**).

No traballo con vectores as expresións lóxicas permiten extraer partes do vector que cumpren unha característica determinada:

```
y=c(3,2,2,1,7,6,0)
    y2=y[y>4] #en y2 gardaranse os elementos de y que son maiores ca 4
    y2
```

```
## [1] 7 6
```

```
  y5=y[!(y>4)] #en y5 gardaranse os elementos de y que NON son maiores ca 4
    y5
```

```
## [1] 3 2 2 1 0
```

ANEXO

Producindo sucesións de valores

R ten diferentes ferramentas para xerar sucesións de números. Unha utilidade básica para estas sucesións sería escoller uns elementos determinados dun vector, por exemplo os 2 primeiros elementos:

```
x1=c(3,2,4,4,3)
x1[1:2]
```

```
## [1] 3 2
```

ou os elementos de posición par:

```
y=c(10.4, 5.6, 3.1, 6.4, 21.7,0,10.4, 5.6, 3.1, 6.4, 21.7)
y[2*1:5]
```

```
## [1] 5.6 6.4 0.0 5.6 6.4
```

Comezamos coas diferentes posibilidades:

- **:**

```
1:30
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

son os **valores desde 1 ata 30** seguidos, ou sexa, equivale a **c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10,. . . , 29, 30)**.

O símbolo ***:*** indica unha sucesión que comeza no número previo e remata no posterior.

Se o multiplicamos por unha constante os valores que produce esa expresión aparecen saltando de un a outro o valor da constante, por exemplo:

```
2*1:15
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30
```

En R os obxectos poden funcionar como variables que gardan constantes. Por exemplo, creamos unha constante n e asignámoslle o valor 10, e despois podemos usala para crear sucesións

```
n = 10
1:n
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Se estamos traballando cun editor de texto podemos reutilizar ese código simplemente asignando outro valor á constante:

```
n = 6
1:n
```

```
## [1] 1 2 3 4 5 6
```

Tamén se poden facer operacións a esa constante e colocalas na expresión da sucesión, pero hai que ter coidado cunha cousa, a expresión debe ir entre paréntese, xa que se non R entendera que se opera con toda a sucesión.

Obsérvese os dous diferentes resultados:

```
1:n-1
```

```
## [1] 0 1 2 3 4 5
```

```
1:(n-1)
```

```
## [1] 1 2 3 4 5
```

As sucesions feitas ata agora foron crecentes, pero é fácil construilas decrecentes, simplemente con algo así:

```
30:1
```

```
##  [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8
## [24]  7  6  5  4  3  2  1
```

- **seq()**

Esta función da máis posibilidades para crear unha sucesión. Ten cinco argumentos:

```
from=value #é o punto de inicio da sucesión
to=value #é o punto de remate da sucesion.
```

Empregando estes dous parámetros e ignorando os demais xa se pode crear unha sucesión to tipo **inicio:final**

```
seq(from=2,to=12)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12
```

```
#que é igual a
seq(2,12)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12
```

```
# ou tamén a
seq(to=12,from=2)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12
```

Se non se especifica nada os saltos son unitarios. Pero isto pode modificarse cos seguintes parámetros:

```
by=value #indica cantas unidades debe saltar en cada elemento da sucesión
length=value, #indica o tamaño que queremos para a sucesión, o que  obriga a seq() a
# dar saltos que se adapten a ese tamaño de vector.
```

```
seq(2,19,by=2)
```

```
## [1]  2  4  6  8 10 12 14 16 18
```

```
seq(2,19,length=4)
```

```
## [1]  2.000000  7.666667 13.333333 19.000000
```

Estes dous parámetros usanse por separado, non ten sentido colocalos no mesmo comando posto que poderían entrar en contradición.

O quinto parámetro ten a particularidade de que se usa cun obxectivo específico:

```
along=vector # usase para construir unha sucesión do tipo 1:length(vector)
```

A expresión correcta sería: **seq(along=vector)** onde *vector* é o nome dun obxecto vector

```
seq(along=y)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11
```

Como se veu nestes exemplos os parámetros das funcións non é necesario poñelos todos, e incluso ás veces non é posible, posto que algúns poden ser contradictorios.

Outra cuestión sobre os parámetros é escribirlle o nome ou non. Non é obrigatorio, posto que R interpretaos en función da posición que ocupan, con todo, penso que o recomendable sería non poñer os máis habituais (*from* e *to*, neste caso) pero colocar sempre os demais, así cométense menos despistes.

Máis exemplos desta función:

```
seq(-5, 5, by=.2) -> s3
s3
```

```
##  [1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4
## [15] -2.2 -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4
## [29]  0.6  0.8  1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6  2.8  3.0  3.2
## [43]  3.4  3.6  3.8  4.0  4.2  4.4  4.6  4.8  5.0
```

```
#ou tamén:
s4 <- seq(length=51, from=-5, by=.2)
s4
```

```
##  [1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4
## [15] -2.2 -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4
## [29]  0.6  0.8  1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6  2.8  3.0  3.2
## [43]  3.4  3.6  3.8  4.0  4.2  4.4  4.6  4.8  5.0
```

- **rep()**

Úsase para repetir un determinado número de veces un obxecto. A forma máis simple é:

```
rep(x, times=5)
```

```
##  [1] 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4
## [15] 21.7 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7
```

ou o que é o mesmo

```
rep(x, 5)
```

```
##  [1] 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4
## [15] 21.7 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7
```

que crea un vector con x replicado 5 veces. Un parámetro alternativo é:

```
rep(x, each=5)
```

```
##  [1] 10.4 10.4 10.4 10.4 10.4  5.6  5.6  5.6  5.6  5.6  3.1  3.1  3.1  3.1
## [15]  3.1  6.4  6.4  6.4  6.4  6.4 21.7 21.7 21.7 21.7 21.7
```

que repite cada elemento de **x** cinco veces antes de pasar ao seguinte.