Contents

PRIMEIROS PASOS

Antes de comezar, é moi conveniente ter datos, codigos e os ficheiros necesarios dentro do mesmo cartafol, e indicarlle a R que ese vai ser o cartafol de traballo que empregaremos.

PREVIO:

Configurar o cartafol de traballo

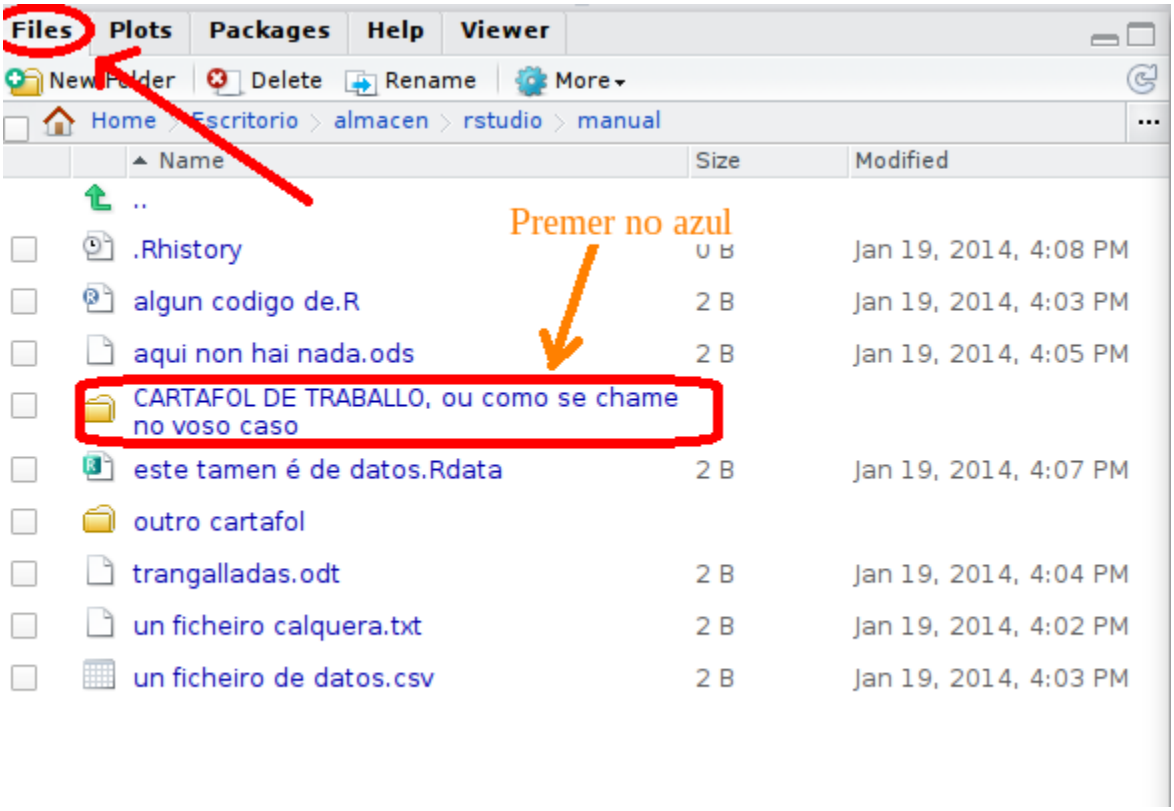
É algo que se debe facer sempre, ao inicio de calquera sesión de traballo con R

Por que se necesita un cartafol de traballo? **R** busca os ficheiros e gárdaos sempre no mesmo cartafol, que se denomina o seu **cartafol ou directorio de traballo**.

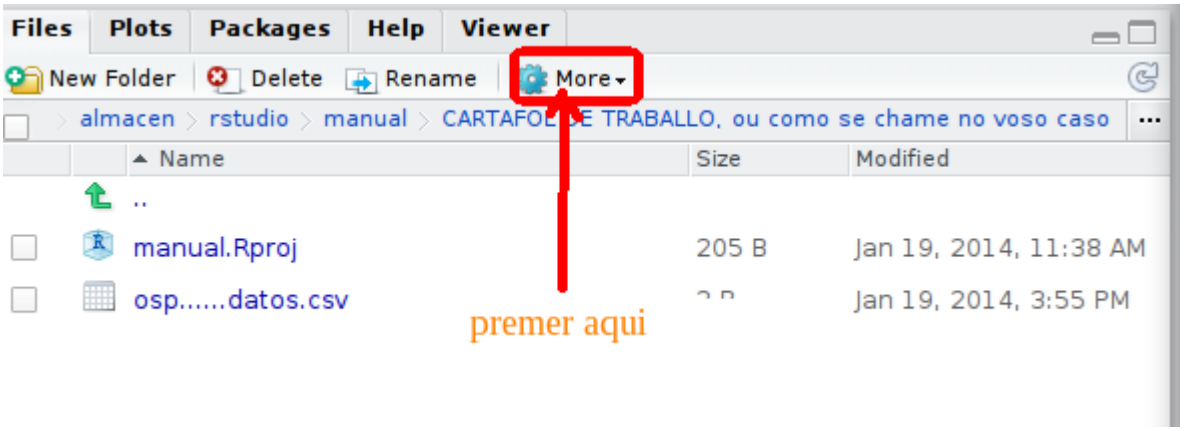
O directorio que é escolle depende do sistema operativo, e tamén de da maneira de abrilo.

Por iso, se temos os datos, o código, outras cousas nun directorio, debemos indicarllo. Ademais vai ser aí onde garde todo o que lle mandemos gardar.

Indicar a R cal é o cartafol de traballo Unha vez teñamos decidido cal é o cartafol de traballo, se estamos con RStudio debemos buscalo no **xestor de ficheiros** e acceder a el:



Unha vez esteades dentro del, usando o **menu do xestor de ficheiros** premedes en **More**



e despois escolledes a opción **Set as Working Directory**

Con eses pasos, xa tedes configurado o cartafol de traballo

PASO 1: Cargar datos

O primeiro paso para traballar con R é introducirlle os datos, e o máis habitual será facer que os lea desde algún tipo de ficheiro.

R manexa comandos para ler ficheiros de datos de moi diferentes formatos e procedentes de diferente tipo de programas, pero o máis cómodo e o que garante mellores resultados e lelos desde **ficheiros de texto**, organizando as variables en forma de columnas, o que habitualmente se coñecen como **ficheiros csv**

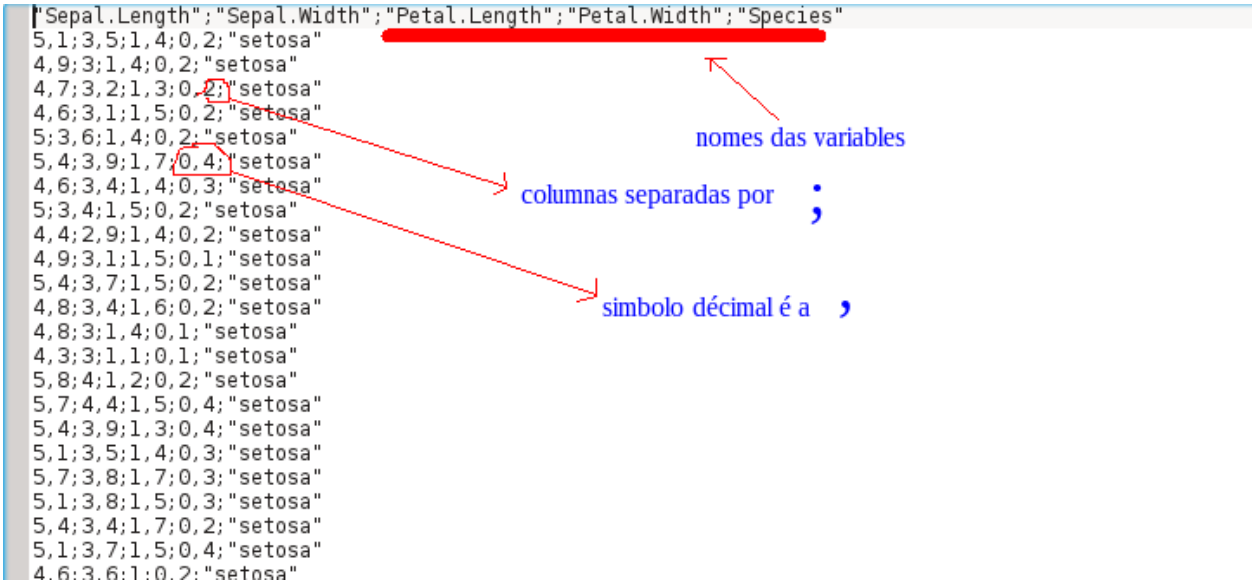
Que son os ficheiros csv? Son **ficheiros de texto** organizados para conter variables.

Organízanse como unha **folla de cálculo**, con filas e columnas, onde cada columna representa unha variable.

```
"Sepal.Length";"Sepal.Width";"Petal.Length";"Petal.Width";"Species"
5,1;3,5;1,4;0,2;"setosa"
4,9;3;1,4;0,2;"setosa"
4,7;3,2;1,3;0,2;"setosa"
4,6;3,1;1,5;0,2;"setosa"
5;3,6;1,4;0,2;"setosa"
5,4;3,9;1,7;0,4;"setosa"
4,6;3,4;1,4;0,3;"setosa"
5;3,4;1,5;0,2;"setosa"
4,4;2,9;1,4;0,2;"setosa"
4,9;3,1;1,5;0,1;"setosa"
5,4;3,7;1,5;0,2;"setosa"
4,8;3,4;1,6;0,2;"setosa"
4,8;3;1,4;0,1;"setosa"
4,3;3;1,1;0,1;"setosa"
5,8;4;1,2;0,2;"setosa"
5,7;4,4;1,5;0,4;"setosa"
5,4;3,9;1,3;0,4;"setosa"
5,1;3,5;1,4;0,3;"setosa"
5,7;3,8;1,7;0,3;"setosa"
5,1;3,8;1,5;0,3;"setosa"
5,4;3,4;1,7;0,2;"setosa"
5,1;3,7;1,5;0,4;"setosa"
4,6;3,6;1;0,2;"setosa"
```

Cousas a ter en conta

- Levan nomes de variable ou non?
- Símbolo que indica a separación das columnas
- Símbolo decimal



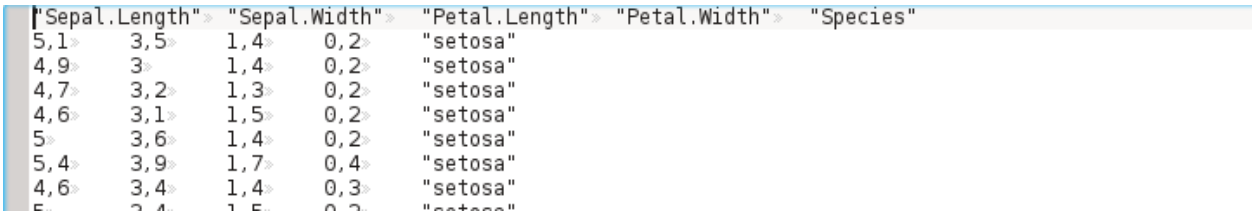
Levan nomes de variable? Non é obrigatorio, pero si frecuente, que na primeira fila vaian os nomes das variables.

Están separados polo mesmo símbolo que indica a separación das columnas

Símbolo de separación de columnas Hai varios diferentes, e de feito pode ser calquera, sempre que o programa que os vai abrir o permita.

Un símbolo moi usado é o punto e coma (**;**), son os que eu uso para as clases, e que en **R** consideran como estandar para o comando **read.csv2()**. É apropiado para usar cando o símbolo de decimal é a coma (**,**).

Outro símbolo a usar (ainda que non se ve) é o **tabulador**. Con el as columnas no texto vense ben organizadas.



Tamén se pode usar a coma (**,**) para separar as columnas, pero nese caso hai que ter coidado co simbolo dos decimais. O habitual é usar a coma cando o separador decimal é un punto (comando **read.csv()** no R)

Simbolo decimal Como separan os decimais:

- coma: (**,**)
- punto: (**.**)

Como crear ficheiros csv? O tipo de programas máis habitual para manexar datos son as follas de cálculo, polo que se vai explicar como conseguir ficheiros csv desde dúas delas: o **calc** de LibreOffice, e o **excel** de Microsoft Office

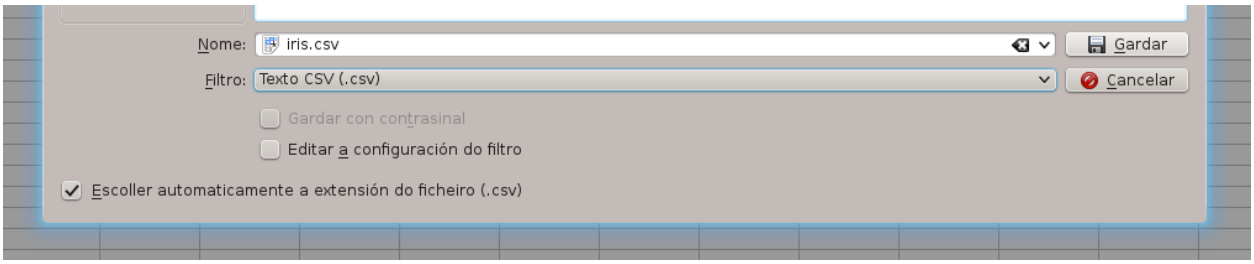
Os preparativos previos son os mesmos para os dous programas, e consisten en organizar as variables que queremos gardar no noso ficheiro:

- Poñer unha primeira liña con nomes de variables (non é obrigatorio)
- Poñer os datos de cada variable nunha columna debaixo do nome que lle corresponda na primeira fila

Algo así:

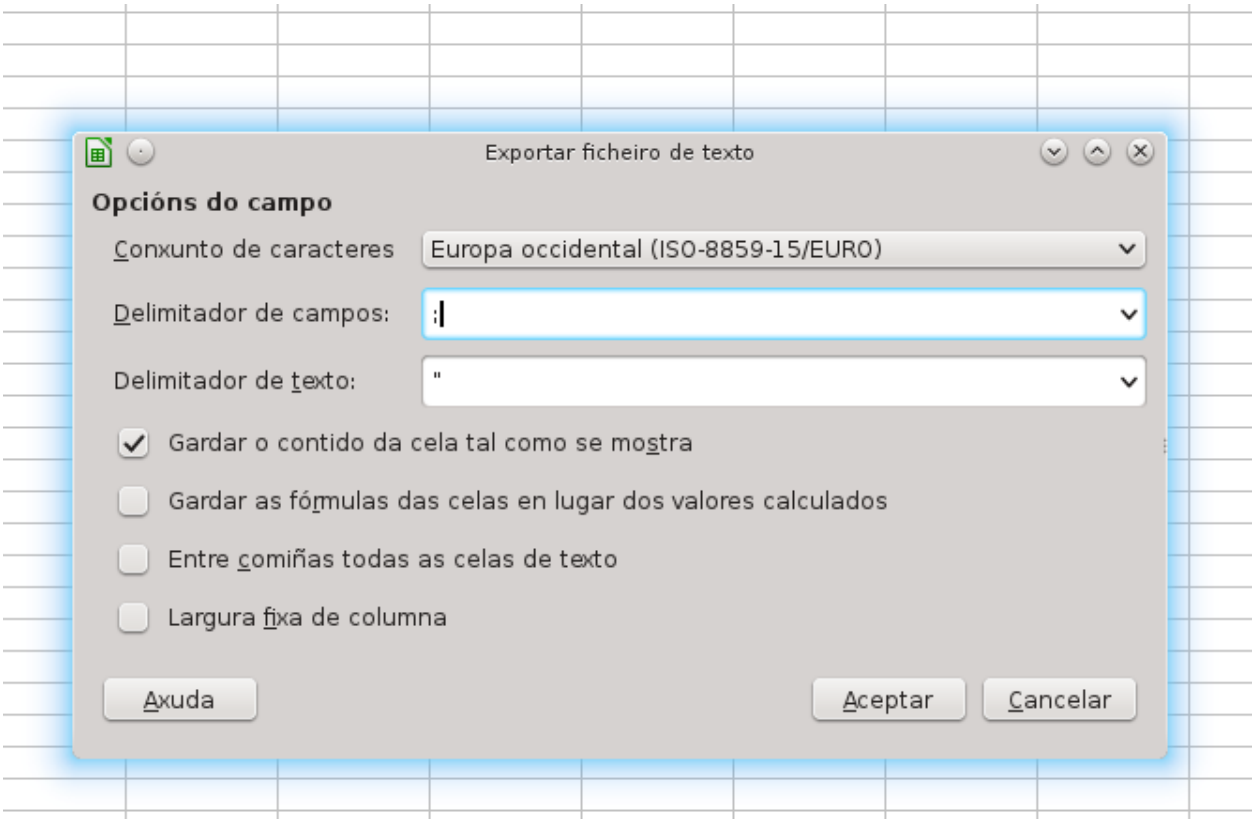
	A	B	C	D	E	F	G	H
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species			
2	5,1	3,5	1,4	0,2	setosa			
3	4,9	3	1,4	0,2	setosa			
4	4,7	3,2	1,3	0,2	setosa			
5	4,6	3,1	1,5	0,2	setosa			
6	5	3,6	1,4	0,2	setosa			
7	5,4	3,9	1,7	0,4	setosa			
8	4,6	3,4	1,4	0,3	setosa			
9	5	3,4	1,5	0,2	setosa			
10	4,4	2,9	1,4	0,2	setosa			
11	4,9	3,1	1,5	0,1	setosa			

Gardar csv con Calc (LibreOffice) Se escolledes no menu **Gardar como** davos a opción de escoller o tipo de formato no que gardar. Escolledes **Texto CSV**.



Despois premedes **gardar**, nese momento é posible que vos volva a preguntar se queredes **Utilizar o formato de texto CSV**. Aceptades.

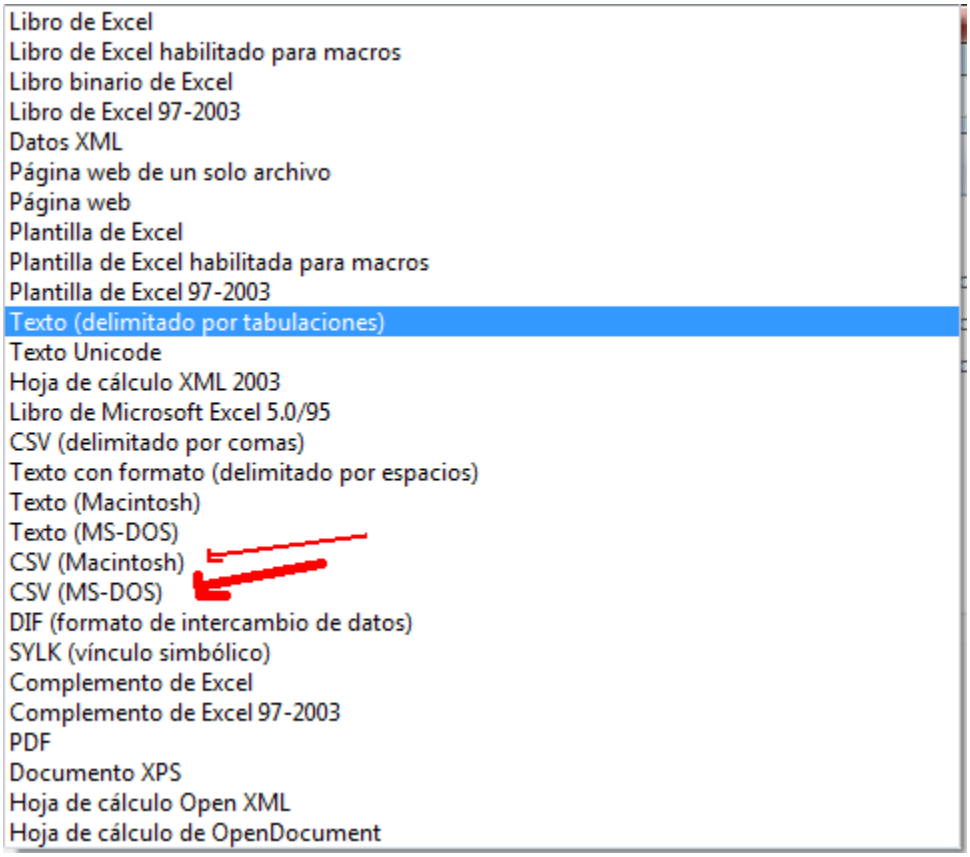
Finalmente aparece outra pantalla para que escollades algunhas cousas, a mais importante é a de delimitador de campos, posto que aí podedes escoller o símbolo que separa as columnas:



Escollede **;** que é o simbolo que usamos na nosa materia, ou calquera outro que vos pareza apropiado para o que esteades a facer.

Premedes en **aceptar**, e xa tedes un ficheiro csv gardado.

Gardar csv con Excel (Microsoft Office) Hai variedade de situacions segundo a versión que utilicedes, pero en xeral, se escolledes **Gardar como** tedes varias opcións de tipo de formato. As opcións para gardar con **coma (,)** decimal e separar por **;** serían **CSV - MS DOS**, ou incluso **csv MAC**


Libro de Excel
Libro de Excel habilitado para macros
Libro binario de Excel
Libro de Excel 97-2003
Datos XML
Página web de un solo archivo
Página web
Plantilla de Excel
Plantilla de Excel habilitada para macros
Plantilla de Excel 97-2003
Texto (delimitado por tabulaciones)
Texto Unicode
Hoja de cálculo XML 2003
Libro de Microsoft Excel 5.0/95
CSV (delimitado por comas)
Texto con formato (delimitado por espacios)
Texto (Macintosh)
Texto (MS-DOS)
CSV (Macintosh)
CSV (MS-DOS)
DIF (formato de intercambio de datos)
SYLK (vínculo simbólico)
Complemento de Excel
Complemento de Excel 97-2003
PDF
Documento XPS
Hoja de cálculo Open XML
Hoja de cálculo de OpenDocument

Cargar datos no R

O primeiro paso para traballar con R é introducirlle os datos, e o máis habitual será facer que os lea desde algún tipo de ficheiros.

R manexa comandos para ler ficheiros de datos de moi diferentes formatos e de diferente tipo de programas, pero o máis cómodo e o que garante mellores resultados e lelos desde **ficheiros de texto**, organizando as variables en forma de columnas, o que habitualmente se coñecen como **ficheiros csv**

Cargar datos desde un ficheiro csv

SEMPRE, SEMPRE, SEMPRE: Indicar AO COMEZAR cal é o CARTAFOL DE TRABALLO

Lembrade!

Antes de comezar a traballar, xusto **despois de abrir o RStudio** é conveniente indicarlle **cal é o cartafol de traballo**

Podedes ver unha breve explicación de como se fai:

configurar o cartafol.de.traballo

Cargar datos dun ficheiro csv: Cousas a ter en conta

- **Levan nomes de variable ou non?**: a primeira fila soe estar reservada para os nomes das variables (non sempre, posto que non é obrigatorio)
- **Símbolo que indica a separación das columnas**: O que adoito usar é o **punto e coma** (**;**). Outras posibilidades son unha **tabulación**, un **espazo en branco**, unha **coma**
- **Símbolo decimal**: a **coma** ou o **punto**

Comandos para ler un ficheiro csv: Cando o **ficheiro csv** verifica o seguinte:

- Ten nome de variables.
- Separador de columnas: (**;**)
- Decimal: (**,**)

```
#O comando máis directo é
read.csv2("nomedeficheiro")
```

```
#Por exemplo:
datos=read.csv2("csv/iris2.csv")

#le o ficheiro iris2.csv, e garda os seus valores nun data.frame chamado datos,
#co cal poderemos traballar a partir de agora
```

Para ficheiros csv ao estilo anglosaxón:

- Ten nome de variables na primeira fila
- Separador de columnas: (**,**)
- Decimal: (**.**)

```
read.csv("nomedeficheiro")
```

```
#Por exemplo:
datos=read.csv("csv/irisb.csv")

#le o ficheiro irisb.csv, e garda os seus valores nun data.frame chamado datos,
#co cal poderemos traballar a partir de agora
```

Para ficheiros de texto variados:

- **Non** ten nome de variables.
- Separador de columnas: (** **)** branco**
- Decimal: (**.**)

```
read.table("nomedeficheiro")
```

```
#Por exemplo:
datos=read.table("iris.csv")

#le o ficheiro iris.csv, e garda os seus valores nun data.frame chamado datos
```

Este último comando é xenérico, pódese modificar as carácterísticas do ficheiro lido para ampliar os tipos de ficheiros csv ou texto que se poden manexar (os demais tamén, claro). Por exemplo:

- **sep=**: indica o símbolo que separa as columnas
- **dec=**: indica o simbolo decimal
- **header=**: indica se ten títulos na 1º liña (**TRUE**) ou non (**FALSE**)

Unha opción interesante é indicarlle se algunha columna contén os nomes de cada observación:

- **row.names=1**: indica que cando lea o ficheiro de datos, considera que a primeira columna son os nomes asignados a cada observación.

```
#Por exemplo:
datos=read.table("iris2.csv",header=TRUE,sep=";",dec=",")

#le o ficheiro iris2.csv, pero tendo en conta que ten os nomes da variable na primeira columna,
#o simbolo decimal é a coma e o separador de columnas é o ;
```

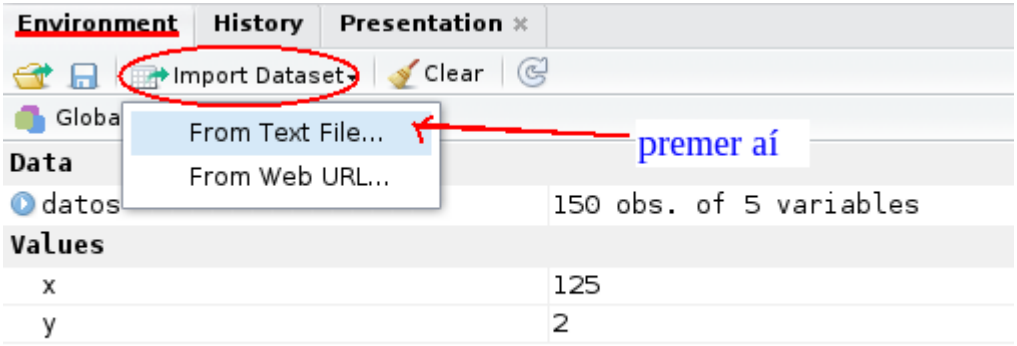
```
#Outro exemplo:
datos=read.table("irisb.csv",header=TRUE,sep=" ")

#le o ficheiro irisb.csv, pero tendo en conta que ten os nomes da variable na primeira columna,
#e o separador de columnas é o espazo en branco
```

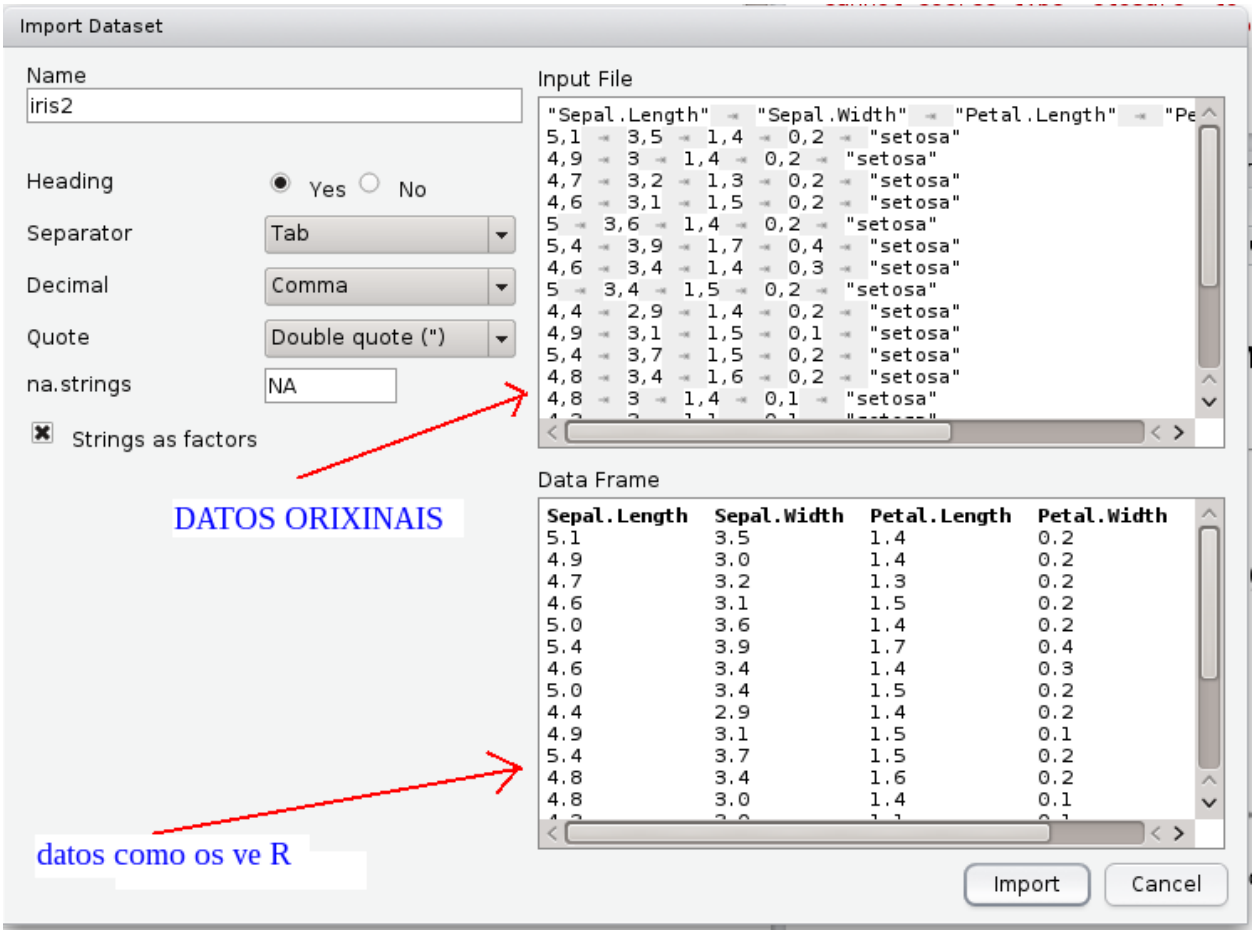
```
#e outro exemplo:
datos=read.csv2("iris3.csv",sep="\t")

#le o ficheiro irisb.csv, pero tendo en conta que ten os nomes da variable na primeira columna,
#simbolo decimal a coma e separador de columnas o tabulador
```

Cargar ficheiro csv usando RStudio En ***Environment***, pódese tamen cargar un ficheiro csv, sen necesidade de usar un comando:



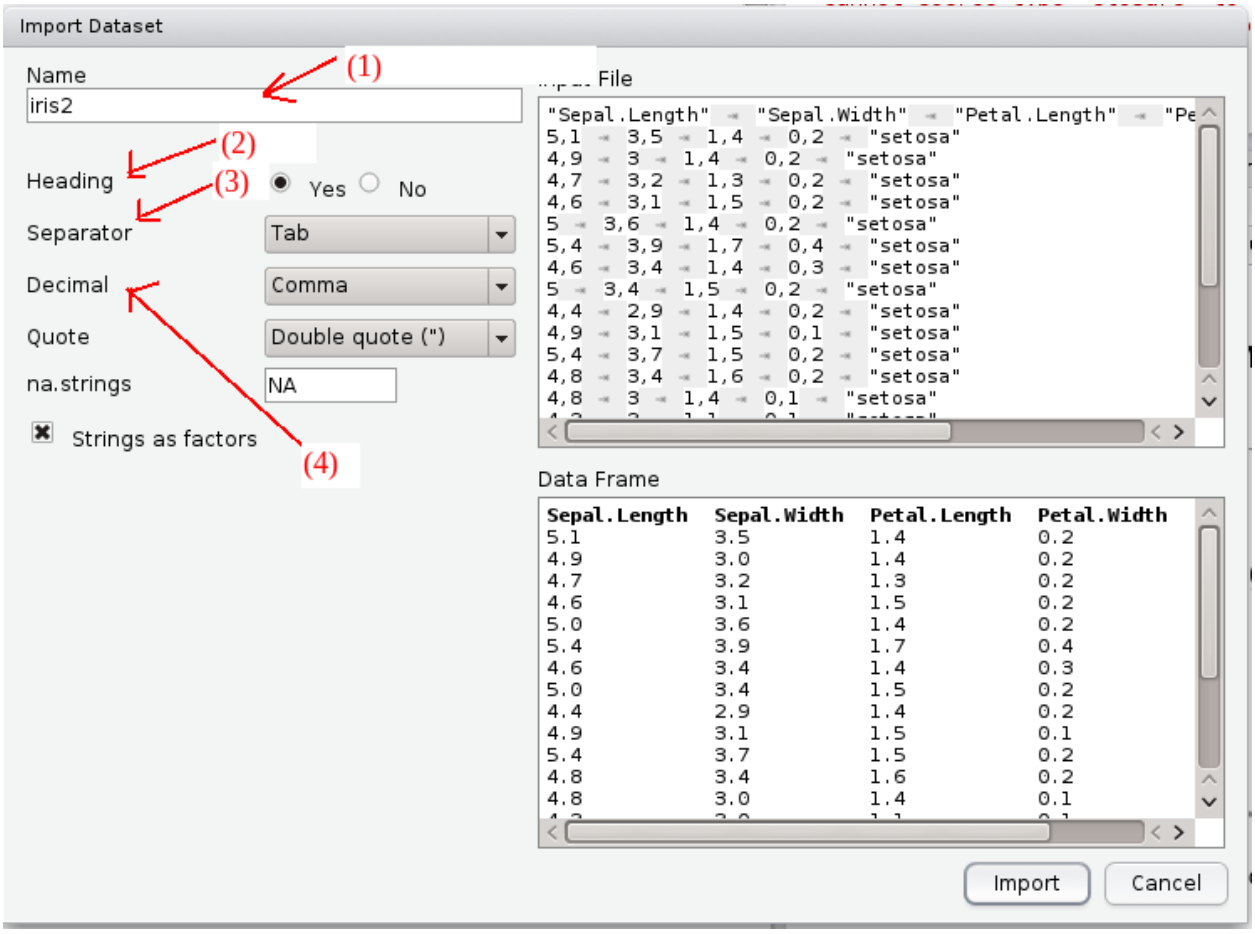
Seleccionase o ficheiro que se quere abrir e aparece unha interface:



Podese ver a forma do ficheiro de texto orixinal (neste caso as columnas estan separadas por tabuladores) e a interpretación que R fai das variables.

Esta información pode ser modificada escollendo as opcións do lado esquedo desa interface

- (1) Permite poñer **un nome ao obxecto** onde R gardará os datos (no exemplo con comandos chameino **datos**)
- (2) A primeira liña do ficheiro **ten nomes de variables?**
- (3) Simbolo que separa as **columnas**. Pêrmite espazo en branco, coma, punto e coma e tabulador
- (4) Simbolo **decimal**: coma ou punto



Abrir e cargar datos de libreoffice calc en R

O máis seguro é gardar desde Calc os datos en ficheiros de tipo **csv**, e abrilos desde R con ese formato, pero se queredes abrilos directamente desde o Calc existe un paquete que axudan a ler os seus ficheiros directamente:

readODS

Tamén podedes copialos e dicirlle a R que os lea desde o portapapeis:

```
#Ir á folla de cálculo, copiar os datos de interese e usar o seguinte código:

datos = read.table("clipboard")

#Funcionoume ben cando non había fórmulas, simplemente con valores.
```

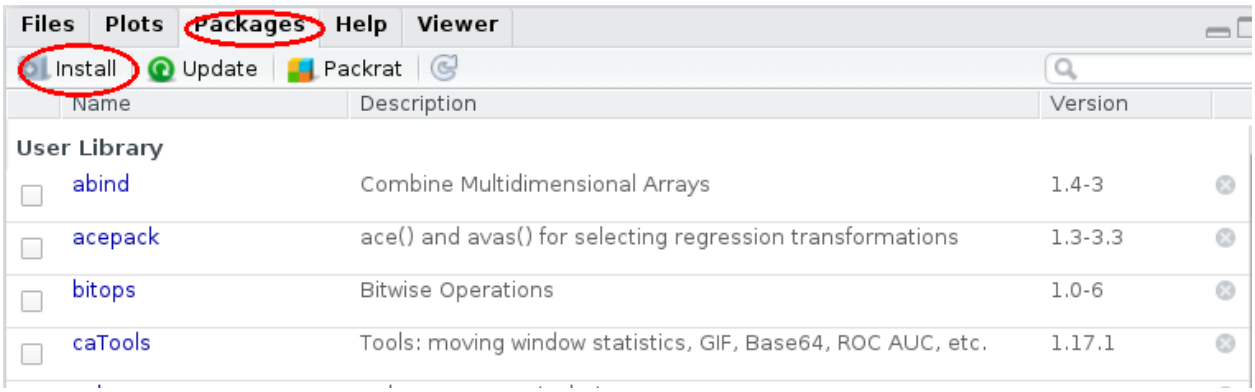
Como deben estar os datos Deben colocarse con unha columna para cada variable, os nomes das variables na primeira liña e os datos de cada unha na columna correspondente.

	A	B	C	D	E	F	G	H
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species			
2	5,1	3,5	1,4	0,2	setosa			
3	4,9	3	1,4	0,2	setosa			
4	4,7	3,2	1,3	0,2	setosa			
5	4,6	3,1	1,5	0,2	setosa			
6	5	3,6	1,4	0,2	setosa			
7	5,4	3,9	1,7	0,4	setosa			
8	4,6	3,4	1,4	0,3	setosa			
9	5	3,4	1,5	0,2	setosa			
10	4,4	2,9	1,4	0,2	setosa			
11	4,9	3,1	1,5	0,1	setosa			

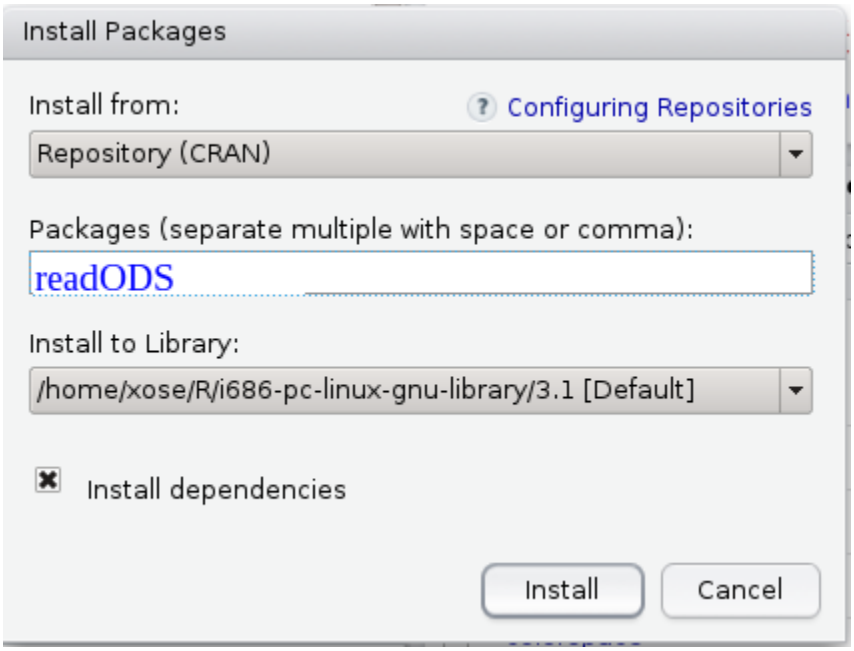
Usar o paquete readODS Este paquete non está no R básico, polo que é necesario instalalo desde os repositorios de R.

Usando RStudio é fácil facelo.

Instalar un paquete Faise desde o xestor de paquetes:



Premendo en **install** aparece o seguinte:



En **packages** debedes escribir o nome do paquete que vos interesa, e premer **install**

Se tedes este paquete instalado, é necesario cargalo na memoria para poder usalo:

```
library("readODS")
```

```
#A función que permite ler os datos é:
datos=read.ods("nome de ficheiro")

#en "nome de ficheiro" iría o nome que vos interese, por exemplo "iris.ods"

#datos é o nome do obxecto no que se gardarán os datos lidos da folla de cálculo
```

Abrir e cargar datos de excel en R

Igual ca con LibreOffice, tamén aquí é máis seguro gardar desde excel os datos en ficheiros de tipo **csv**, e abrilos desde R con ese formato

Se queredes abrilos directamente desde excel existen diferentes paquetes que axudan a R a ler eses ficheiros.

Podedes ver unha listaxe AQUÍ

Eu probei a facelo co paquete **readxl**.

Este paquete non está no R básico, polo que é necesario instalalo desde os repositorios de R.

Por suposto, os datos deben colocarse con unha columna para cada variable, os nomes das variables na primeira liña e os datos de cada unha na columna correspondente.

```
#Despois cargades o paquete na memoria:

library("readxl")
```

```
#A función que permite ler os datos é:
datos=read_excel("nome de ficheiro")

#en "nome de ficheiro" iría o nome que vos interese, por exemplo "iris.xslx"

#datos é o nome do obxecto no que se gardarán os datos lidos da folla de cálculo
```

Para excel tamén existe a opción de ler desde o portapapeis

Podedes copiar os datos e dicirlle a R que os lea desde o portapapeis:

```
#Ir á folla de cálculo, copiar os datos de interese e usar o seguinte código:

datos = read.table("clipboard")

#Funcionoume ben cando non había fórmulas, simplemente con valores.
```

Os data.frame

Cando se cargan datos en R, desde un ficheiro csv ou desde algún outro tipo de fontes, o habitual é gardalos automáticamente nun obxecto, que quedará almacenado na memoria do ordenador.

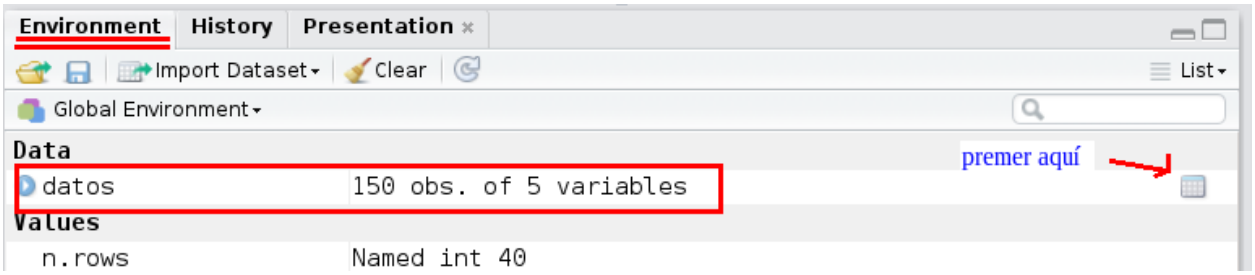
O obxecto que se crea é un **data.frame**, e pódese describir dicindo que equivale a unha colección de variables (1 por columna), que teñen todas o mesmo número de valores.

Algo equivalente á seguinte imaxe:

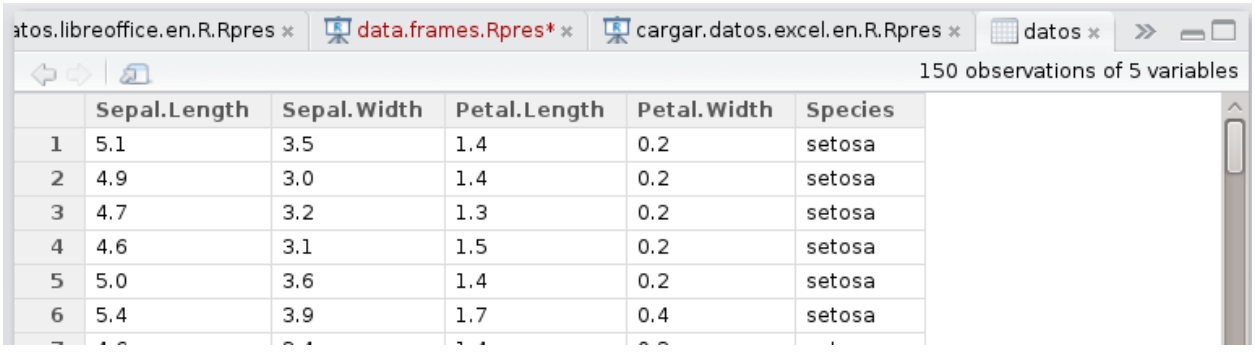
	A	B	C	D	E	F	G	H
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species			
2	5,1	3,5	1,4	0,2	setosa			
3	4,9	3	1,4	0,2	setosa			
4	4,7	3,2	1,3	0,2	setosa			
5	4,6	3,1	1,5	0,2	setosa			
6	5	3,6	1,4	0,2	setosa			
7	5,4	3,9	1,7	0,4	setosa			
8	4,6	3,4	1,4	0,3	setosa			
9	5	3,4	1,5	0,2	setosa			
10	4,4	2,9	1,4	0,2	setosa			
11	4,9	3,1	1,5	0,1	setosa			

```
#Por exemplo, lemos o seguinte ficheiro:
datos=read.csv2("csv/iris.csv")
```

Despois de executar o comando anterior temos en memoria un obxecto **datos** que ten estructura de data frame.



Como é un data frame? Podes velo no ***Environment***, premendo na cuadrícula á dereita do obxecto **datos**.



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Pódese interpretar como un "almacen de variables", cada unha co seu nome e ocupando unha columna.

Como se traballa cun data frame?

Pódense usar os seus datos sen necesidade de sacalos do data.frame, pero se tes un nivel moi elemental pode ser máis cómodo crear obxectos novos, cada un deles contendo unha variable.

O procedemento consiste en usar un comando que **cree un obxecto para cada variable**, e iso faise **asignando un nome ao contido de cada columna** do data frame.

```
#a primeira columna ("Sepal.Length")
longo.sepalo=datos[[1]]

#agora teño un obxecto novo, chamado longo.sepalo, que coincide coa primeira columna
#do data.frame, pero traballará independentemente del.

#a 2ª columna:
ancho.sepalo=datos[[2]]
```

OLLO! cos dous corchetes:

datos[[2]]: significa "*o que esta dentro da segunda columna*", que no caso deste exemplo **é un vector número**,

pero se só poñedes un corchete:

datos[2]: o que tedes é outro **data.frame**, pero só **con 1 columna** dentro do cal está o vector numérico *datos[[2]]*

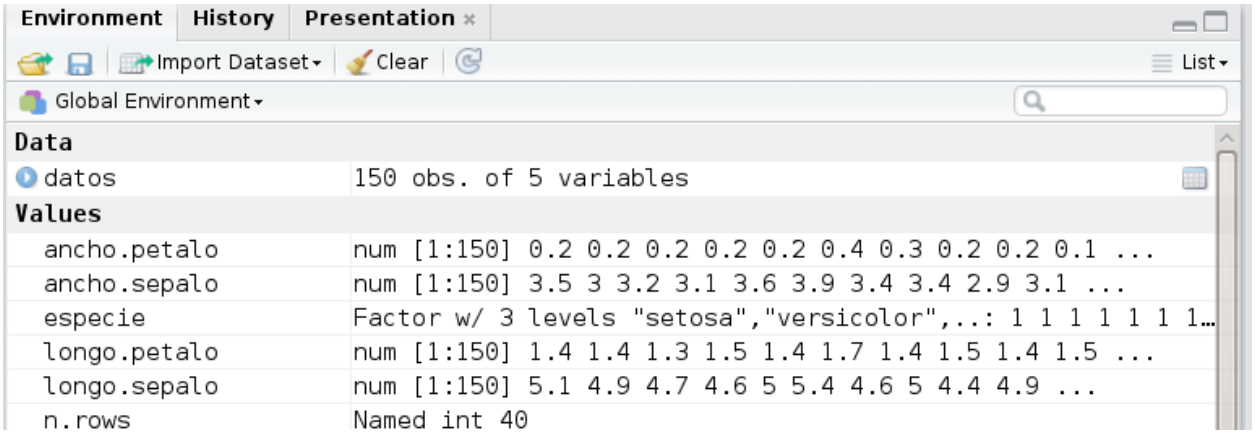
Por iso, para crear variables novas con estructura de vector hai que usar os dous corchetes.

Outra maneira de extraer as variables dun data.frame Pódense indicar tamén usando un **$ e o nome que corresponde a cada columna**:

```
#Por exemplo:
longo.petalo=datos$Petal.Length
ancho.petalo=datos$Petal.Width
especie=datos$Species

#Así temos tres novas variables coas que traballar
```

Podemos ver o que acabamos de crear mirando no **espazo de traballo** de RStudio:



Que hai en cada columna: vectores Son a **estructura de datos máis simple** de R e poden describirse como **unha colección ordeada de elementos do mesmo tipo**, e son a maneira habitual de manexar variables estatísticas en R, se ben poden estar como obxectos independentes de tipo vector ou incluidos en estructuras máis amplas.

Dous exemplos de vectores serían:

Un vector numérico sería c(10.4, 5.6, 3.1, 6.4, 21.7)

Un vector de caracteres c("leite","carne","arroz","froita")

A súa estructura é semellante: varios elementos do mesmo tipo en cada un deles, se ben son de tipo diferente en cada vector. Tamén e diferente o seu tamaño, 5 elementos o primeiro e 4 o segundo.

Outra característica común é como se accede aos elementos de cada vector:

```
vector1=c(10.4, 5.6, 3.1, 6.4, 21.7)
vector2=c("leite","carne","arroz","froita")

vector1[2]
```

```
## [1] 5.6
```

```
vector2[c(3,1)]
```

```
## [1] "arroz" "leite"
```

cada elemento ten un índice de referencia, habitualmente o seu número de orde, pero tamén se pode modificar para que sexa un nome específico do elemento.

Asignación: a función c(): concatenar

Nos comandos anteriores usouse a función **c()** para asociar os elementos dun vector co seu nome (*vector1=c(10.4, 5.6, 3.1, 6.4, 21.7)*), ou para indicar a un vector que queremos ver 2 (poderían ser máis) dos seus elementos (*vector2[c(3,1)]*).

Esta operación denomínase **asignar** e o comando *c()* é a maneira máis habitual de realizala.

Con todo, a utilidade específica da función c() é concatenar, ou sexa, **crear un novo obxecto colocando elementos ou obxectos un detras do outro**. Por exemplo

```
y <- c(vector1, 0, 2*vector1)
y
```

```
##  [1] 10.4  5.6  3.1  6.4 21.7  0.0 20.8 11.2  6.2 12.8 43.4
```

ANEXO

Gardar datos no R

Os comandos tipo *read.algo* para ler datos dun ficheiro *csv* teñen o seu equivalente para gardalos;

```
#O comando máis directo é
write.csv2(data.frame.con.datos,file="nomedeficheiro")
```

garda os datos do *data.frame.con.datos* nun ficheiro chamado "nomedeficheiro", de xeito que a primeira columna é o nome de cada fila (ou o nº de orde, se non teñen nome) e despois cada variable nunha columna diferente, colocando o seu **nome na primeira fila**, ademais usa **;** como separador de columnas e **,** como separador decimal.

```
#Por exemplo:
write.csv2(datos,file="iris2retocado.csv")
```

Unha opción que pode ser interesante é pedirlle que non garde os nomes das observacións (ou o seu nº de orde), para conseguilo úsase o parámetro ***row.names = FALSE***.

ollo!: para read ese parámetro igualábase a un número para indicar en que columna se incluen os nomes das observacións

Outros comandos *write*:

```
write.csv(data.frame.con.datos,file="nomedeficheiro")
```

Ten nome de variables na primeira fila; separador de columnas: (**,**); decimal: (**.**)

```
write.table(data.frame.con.datos,file="nomedeficheiro")
```

Comando xenérico. Pódese modificar usando parámetros, de xeito semellante ao equivalente *read.table()*

```
#Por exemplo:
write.table(datos,file="iris2.csv",header=TRUE,sep=";",dec=",",row.names=FALSE)

#le o ficheiro iris2.csv, pero tendo en conta que ten os nomes da variable na primeira columna,
#o simbolo decimal é a coma e o separador de columnas é o ;
#non incluira os nomes das observacións
```

Gardar en formato binario: ficheiros Rdata

Outra posibilidade para gardar data.frames de datos e outro tipo de obxectos é facelo no formato binario propio de R.

Deste xeito non se está limitado a obxectos con formato de data.frame, se non que se pode gardar calquera cousa que teñamos na memoria de R.

Este formato é o que R emprega para gardar o espazo de traballo (o *workspace*), que está formado por todos os obxectos que están na memoria de R. Tense así unha imaxe do traballo que se está a realizar nun momento dado.

En calquera momento, volvendo a cargar ese ficheiro teremos de novo o noso traballo no punto onde o deixáramos ao gardar.

```
#gardar o espazo de traballo enteiro nun ficheiro denominado ".Rdata"
save.image()

#en linux este ficheiro non se ve, por que comeza por punto
#(en OS X, de Apple, creo que tampouco)
```

Con este método, pero outro comando (*save*), pódense gardar tamén unha parte dos obxectos, sen ter que gardalos con todos. Necesítase simplemente indicar a lista de obxectos a gardar e un nome para o ficheiro *Rdata*:

```
save(datos,vector1,auxi,file="practica.Rdata")
```

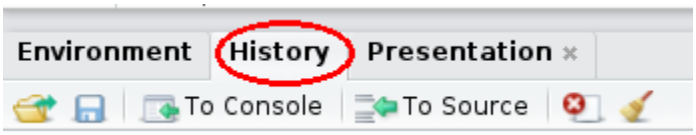
Para cargar en R un destes ficheiros, xa sexa un *workspace* completo, ou contendo só uns poucos obxectos, emprégase o comando *load()*, co nome do ficheiro que nos interesa:

```
load("practica.Rdata")
```

Gardar o historial de comandos

Outra cousa que se pode gardar e volver a cargar é o historial de comandos executados nunha sesión.

Podedes ver un exemplo de historial premendo na pestana *History*, no *Espazo de traballo* de RStudio:



Cando iniciamos R o programa sempre carga o ficheiro de historial da sesión anterior, para que poidamos volver a executar os seus comandos, pero existe a posibilidade de recuperar historiais de sesións diferentes, mediante comandos.

O comando que garda o historial da sesión actual é:

```
#gardao no ficheiro '.Rhistory'
savehistory()

#gardao no ficheiro 'sesion.Rhistory'
savehistory(file="sesion.Rhistory")
```

Para recuperalo faise:

```
#carga '.Rhistory'
loadhistory()

#carga o ficheiro 'sesion.Rhistory'
loadhistory(file="sesion.Rhistory")
```

Os ficheiros co historial son simples ficheiros de texto, en caso de que queiramos observar o que hai nun deles pódese abrir con calquera editor de texto simple.

ANEXO

Tipos de datos

- **numeros**

Son o tipo *numeric*, ainda que poden serpararse en *integer*, equivalentes aos números enteiros, e *double*, equivalentes aos números reais.

Para R a diferencia vai estar en como os almacena, xa que integer e double almacénanse de maneira diferente, se ben para traballar con eles a nivel elemental é máis práctico identificalos con números enteiros e números reais

```
x=c(1,2,2,1,3,7)
mode(x) #identifica o "modo" do obxecto,
```

```
## [1] "numeric"
```

```
typeof(x)
```

```
## [1] "double"
```

```
#podemos cambiarlle o tipo aos elementos de x, indicando que os queremos como enteiros,
#xa que como números son enteiros
x=as.integer(x)
typeof(x)
```

```
## [1] "integer"
```

```
mode(x)
```

```
## [1] "numeric"
```

- **atributos**

Para datos cualitativos R manexa o tipo *character*, que está referido ao dato en si, a maneira de agrupalos nun obxecto pode dar lugar a un vector de caracteres ou a un factor (ordenado ou non)

```
y=c("a","b")
mode(y)
```

```
## [1] "character"
```

```
typeof(y)
```

```
## [1] "character"
```

```
#O mesmo con:
y2=c("un","dous","tres")
mode(y2)
```

```
## [1] "character"
```

```
typeof(y2)
```

```
## [1] "character"
```

- **lóxicos**

Son o tipo *logical*. Os valores *TRUE*, *FALSE* (en maiuscula os dous), son valores lóxicos, e forman tamén un tipo de datos. Aparecen como resultado dunha condición, por exemplo:

```
6>4
```

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

```
3>6
```

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

Un uso frecuente é para extraer un subconxunto dun obxecto (un vector, un data.frame...) que verifique unha condición determinada.

```
v=c(3,2,2,1,4,5)
#condicion
v>=3#pódese ver o resultado, os TRUE son os elementos que cumpren a condición, os FALSE os que non
```

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

```
#Podemos introducir esa condición directamente no vector v
v[v>=3]
```

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

```
#ou podemos gardar o resultado da condición nun vector lóxico, e aplicalo despois ao vector v
condicion=v>=3
condicion
```

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

```
v[condicion]
```

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

```
mode(condicion)
```

```
## [1] "logical"
```

```
typeof(condicion)
```

```
## [1] "logical"
```

Tipos de datos: estructuras de datos

Os anteriores tipos referíanse aos elementos simples que van formar os diferentes obxectos. Pero estes elementos traballan habitualmente agrupados en obxectos, de diferentes tipos, dos cales os básicos son os seguintes:

vector

É unha colección de elementos que teñen o mesmo *modo*, ou sexa, todos números (*vector numérico*), caracteres (*vector de caracteres*), valores lóxicos (*vector lóxico*)...

Sería a estructura de datos máis simple.

```
v1=c(2,2,3,3,1,2,8)
str(v1) #vector numerico de 7 elementos indexados do 1 ao 7
```

```
##  num [1:7] 2 2 3 3 1 2 8
```

```
mode(v1) #como modo ten o modo dos elementos que o compoñen
```

```
## [1] "numeric"
```

```
typeof(v1)
```

```
## [1] "double"
```

```
length(v1)#longo do vector
```

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

```
#Se queremos referirnos a un elemento do vector, hai que indicar o número da súa posición
#entre corchetes:
v1[3]
```

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

Cando indicamos indices de posición dentro dun obxecto deben usarse corchetes *[]*, que en algúns casos deben ser corchetes dobres *[[]]*. Os parénteses quedan reservados para os valores das funcións:

```r
mean(v1)
```

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

Matrices e arrays

Un vector é un obxecto que ten unha dimensión, o seu tamaño, polo tanto sitúase aos seus elementos mediante un único índice

Se traballamos con dúas dimensións, nº de filas e nº de columnas, entón temos unha matriz

```r
m1=matrix(c(2,3,3,2,1,9),nrow = 2,ncol = 3)#crea unha matriz con 2 filas e 3 columnas

m1
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    1
## [2,]    3    2    9
```

```r
dim(m1) #ten 2 dimensións, o primeiro é o número de filas, o segundo o número de columnas
```

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

```r
#Para referirnos a un elemento debemos indicar a súa posición na fila (1º) e na columna (2º)
m1[1,3]
```

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

Se manexamos máis de 2 dimensións o tipo de estructura é o ***array***, que ven sendo unha matriz na cal as dimensións van máis ala do nº de filas e nº de columnas.

```r
a1=array(c(2,3,3,2,1,9,0,0,0,2,3,5,1,1,3,2), dim=c(2,4,2))
#Os valores identifícanse agora con 2 índices
dim(a1)
```

```
## [1] 2 4 2
```

```r
a1[2,3,1]
```

```
## [1] 9
```

factor

É a maneira máis habitual de traballar con variables cualitativas. Diferénciase dos vectores de caracteres no xeito de almacenarse, xa que nos vectores de caracteres almacénase todos os valores cualitativos, repetindoos, mentres que o factor almacénaos unha vez, como lista de nomes, gardando despois o número que referencia a cada un deles.

```r
v2=c("si","non","depende","depende","si","si","non")
v2 #vector de caracteres
```

```
## [1] "si"      "non"     "depende" "depende" "si"      "si"      "non"
```

```r
as.numeric(v2) #non deixa transformar os caracteres en números
```

```
## Warning: NAs introducidos por coerción
```

```
## [1] NA NA NA NA NA NA NA
```

```r
v2=factor(v2)
v2
```

```
## [1] si      non     depende depende si      si      non
## Levels: depende non si
```

```r
as.numeric(v2) #indica o numero que almacenou o factor
```

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

```r
levels(v2) #os niveis do factor
```

```
## [1] "depende" "non"     "si"
```

data.frame

Pode describirse como unha colección de vectores do mesmo tamaño, pero que poden ter diferentes tipos de datos

```r
v1=c(2,2,3,3,1,2,8)
v2=c("si","non","depende","depende","si","si","non")
datos=data.frame(v1,v2)#a función data.frame() creou un data.frame que ten
#dúas columnas, de nome v1 e v2, a primeira delas un vector numérico
#e a segunda un factor
#(data.frame() transformou o vector de caracteres en factor)
str(datos)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ v1: num  2 2 3 3 1 2 8
##  $ v2: Factor w/ 3 levels "depende","non",..: 3 2 1 1 3 3 2
```

```r
names(datos)
```

```
## [1] "v1" "v2"
```

```
datos
```

```
##   v1      v2
## 1  2      si
## 2  2     non
## 3  3 depende
## 4  3 depende
## 5  1      si
## 6  2      si
## 7  8     non
```

Un data.frame coincide coas matrices en que ten 2 dimensións: nº de fila e nº de columna.

A diferencia está en que as columnas poden estar formadas por datos de tipos diferentes, polo que se poden interpretar como unha estructura para conter en cada columna os datos dunha variable.

Para os elementos do data.frame pódense usar 1 corchete *[]* ou dous *[[]]*, pero producen diferentes cousas:

```
#Extraer elementos do data.frame
datos[4,1] #na 4ª fila e na 1ª columna
```

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

```
datos[1:3,2] #as 3 primeiras filas e a 2ª columna
```

```
## [1] si      non     depende
## Levels: depende non si
```

```
datos[1,] #a primeira fila
```

```
##   v1 v2
## 1  2 si
```

O dobre corchete aparece cando nos interesa variables enteiras, ou sexa, columnas. Se usamos corchetes simples o que se extrae ten estructura de data.frame, mentres que se usamos corchetes dobres temos o que "esta dentro" do data.frame:

```
datos[1]
```

```
##   v1
## 1  2
## 2  2
## 3  3
## 4  3
## 5  1
## 6  2
## 7  8
```

```
str(datos[1])  #é un data.frame que só conten a primeiro columna de datos
```

```
## 'data.frame':    7 obs. of  1 variable:
##  $ v1: num  2 2 3 3 1 2 8
```

```
datos[[1]]
```

```
## [1] 2 2 3 3 1 2 8
```

```
str(datos[[1]]) #é un vector que está dentro da primeira columna do data.frame datos
```

```
##  num [1:7] 2 2 3 3 1 2 8
```

listas

Un objeto de tipo **list** é unha colección de obxectos que poden ser de diferentes tipos.

Pode pensarse nel como un vector no que cada elemento pode ser un obxecto, en lugar dun elemento individual.

```
elemento="ola"
vector=c(4,3,3,2)
matriz=matrix(c(4,3,3,2),nrow = 2,ncol = 2)
factor1=factor(c("ola","abur","ciao"))
lista=list(elemento,vector,matriz,factor1)
lista
```

```
## [[1]]
## [1] "ola"
##
## [[2]]
## [1] 4 3 3 2
##
## [[3]]
##      [,1] [,2]
## [1,]    4    3
## [2,]    3    2
##
## [[4]]
## [1] ola  abur ciao
## Levels: abur ciao ola
```

Nas listas tamén se poden usar corchete ou dobre corchete.

Con **corchete extraemos unha lista máis pequena**, que conten os elementos solicitados da lista

```
lista[1]
```

```
## [[1]]
## [1] "ola"
```

```
lista[1:2]
```

```
## [[1]]
## [1] "ola"
##
## [[2]]
## [1] 4 3 3 2
```

```
lista[c(1,3)]
```

```
## [[1]]
## [1] "ola"
##
## [[2]]
##      [,1] [,2]
## [1,]    4    3
## [2,]    3    2
```

Con dobre corchete extraemos o que hai dentro dun elemento da lista, que pode ser dun tipo diferente a *list*, mentres que con 1 corchete o extraido seguia senda *list*:

```
lista[[3]]
```

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

```
str(lista[[3]])
```

```
##  num [1:2, 1:2] 4 3 3 2
```

```
str(lista[3])
```

```
## List of 1
##  $ : num [1:2, 1:2] 4 3 3 2
```

Os elementos dunha lista tamén se poden extraer por nome, cando o teñen asignado, algo que tamén ocorre nos data.frames:

```
elemento="ola"
vector=c(4,3,3,2)
matriz=matrix(c(4,3,3,2),nrow = 2,ncol = 2)
factor1=factor(c("ola","abur","ciao"))
lista=list(primeiro=elemento,segundo=vector,terceiro=matriz,cuarto=factor1)
lista
```

```
## $primeiro
## [1] "ola"
##
## $segundo
## [1] 4 3 3 2
##
## $terceiro
##      [,1] [,2]
## [1,]    4    3
## [2,]    3    2
##
## $cuarto
## [1] ola  abur ciao
## Levels: abur ciao ola
```

```
lista$terceiro
```

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

As listas poden dar lugar a estructuras moi complexas, xa que se poden incluir listas como elementos de outras listas.

```
elemento="ola"
vector=c(4,3,3,2)
matriz=matrix(c(4,3,3,2),nrow = 2,ncol = 2)
factor1=factor(c("ola","abur","ciao"))
lista0=list(un="ciao",dous=c(25,64))
lista=list(primeiro=elemento,segundo=vector,terceiro=lista0)
lista
```

```
## $primeiro
## [1] "ola"
##
## $segundo
## [1] 4 3 3 2
##
## $terceiro
## $terceiro$un
## [1] "ciao"
##
## $terceiro$dous
## [1] 25 64
```

```
lista$terceiro
```

```
## $un
## [1] "ciao"
##
## $dous
## [1] 25 64
```

ANEXO: vectores, como se manexan?

Vectores e asignacións

Estes apuntamentos están adaptados do manual de introdución ao R: https://cran.r-project.org/doc/manuals/r-release/R-intro.html

Para o traballo con R hai que manexar diferentes tipos de estructuras, xa sexa de datos, de resultados ou cousas máis sofisticadas. Xenericamente R refírese a todas elas como **obxectos**.

Por exemplo, se temos que calcular **unha regresión** con dúas variables poderíamos manexar **dous obxectos** diferentes, un deles cunha **variable X** e outro cunha **variable Y2**, e despois de realizar os cálculos podemos ter **outro obxecto diferente** no que R nos gardou diferentes **resultados** e valores relacionados coa regresión que acabamos de realizar.

Na regresión tamén poderíamos usar un único obxecto que conteña a X e Y2

Con respecto aos obxectos que son **estructuras de datos**, **a máis simple é o *vector***, que ven sendo **unha colección ordeada de elementos do mesmo tipo**.

> Un vector numérico sería (10.4, 5.6, 3.1, 6.4, 21.7)
>
> Un vector de caracteres ("leite","carne","arroz","froita")
>
> ou incluso un vector formado por valores lóxicos (TRUE,TRUE,FALSE,FALSE,TRUE)

As tres estructuras son semellantes, pero cambia o tipo de datos que conteñen, o que vai producir diferencias no comportamento dos comandos que lle apliquemos.

Como se crea un vector en R?

En R **calquera obxecto existe cando lle asignamos un nome** e **queda gardado na memoria**. Por esta razón, se queremos crear un vector temos que coller unha colección de elementos e asignarlle un nome que será a maneira na que R o vai identificar.

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

Nese comando usamos a función **c()** para agrupar os valores entre paréntese e crear un obxecto con eles, os símbolos **<-** (*Os signos* < e **-**) son o operador de asignamento, e neste caso **asignan** os valores entre paréntese a un obxecto con nome ***x*** (*Ollo! x minúscula, se puxésemos X sería un nome de obxecto diferente, posto que R considera signos diferentes as maiúsculas e as minúsculas*). Con isto temos creado un vector de nome x, que contén os valores 10.4, 5.6, 3.1, 6.4 e 21.7.

Para crear un vector de ***strings*** (anacos de texto) o procedemento sería semellante, lembrando unicamente que **os textos deben ir entre comiñas**, xa sexan simples ou dobres, se non **R** considéraos obxectos e non textos.

Con respecto ao operador de asignamento comentar que na maioría dos contextos pode ser sustituido polo simbolo =, que será o que eu use con preferencia, ainda que é frecuente atopar recomendacións en sentido contrario.

A función c(): concatenar

A utilidade específica da función c() é concatenar, ou sexa, **crear un novo obxecto colocando elementos ou obxectos un detras do outro**. Por exemplo

```
y <- c(x, 0, x)
```

crea un novo obxecto y, que será un vector con 11 elementos, os cinco primeiros iguais aos de x, seguido por un cero e os cinco finais volven ser os de x.

```
y
```

```
##  [1] 10.4  5.6  3.1  6.4 21.7  0.0 10.4  5.6  3.1  6.4 21.7
```

Aritmética de vectores

Cos vectores vanse poder realizar operacións aritméticas, de xeito semellante a como se farían con números, se ben aparecera algunha peculiaridade derivada do feito de que os vectores teñen tamaño (longo: **length**) que poden ser diferentes nos vectores que manexamos.

Unha operación escríbese así:

```
#Se temos dous vectores x1 e x2
    x1=c(3,2,4,4,3)
    x2=c(5,0,-3,1,9)

#A súa suma será:
    x1+x2 #sumo o vector x consigo mesmo
```

```
## [1]  8  2  1  5 12
```

estou indicando con esa expresión que cada valor de x1 se sume co valor de x2 que opupe a mesma posición:

x1+x2=c(3+5,2+0,4-3,4+1,3+9)

Isto cadra ben por que o longo dos vectores é o mesmo. **Que pasaría cando os vectores teñen tamaños diferentes?**

Nese claso "recíclanse" os valores, i.e., cando se acaban os valores do vector máis pequeno volven a comezar desde o inicio. Por exemplo

```
#un vector máis pequeno:
    x3=c(3,0,4)
    x1+x3 #da o resultado c(3+3,2+0,4+4,4+3,3+0), recicla os valores de x3
```

```
## Warning in x1 + x3: longitud de objeto mayor no es múltiplo de la longitud
## de uno menor
```

```
## [1] 6 2 8 7 3
```

```
#cantas veces sexa necesario ata que completa os valores de x1
```

Hai que ter coidado, posto que esta peculiaridade pode producir resultados non esperados, se non somos conscientes da diferencia dos tamaños dos vectores.

Nas operacions con vectores pódense manexar constantes sen ningún problema, posto que R vai tratalos como vectores de longo 1. Por exemplo:

```
v <- 2*x + y + 1
```

```
## Warning in 2 * x + y: longitud de objeto mayor no es múltiplo de la
## longitud de uno menor
```

```
v
```

```
##  [1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
```

constrúe un vector de longo 11 engadindo, elemento a elemento, o vector y (o máis grande) unha vez, 2*x repíteo 2.2 veces e o 1 repetirao 11 veces, unha por cada elemento do vector y, que é o de maior tamaño na expresión.

Operacións aritméticas e funcións

As máis habituais: +, -, *, / e ^ para as potencias

Tamén están dispoñibles funcións como: **log()** (logaritmo neperiano), **exp()** (exponencial), **sin()** (seno), **cos()** (coseno), **tan()** (tanxente), **sqrt()** (raíz cadrada)...

Todas estas funcións teñen en común que se aplican habitualmente a un único número máis que a un vector:

```
exp(3)
```

```
## [1] 20.08554
```

```
sqrt(16)
```

```
## [1] 4
```

Que pasa se o aplicamos a un vector? Simplemente que obtemos outro vector, do mesmo tamaño pero cos resultados producidos pola función:

```
exp(x2)
```

```
## [1] 1.484132e+02 1.000000e+00 4.978707e-02 2.718282e+00 8.103084e+03
```

```
log(x1)
```

```
## [1] 1.0986123 0.6931472 1.3862944 1.3862944 1.0986123
```

```
sqrt(x3)
```

```
## [1] 1.732051 0.000000 2.000000
```

Outras funcións, teñen sentido aplicadas a un vector de máis de 1 elemento. Por exemplo **max()** (máximo), **min()** (mínimo), **length()** (nº de elementos do vector), **sum()** (suma dos elementos do vector), **prod()** producto dos elementos do vector.

Nesta categoría xa comezan a aparecer funcións estatísticas: **mean()** (media aritmética), **median()** (mediana), **var()** (varianza mostral), **sd()** (desviación típica)

```
mean(x1)
```

```
## [1] 3.2
```

```
sd(x2)
```

```
## [1] 4.669047
```

```
var(x3)
```

```
## [1] 4.333333
```

Unha función que necesita dous parámetros é **cov()** (covarianza) ou **cor()** (coeficiente de correlación)

```
cov(x1,x2)
```

```
## [1] -1.1
```

```
cor(x1,x2)
```

```
## [1] -0.2815888
```

Con todo, **para a maioría das funcións de R é habitual manexar máis de dous parámetros**, incluso de características diferentes. Un exemplo sería **lm()**, a función para unha regresión linear.

Segundo a **axuda de R** estes serían os parámetros desa función:

> lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

De calquera xeito, non adoita ser necesario empregalos todos. R proporciona valores por defecto para a maioría dos parámetros, de maneira que abonda con proporcionarlle aqueles que nos resulten útiles, ou que sexan obrigatorios:

```
lm(x2~x1) #regresión de x2 en función de x1
```

```
##
## Call:
## lm(formula = x2 ~ x1)
##
## Coefficients:
## (Intercept)           x1
##       7.429       -1.571
```

Todas estas funcións poden combinarse para realizar novas operacións:

```
sum(x1)/length(x1) #suma dos valores de x1 entre o seu numero de datos, i.e. a media aritmética
```

```
## [1] 3.2
```

A formula da varianza mostral podería escribirse:

```
sum((x-mean(x))^2)/(length(x)-1)
```

```
## [1] 53.853
```

Matrices e arrays

Os vectores son o tipo de obxecto máis importante en R, e poden describirse como **unha colección de elementos aos que se identifica con un índice**.

Partindo desta denominación é moi fácil pensar nunha **matriz**:

unha colección de elementos aos que se identifica con dous índices

ou nun **array**:

unha colección de elementos aos que se identifica con tres ou máis índices

Así un elemento dunha matriz sería:

```
  mx[2,3] #elemento da 2ª fila e 3ª columna da matriz mx
```

e para un array

```
    ax[5,1,1] elemento do array ax que ocupa a posición 5ª, 1ª, 1ª
```

Como se construe unha matriz? As matrices son un obxecto equivalente a un vector (**unha coleccion de elementos do mesmo tipo**) pero **con dúas dimensións**. Pódense construir coa función **matrix()**:

```
    x5=c(3,1,1,2,3,4,1,9,3,0,2,1) #definimos un vector para axudarnos no exemplo
    mx=matrix(x5,nrow = 4, ncol = 3)
    mx
```

```
##      [,1] [,2] [,3]
## [1,]    3    3    3
## [2,]    1    4    0
## [3,]    1    1    2
## [4,]    2    9    1
```

```
    dim(mx) #a matriz mx ten dúas dimensións, a primeira corresponde ás 4 filas e
```

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

```
#a segunda ás 3 columnas
```

Exr. 1.- Construide 3 vectores chamados u1, u2, u3. O primeiro cos valores 3, 2, 1, 1, 4, 5, 2, 5; o segundo cos cadrados do primeiro, e o terceiro cos logaritmos da suma dos dous primeiros

a) calculade o máximo e o mínimo de cada un deses vectores

b) calculade a media aritmética de cada un deses vectores, e despois o máximo desas tres medias

c) Calculade a suma da diferencia en valor absoluto (función abs()) dos valores de u1 menos a súa media aritmética.

2.- A función rnorm(n, mean = 0, sd = 1) produce un vector de n valores aleatorios procedentes dunha distribución normal de media mean (0 por defecto) e desviación típica sd (1 por defecto), podendo sustituir mean e sd polos valores que a nós nos interese, ou ignoralos se queremos simular unha Normal(0,1).

A función runif(n, min = 0, max = 1) fai o mesmo para unha distribución uniforme U(min,max) Sabendo iso, construide:

- un vector X que proceda dunha U(-1,1),
- un vector e que proceda dunha N(0,1), e
- un vector Y que sexa igual a 2*X+e
- Calculade a correlación entre X e Y
- Facede a regresión de Y en función de X

ANEXO: vectores, como se indexan?

Vectores de índices: escoller e modificar subconxuntos dun conxunto de datos

Igual ca en matemáticas, cada elemento de un vector ten un índice que o identifica, habitualmente numérico, se ben, tamén poden ser de outro tipo.

Tendo iso en conta podemos identificar calquera elemento ou subconxunto de elementos indicando o seu índice, que no caso de R exprésase co nome do vector e o indice ou índices entre corchetes:

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)#para operar con el

x[1] #o primeiro elemento
```

```
## [1] 10.4
```

```
x[3:5] #os elementos 3, 4 e 5
```

```
## [1]  3.1  6.4 21.7
```

```
x[c(2,4)] #os elementos 2º e 4º
```

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

Para describir as posicións dos elementos que queremos do vector podemos usar calquera das formas de describir sucesións numericas, e tamén condicións lóxicas, sempre e cando vaian entre corchetes

```
x1=c(3,2,4,4,3)
x2=c(5,0,-3,1,9)

x3=c(3,0,4,6,1)

x3[x3>2] #Os elementos de x3 maiores ca 2
```

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

```
x3[x1>2] #Os elementos de x3 que ocupan a mesma posición ca os elementos de x1 maiores ca 2
```

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

```
#aqueles elementos de x3 que coinciden na mesma posicion ca
#os elementos de x1 maiores ca os correspondentes de x2
x3[x1>x2]
```

```
## [1] 0 4 6
```

Se colocamos un signo negativo ao índice indicamos que queremos todos os elementos do vector excepto o marcado no índice:

```
y=c(10.4, 5.6, 3.1, 6.4, 21.7,0,10.4, 5.6, 3.1, 6.4, 21.7)

x[-3] #todos os elementos de x menos o terceiro
```

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

```
y[-(2:4)] #todos os elementos de y excepto o 2º, 3º e 4º
```

```
## [1] 10.4 21.7  0.0 10.4  5.6  3.1  6.4 21.7
```

De cara a crear vectores novos tamén se poden repetir os índices, e no novo vector apareceran repetidos os valores

```
y4=y[c(1,2,1,2,3,4,3,4)]
y4
```

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

Usar strings como indices

É posible colocar textos como indices dun vector, usando a función **names()**, deste xeito estaríamos asignando un nome a cada elemento do vector, en lugar de indicar unha posición.

Por exemplo:

```
fruit <- c(5, 10, 1, 20) #crea un vector con catro valores
names(fruit) <- c("orange", "banana", "apple", "peach") #asigna un nome a cada valor
lunch <- fruit[c("apple","orange")] #crea un novo vector só con dous elementos do vector fruit:
lunch
```

```
##  apple orange
##      1      5
```

ANEXO

Factores

Un **factor** en R é o tipo de dato habitual para as **variables categóricas**.

Unha variable categórica tamén se pode gardar nun vector de caracteres, pero o factor ten a vantaxe de aforrar memoria de ordenador, posto que só garda unha copia dos nomes dos niveis da variable, asignándolle un número para almacenalo e xestionalo, mentres que os vectores de caracteres almacenan os nomes dos niveis para cada un dos elementos do vector.

Un exemplo sería o seguinte:

```
exemplo=c("alta","alta","alta", "baixa","media","media")
exemplo
```

```
## [1] "alta"  "alta"  "alta"  "baixa" "media" "media"
```

```
#Temos en exemplo un vector de caracteres. Transfórmase en factor, mediante a seguinte función:
exemplo=factor(exemplo)
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: alta baixa media
```

pódese ver como no factor aparecen os seus niveis, ademais da lista de valores, mentres que no vector de caracteres só aparece a lista de valores.

A diferencia máis importante está na almacenaxe xa que o vector de caracteres garda os caracteres mentres o factor garda unha copia dos niveis e a lista cos numeros que representan a cada un deles.

```
#pódese ver a almacenaxe numérica pedindo a R que represente o factor como números: as.numeric()

as.numeric(exemplo)
```

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

Cando se representa graficamente un factor, ou cando se fai unha táboa dos seus niveis a orde coa que se representan é a alfabética.

```
table(exemplo)
```

```
## exemplo
##  alta baixa media
##     3     1     2
```

Con todo, ao crealo pódese **especificar unha orde diferente** usando o parámetro *levels*

```
exemplo=c("alta","alta","alta", "baixa","media","media")

#Temos en exemplo un vector de caracteres. Transfórmase en factor, mediante a seguinte función:
exemplo=factor(exemplo,levels=c("baixa","media","alta"))
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: baixa media alta
```

```
#xa se pode ver como cambiou a orde na que aparecen os niveis.

#Agora se facemos a táboa terase:
table(exemplo)
```

```
## exemplo
## baixa media  alta
##     1     2     3
```

ou *incluir niveis que non aparecen nos datos**

```
exemplo=c("alta","alta","alta", "baixa","media","media")

#Temos en exemplo un vector de caracteres. Transfórmase en factor, mediante a seguinte función:
exemplo=factor(exemplo,levels=c("baixa","media","depende","alta"))

#xa se pode ver como cambiou a orde na que aparecen os niveis.

#Agora se facemos a táboa terase:
table(exemplo)
```

```
## exemplo
##   baixa   media depende    alta
##       1       2       0       3
```

Existe tamén a función ***levels()***, que permite manipular os niveis dun factor xa creado, pero hai que ter coidado, posto que o que fai é cambiar as etiquetas que corresponden a cada un dos números almacenados no factor:

```
exemplo=c("alta","alta","alta", "baixa","media","media")

exemplo=factor(exemplo,levels=c("baixa","media","alta"))
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: baixa media alta
```

```
levels(exemplo)=c("baixa","media","depende","alta")
exemplo
```

```
## [1] depende depende depende baixa   media   media
## Levels: baixa media depende alta
```

```
#sustituiu a etiqueta do 3º valor existente, que agora é "depende"",
#pasando "alta" a ser a etiqueta do 4º valor, do que non hai ningún caso.
table(exemplo)
```

```
## exemplo
##   baixa   media depende    alta
##       1       2       3       0
```

Por iso, para incluir ou quitar niveis é máis práctico facelo recreando o factor:

```
exemplo=factor(c("alta","alta","alta", "baixa","media","media"))
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: alta baixa media
```

```
exemplo=factor(exemplo,levels=c("baixa","media","depende","alta","moi alta"))
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: baixa media depende alta moi alta
```

```
exemplo=factor(exemplo,levels=c("baixa","media","depende","alta"))
exemplo
```

```
## [1] alta  alta  alta  baixa media media
## Levels: baixa media depende alta
```

```
exemplo=factor(exemplo,levels=c("media","depende","alta"))
exemplo
```

```
## [1] alta  alta  alta  <NA>  media media
## Levels: media depende alta
```

```
#Coidado se eliminades un nivel que ainda exista no factor, por que
#obteredes un NA (elemento valeiro) na súa posición
exemplo=exemplo[-4]
exemplo
```

```
## [1] alta  alta  alta  media media
## Levels: media depende alta
```

ANEXO

Os data.frame

Cando se cargan datos en R, desde un ficheiro csv ou desde algún outro tipo de fontes, o habitual é gardalos automáticamente nun obxecto, que quedará almacenado na memoria do ordenador.

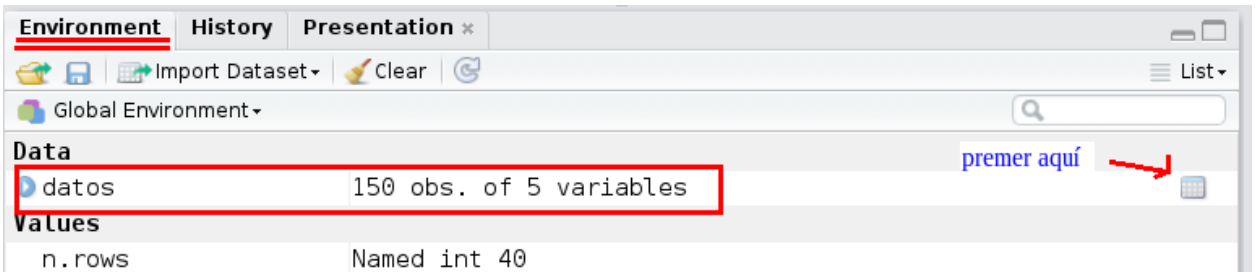
O obxecto que se crea é un **data.frame**, e pódese describir dicindo que equivale a unha colección de variables (1 por columna), que teñen todas o mesmo número de valores.

Algo equivalente á seguinte imaxe:

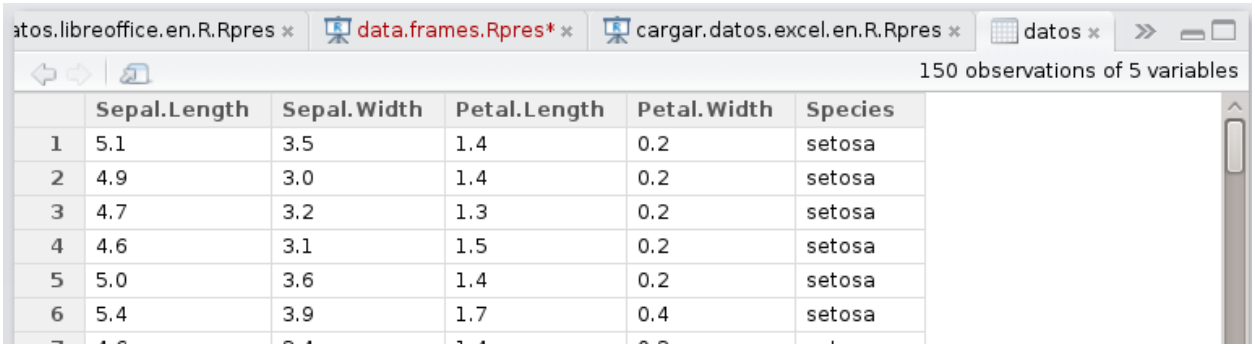
	A	B	C	D	E	F	G	H
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species			
2	5,1	3,5	1,4	0,2	setosa			
3	4,9	3	1,4	0,2	setosa			
4	4,7	3,2	1,3	0,2	setosa			
5	4,6	3,1	1,5	0,2	setosa			
6	5	3,6	1,4	0,2	setosa			
7	5,4	3,9	1,7	0,4	setosa			
8	4,6	3,4	1,4	0,3	setosa			
9	5	3,4	1,5	0,2	setosa			
10	4,4	2,9	1,4	0,2	setosa			
11	4,9	3,1	1,5	0,1	setosa			

```
#Por exemplo, lemos o seguinte ficheiro:
datos=read.csv2("csv/iris.csv")
```

Despois de executar o comando anterior temos en memoria un obxecto **datos** que ten estructura de data frame.



Como é un data frame? Podes velo no ***Environment***, premendo na cuadrícula á dereita do obxecto **datos**.



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Pódese interpretar como un "almacen de variables", cada unha co seu nome e ocupando unha columna.

Como se traballa cun data frame?

Pódense usar os seus datos sen necesidade de sacalos do data.frame, pero se tes un nivel moi elemental pode ser máis cómodo crear obxectos novos, cada un deles contendo unha variable.

O procedemento consiste en usar un comando que **cree un obxecto para cada variable**, e iso faise **asignando un nome ao contido de cada columna** do data frame.

```
#a primeira columna ("Sepal.Length")
longo.sepalo=datos[[1]]

#agora teño un obxecto novo, chamado longo.sepalo, que coincide coa primeira columna do data.frame, pero

#a 2ª columna:
ancho.sepalo=datos[[2]]
```

OLLO! cos dous corchetes:

datos[[2]]: significa "*o que esta dentro da segunda columna*", que no caso deste exemplo **é un vector númerico**,

pero se só poñedes un corchete:

datos[2]: o que tedes é outro **data.frame**, pero só **con 1 columna** dentro do cal está o vector numérico *datos[[2]]*

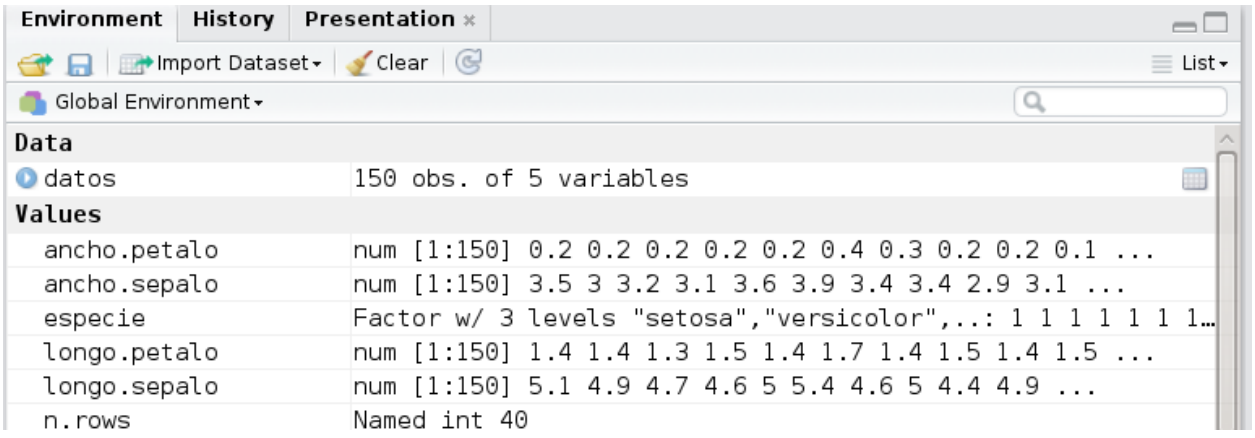
Por iso, para crear variables novas con estructura de vector hai que usar os dous corchetes.

Outra maneira de extraer as variables dun data.frame Pódense indicar tamén usando un **$ e o nome que corresponde a cada columna**:

```
#Por exemplo:
longo.petalo=datos$Petal.Length
ancho.petalo=datos$Petal.Width
especie=datos$Species

#Así temos tres novas variables coas que traballar
```

Podemos ver o que acabamos de crear mirando no **espazo de traballo** de RStudio:



Manexo de data.frames: quitar columnas

- Quitarlle unha columna

```
#simplemente indicando o número de columna, pero con signo menos
datos2=datos[-5]
```

- Quitarlle varias columnas

```
#simplemente indicando os números de columna, pero con signo menos
#e usando a función c() para asignalos
datos3=datos[-c(1,2,5)]
```

3 Manexo de data.frames: quitar filas

Un data.frame pode describirse como unha matriz, indicando o nº de fila e o número de columna para ver un dos seus elementos

datos[2,4] é a 2ª fila da 4ª columna, pero

datos[2,] é a 2ª fila, e

datos[,3] a 3ª columna

Entón, para quitar filas e columnas podemos usar esa notación cun signo menos diante:

- Quitarlle unha fila

```
#quitar a 5ª fila
datos4=datos[-5,]

#quitar as filas 1, 2 e 5
datos5=datos[-c(1,2,5),]

#quitar desde a fila 6 á 45
datos6=datos[-(6:45),]

#6:45 significa precisamente desde 6 a 45
```

```
#un data.frame novo formado polas columnas 1 a 4 do data.frame datos
novo=datos[c(1:4)]

#o mesmo ca antes, pero só as 100 primeiras columnas
novo2=datos[1:100,c(1:4)]
```

Extraer data.frames dun data.frame

```
#a funcion cbind() "apega" vectores, matrices ou data.frames por columnas
outro.df=cbind(datos2,datos3)

#outro.df ten as mesmas filas ca datos2 e datos3, pero inclue todas as suas columnas
```

Unir data.frames **Ollo!**, os data.frames que peguedes con cbind() deben ter **o mesmo nº de filas**

Engadir filas a un data.frame Pode facerse con rbind():

```
#a funcion cbind() "apega" vectores, matrices ou data.frames por filas
outro.df2=rbind(datos,datos6)

#outro.df2 ten as mesmas columnas ca datos e datos6, pero inclue todas as suas filas
```

Ollo!: os data.frames que pegades por filas deben **ter o mesmo nº de columnas**, e ademais **cos mesmos nomes**

Crear un data.frame Os data.frame, ademais de producirse cando se le un ficheiro de datos, tamén poden ser creados mediante un comando. Se dispoñemos de vectores co mesmo tamaño podemos usar o comando ***data.frame()*** para agrupalos dentro dun data.frame.

```
#creando dous vectores para o exemplo:
nomes=c("Oscar","Luisa","Remedios","Filipe")
#os textos deben ir entre comiñas simpres ou dobres
idade=c(25,32,18,41)

novo.data.frame=data.frame(nomes,idade)
#poededes velo facendo
novo.data.frame
```

```
##      nomes idade
## 1    Oscar    25
## 2    Luisa    32
## 3 Remedios    18
## 4   Filipe    41
```

ANEXO

Listas

En R unha lista (*list*) pode describirse como un obxecto con estructura de vector pero que pode incluir elementos de diferente tipo.

Por exemplo o seu primeiro elemento pode ser un número, o segundo unha categoría, o terceiro un data.frame, ... Incluso pode incluir outra lista como elemento.

```
#Por exemplo, lemos o seguinte ficheiro:
lista1=list(4,"ola",data.frame(c(1,2,3),c("a","b","c")),list(c(1,2),"abur"))

lista1
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] "ola"
##
## [[3]]
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## [[4]]
## [[4]][[1]]
## [1] 1 2
##
## [[4]][[2]]
## [1] "abur"
```

Crear unha lista

Para crear unha lista úsase a función *list()*, incluindo como parámetros os diferentes obxectos que vai conter. Un exemplo do seu uso está no codigo anterior, onde se creou a lista *lista1* con catro elementos diferentes.

Crear unha lista valeira

Unha lista tamén pode ser creada sen elementos, unicamente coas posicións reservadas, para despois ir colocando nela os elementos que nos interesen.

Faise coa función vector

vector(mode = “logical”, length = 0)

usando esa función, podemos indicarlle o tipo de datos (o modo) que manexará e o seu tamaño.

Para crear unha lista valeira de 4 elementos, que terán o modo *list*

```
lista3=vector("list",4)
lista3
```

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

Despois incluranse diferentes elementos, indicando a posición que van ocupar entre corchetes:

```
lista3[4]="ola"
lista3
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## [1] "ola"
```

```
lista3[[2]]=c(2,1,3)
lista3
```

```
## [[1]]
## NULL
##
## [[2]]
## [1] 2 1 3
##
## [[3]]
## NULL
##
## [[4]]
## [1] "ola"
```

Nomes dos elementos da lista Nas listas, para manipulalas con comodidade, é práctico que cada elemento teña un nome. Iso pode conseguirse no momento de crealas:

```r
lista2=list(numero=4,palabra="ola",datos=data.frame(c(1,2,3),c("a","b","c")),vector=c(1,2))

lista2
```

```
## $numero
## [1] 4
##
## $palabra
## [1] "ola"
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## $vector
## [1] 1 2
```

```r
names(lista2)
```

```
## [1] "numero"  "palabra" "datos"   "vector"
```

Ou tamén indicándolle os nomes a posteriori, lembrando que debe haber un nome por cada elemento da lista

```r
names(lista1)=c("numero","palabra","datos","outralista")
```

Indexar unha lista

A cada elemento dunha lista vaille corresponder un índice, que pode ser o número de orde que ocupa ou pode ser o seu nome.

Para referirnos logo a eses elementos podemos usar un ou dous corchetes, ou o seu nome.

- **Un corchete** (*[]*)

Usando un corchete obtemos outro obxecto que **conserva a estructura de lista**

```r
str(lista1)
```

```
## List of 4
##  $ numero    : num 4
##  $ palabra   : chr "ola"
##  $ datos     :'data.frame':  3 obs. of  2 variables:
##   ..$ c.1..2..3.      : num [1:3] 1 2 3
##   ..$ c..a....b....c..: Factor w/ 3 levels "a","b","c": 1 2 3
##  $ outralista:List of 2
##   ..$ : num [1:2] 1 2
##   ..$ : chr "abur"
```

```
lista1[2]
```

```
## $palabra
## [1] "ola"
```

```
str(lista1[2]) #pode verse como o obxecto lista1[2] é de tipo list
```

```
## List of 1
##  $ palabra: chr "ola"
```

```
lista1[c(1,3)]
```

```
## $numero
## [1] 4
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
```

```
str(lista1[c(1,3)]) #o obxecto lista1[[c(1,3)]] segue a ser de tipo list
```

```
## List of 2
##  $ numero: num 4
##  $ datos :'data.frame':  3 obs. of  2 variables:
##   ..$ c.1..2..3.      : num [1:3] 1 2 3
##   ..$ c..a....b....c..: Factor w/ 3 levels "a","b","c": 1 2 3
```

- **Dous corchetes** (*[[]]*)

Usando dous corchete obtemos **o obxecto que está contido dentro do elemento da lista**

```
lista1[[2]]
```

```
## [1] "ola"
```

```
str(lista1[[2]]) #pode verse como o obxecto lista1[[2]] é de tipo character
```

```
##  chr "ola"
```

```
lista1[[3]]
```

```
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
```

```
str(lista1[[3]]) #pode verse como o obxecto lista1[[3]] é de tipo data.frame
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ c.1..2..3.      : num  1 2 3
##  $ c..a....b....c..: Factor w/ 3 levels "a","b","c": 1 2 3
```

- **Nomes**

Cando as listas teñen nome pódese extraer un elemento indicando o seu nome:

```
lista1$datos
```

```
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
```

```
str(lista1$datos) #compórtase igual que lista1[[3]]: extrae o obxecto da lista
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ c.1..2..3.      : num  1 2 3
##  $ c..a....b....c..: Factor w/ 3 levels "a","b","c": 1 2 3
```

Usando o nome do elemento temos un comportamento semellante ao que se ten usando corchetes dobres, extráese o obxecto que está dentro do elemento da lista.

Extraer subconxuntos dunha lista

Se queremos extraer elementos individuais dunha lista podemos usar o seu índice tal é como se viu no apartado anterior. En cambio, se queremos **extraer unha parte da lista** hai que facelo **empregando os corchetes simples**, xa que así obtemos unha nova lista que incluirá parte dos elementos da lista orixinal.

```
ola1=lista2[2] #outra lista co segundo elemento de lista2
ola1
```

```
## $palabra
## [1] "ola"
```

```
ola2=lista2[1:2] #outra lista co primeiro e segundo elementos de lista2
ola2
```

```
## $numero
## [1] 4
##
## $palabra
## [1] "ola"
```

```r
ola3=lista2[c(1,4)] #outra lista cos elementos 1 e 4
ola3
```

```
## $numero
## [1] 4
##
## $vector
## [1] 1 2
```

```r
ola4=lista2[-1] #agora quitámoslle a lista o primeiro elemento
ola4
```

```
## $palabra
## [1] "ola"
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## $vector
## [1] 1 2
```

```r
ola5=lista2[-c(2,4)] #agora quitámoslle o 2º e 4º elementos
ola2
```

```
## $numero
## [1] 4
##
## $palabra
## [1] "ola"
```

Extraer elementos dun elemento da lista

Se usamos **dobres corchetes ou o nome de elemento**, veuse que se obtén o obxecto incluido dentro do elemento da lista. Esta característica pódese aplicar para traballar cos elementos da lista, e incluso facerlle modificacións

```r
lista2[[4]] #é un vector, polo que podemos manexalo como calquera outro vector
```

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

```r
lista2[[4]][1] #O primeiro elemento do vector
```

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

```r
length(lista2[[4]]) #tamaño do vector
```

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

```
lista2[[4]][3]=18 #engadirlle ao vector un novo elemento
lista2[[4]]=lista2[[4]][-1] #quitarlle ao vector o primeiro elemento
lista2[[4]]
```

```
## [1]  2 18
```

Todo iso pode facerse tamén usando o nome do elemento, o que pode ser máis claro para manexar

```
lista2$vector #é un vector, polo que podemos manexalo como calquera outro vector
```

```
## [1]  2 18
```

```
lista2$vector[1] #O primeiro elemento do vector
```

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

```
length(lista2$vector) #tamaño do vector
```

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

```
lista2$vector[3]=23 #engadirlle ao vector un novo elemento
lista2$vector=lista2$vector[-1] #quitarlle ao vector o primeiro elemento
lista2$vector
```

```
## [1] 18 23
```

Na lista queda incorporado o cambio:

```
lista2
```

```
## $numero
## [1] 4
##
## $palabra
## [1] "ola"
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## $vector
## [1] 18 23
```

Manexar un vector dentro dunha lista é bastante simple, por que un vector é unha estructura simple. Máis complexo é manexar un data.frame ou outra lista que estean contidos dentro dunha lista.

```
#Un data.frame coa primeira columna do data.frame lista1[[3]]
lista1[[3]][1]
```

```
##   c.1..2..3.
## 1          1
## 2          2
## 3          3
```

```
#Extraer o vector do data.frame anterior
lista1[[3]][[1]]
```

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

```
#Cambiarlle o nome as columnas do data.frame
names(lista1[[3]])=c("d1","d2")

#Engadirlle outra columna (ten que ter 3 elementos como as demais)
lista1[[3]][[3]]=c(4,12,9)

#extraer un data.frame coas filas 1 e 2 das columnas 1 e 3
lista1[[3]][c(1,2),c(1,3)]
```

```
##   d1 V3
## 1  1  4
## 2  2 12
```

Con unha lista sería algo semellante, pero con máis complicación ainda

```
#Unha lista co primeiro elemento da lista lista1[[4]]
lista1[[4]][1]
```

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

```
#Extraer o vector da lista anterior
lista1[[4]][[1]]
```

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

```
#Extraer un elemento do vector anterior
lista1[[4]][[1]][2]
```

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

Engadir un elemento a unha lista

- Usando o número:

Se a lista ten N elementos engádese o elemento N+1:

```
length(lista2) #se ten 4 elementos o engadido será o quinto
```

```
## [1] 4
```

```
lista2[[5]]=c(8,9,3,4)

#Pode automatizarse usando unha variable
tam=length(lista2)
lista2[[tam+1]]=c("luns","martes","mercores")
```

- Usando un nome

```
lista2$meses=c("xaneiro","febreiro","marzo","abril","maio")
lista2
```

```
## $numero
## [1] 4
##
## $palabra
## [1] "ola"
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## $vector
## [1] 18 23
##
## [[5]]
## [1] 8 9 3 4
##
## [[6]]
## [1] "luns"     "martes"   "mercores"
##
## $meses
## [1] "xaneiro"  "febreiro" "marzo"    "abril"    "maio"
```

Eliminar un elemento da lista

Podese facer "anulando" o elemento, ou sexa, igualando o elemento da lista a *NULL*

```
lista2[[5]]=NULL

#Ou con nome
lista2$meses=NULL
lista2
```

```
## $numero
```

```
## [1] 4
##
## $palabra
## [1] "ola"
##
## $datos
##   c.1..2..3. c..a....b....c..
## 1          1                a
## 2          2                b
## 3          3                c
##
## $vector
## [1] 18 23
##
## [[5]]
## [1] "luns"     "martes"   "mercores"
```

Transformar cousas en listas e viceversa

Se temos unha matriz, un vector ou un data.frame podemos darlle estructura de lista simplemente coa función *as.list()*

```
X=data.frame(un=c(1,2,3),dous=c("luns","martes","mercores"))
X.l=as.list(X)
X.l
```

```
## $un
## [1] 1 2 3
##
## $dous
## [1] luns     martes   mercores
## Levels: luns martes mercores
```

O contrario tamén se pode facer sempre e cando o permita a estructura da lista:

```
X.d=as.data.frame(X.l)
X.d
```

```
##   un     dous
## 1  1     luns
## 2  2   martes
## 3  3 mercores
```

```
#Se os elementos teñen diferente tamaño non se pode facer
as.data.frame(lista1)
```

O que si se pode facer é organizar todos os elementos dos elementos dunha lista como un vector, mediante a función *unlist()*

```
unlist(lista1)
```

```
##      numero     palabra   datos.d11   datos.d12   datos.d13   datos.d21
##         "4"       "ola"         "1"         "2"         "3"         "1"
##   datos.d22   datos.d23   datos.V31   datos.V32   datos.V33 outralista1
##         "2"         "3"         "4"        "12"         "9"         "1"
## outralista2 outralista3
##         "2"      "abur"
```