

Introducció a l'anàlisi de dades amb R



Índex

1 INTRODUCCIÓ AL R I A L'ANÀLISI DE DADES.....	4
1.1 QUÈ ÉS R I PER A QUÈ SERVEIX	4
1.2 INSTAL·LACIÓ D'R.....	4
1.3 INSTAL·LACIÓ D'RSTUDIO	7
1.4 INTRODUCCIÓ A L'ENTORN DE PROGRAMACIÓ RSTUDIO	9
1.5 IDEES CLAU: INTRODUCCIÓ AL R I A L'ANÀLISI DE DADES.....	11
2 PROGRAMACIÓ AMB R.....	11
2.1 DIFERENTS TIPUS D'OBJECTES EN R (I).....	12
2.2 DIFERENTS TIPUS D'OBJECTES EN R (II).....	14
2.3 CONDICIONALS.....	16
2.4 RESOLUCIÓ DELS EXERCICIS CONDICIONALS.....	19
2.5 FUNCIONS (I)	22
2.6 FUNCIONS (II)	24
2.7 RESOLUCIÓ DELS EXERCICIS FUNCIONS	26
2.8 BUCLES (I)	28
2.9 BUCLES (II)	28
2.10 RESOLUCIÓ D'EXERCICIS: BUCLES	32
2.11 IDEES CLAU: PROGRAMACIÓ AMB R	34
3 TREBALLAR AMB BASES DE DADES	35
3.1 IMPORTAR I EXPORTAR BASES DE DADES	35
3.2 FILTRAR BASES DE DADES	37
3.3 RESOLUCIÓ DE L'EXERCICI PER FILTRAR BASES DE DADES	40
3.4 TRANSFORMAR BASES DE DADES.....	42
3.5 RESOLUCIÓ DE L'EXERCICI PER TRANSFORMAR BASES DE DADES.....	45
3.6 IDEES CLAU: TREBALLAR AMB BASES DE DADES	47

4 VISUALITZACIÓ DE DADES.....	47
4.1 INTRODUCCIÓ A LA VISUALITZACIÓ DE DADES.....	47
4.2 GRÀFICS DE BARRES I DIAGRAMES DE SECTORS	51
4.3 HISTOGRAMES I DIAGRAMES DE CAIXA	53
4.4 NÚVOL DE PUNTS I RECTA DE REGRESSIÓ (I).....	55
4.5 NÚVOL DE PUNTS I RECTA DE REGRESSIÓ (II).....	57
4.6 PERSONALITZACIÓ DE GRÀFICS (I)	58
4.7 PERSONALITZACIÓ DE GRÀFICS (II)	60
4.8 AFEGIR LLEGENDES ALS NOSTRES GRAFICS	62
4.9 IDEES CLAU: VISUALITZACIÓ DE DADES	64

1 INTRODUCCIÓ AL R I L'ANÀLISI DE DADES

En aquest primer mòdul explicarem el procés d'instal·lació d'R i la seva interfície gràfica RStudio.

En segon lloc, presentarem R com a llenguatge de programació, fent especial èmfasi en la utilitat pràctica que podem donar-li en els nostres processos d'anàlisi de dades.

Posteriorment, exposarem les principals utilitats que ens ofereix aquesta interfície gràfica i explorarem les maneres més senzilles de treure-li el major potencial possible. Aprendrem a crear i guardar un script, aprofitar la quadrícula d'RStudio per gestionar la informació disponible i a fer un ús eficient de les seves funcionalitats.

1.1 QUÈ ÉS R I PER A QUÈ SERVEIX

Hola a tots i a totes!

R és un llenguatge de programació de llicència oberta i totalment gratuït per a la computació estadística i la creació de gràfics. Recentment ha anat incrementant la seva popularitat dins de l'àmbit de la ciència de dades, i juntament amb Python i Java ocupa les primeres posicions d'aquest sector en ampli creixement.

Com molts altres llenguatges de programació, es fonamenta en l'escriptura en línia d'ordres. Tot i així, està extremadament estès l'ús de la interfície gràfica RStudio, per la seva practicitat i estructura. R també destaca per tenir una comunitat molt activa que desenvolupa paquets, especialment orientats a la modelització, l'estadística més clàssica i la visualització de dades, que són els punts forts del llenguatge. R disposa d'un repositori de paquets extensíssim, on podem trobar paquets de pràcticament totes les variants de la ciència de dades.

A continuació, farem una llista d'algunes de les principals particularitats d'R, que hauríem de conèixer com a usuaris i usuàries:

1. R és lent. Això pot semblar curiós com a primera característica, però té una justificació. R va ser dissenyat per fer fàcil a l'usuari o usuària la modelització i l'anàlisi estadística, no per ser el llenguatge més amable per a un ordinador. Hem de ser conscients que no pot competir en velocitat amb altres llenguatges, però sí guanyar en usabilitat.
2. R és extremadament popular en alguns sectors, i en canvi, pràcticament desconegut en d'altres. Està pràcticament desaparegut en un entorn més orientat a l'enginyeria o la informàtica, però domina en la indústria farmacèutica, les finances, el màrqueting, els audiovisuals i sobretot l'acadèmia, ja que és on més s'usa l'estadística formal, un dels punts més forts d'R. La seva importància com a eina de Business Intelligence és cada vegada més gran, principalment per la simplicitat del seu codi i les facilitats que ofereix a l'hora de fer visualitzacions i resums estadístics.

3. Els models en R són increïblement fàcils d'utilitzar. Com que R és un llenguatge que prioritza la usabilitat, la modelització i el seu codi associat són molt més senzills que en altres llenguatges i resulten més propers per a persones no expertes en *data science*.
4. R funciona molt més ràpidament si fem servir un tipus de codi orientat a l'ús d'estructures vectorials i planifiquem correctament les variables que necessitarem. A diferència de Python, per exemple, alguns bucles i especialment la concatenació de variables poden ser molt ineficients. Però, com ja hem comentat, una bona planificació i l'ús d'alternatives que aprofitin l'estructura vectorial del llenguatge per executar-se ràpidament poden ajudar a compensar el punt anterior, que és la seva baixa velocitat.
5. R és més complicat d'aprendre al principi. Tot i així, quan ja haguem entrat en la seva lògica i ja hi haguem realitzat un parell de projectes, el seu codi ens semblarà molt intuïtiu i aprendrem noves funcionalitats molt més ràpidament. Els primers passos programant amb R són més complexos que altres llenguatges més humans com Python, on cada tipus d'objecte té unes funcions associades. A R, aquestes funcions, per exemple, les associades a un tipus concret de model, haurem de conèixer-les per endavant o recórrer a l'ajuda que ens ofereix el programa, la qual cosa alenteix el procés d'escriptura del codi al principi. Afortunadament, l'ajuda que ens ofereix R és molt completa i plena d'exemples, i en la gran majoria de paquets s'inclouen bons tutorials.

En resum, R és un llenguatge de programació orientat a l'anàlisi estadística i la visualització de dades. Pot ser més complicat d'aprendre, al principi, que altres llenguatges, però, a causa de la seva creixent popularitat i versatilitat, el seu domini constitueix una competència de gran valor en el mercat laboral.

Seguim!

1.2 INSTAL·LACIÓ D'R

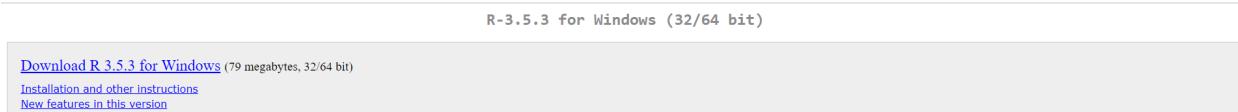
Hola de nou!

En aquest article avançarem en el procés d'instal·lació d'R, veurem que és molt senzill, independent del sistema operatiu amb el qual estiguem treballant.

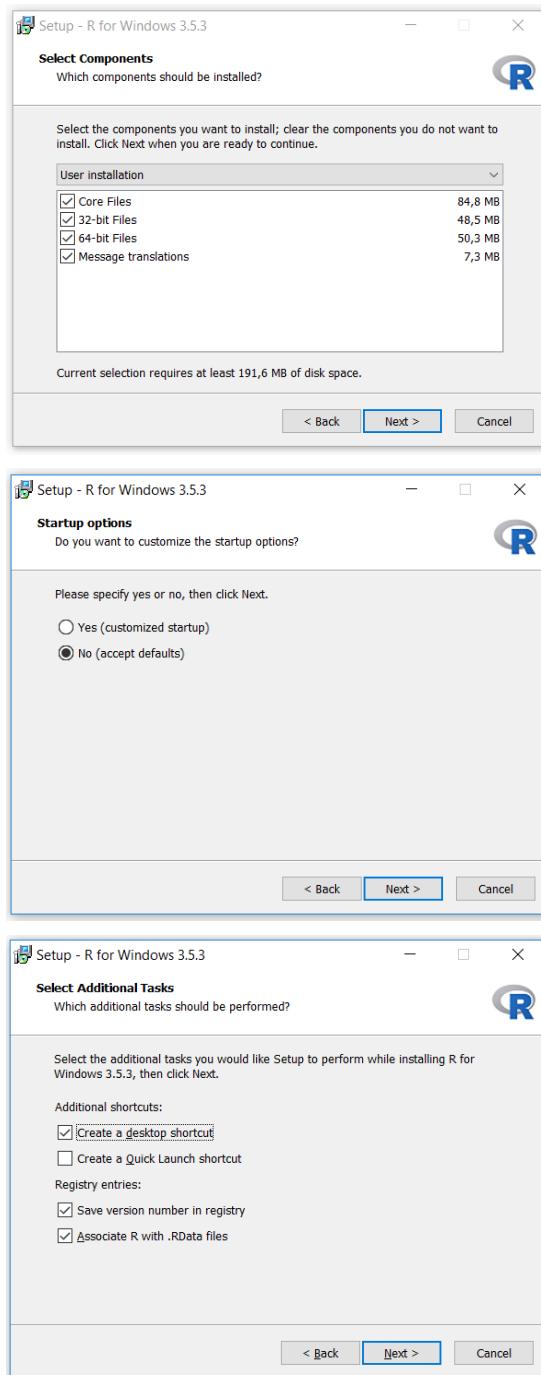
El primer pas que realitzarem és buscar “Install R” a Google i entrarem a la primera pàgina que ens aparegui.

Potser la web no és prou amable, ja que es tracta d'un projecte lliure. Haurem de clicar a “**Download R**”, ressaltat en negreta, i ens redirigirà a una pàgina de servidors.

En principi, hauria de ser indiferent quin d'ells seleccionem, així que clicarem el primer. A continuació haurem de seleccionar el nostre sistema operatiu. En aquest cas, utilitzarem Windows per a aquest tutorial, tot i que els passos són pràcticament idèntics per a tots els sistemes operatius.



A la pàgina següent, tornarem a escollir l'opció ressaltada en negreta “**Install R for the first time**”. El següent pas és el definitiu i descarregarem el fitxer d'instal·lació, que podem executar perquè se'n obri l'assistent d'instal·lació.



Fins aviat!

1.3 INSTAL·LACIÓ D'RSTUDIO

Hola de nou!

En aquest article avançarem en el procés d'instal·lació de la interfície gràfica RStudio, una interfície gràfica que treballa sobre R i que ens facilitarà enormement la feina, a la vegada que ens permetrà una visualització més agradable i estructurada de l'entorn de programació.

Com és d'imaginar, hem de tenir completament instal·lada l'última versió disponible d'R per dur a terme aquest procés.

El primer pas que realitzarem és buscar “Install RStudio” a Google i entrarem a la primera pàgina que ens aparegui.

Google

download rstudio for windows

All Videos Images News Shopping More Settings Tools

About 2,750,000 results (0.45 seconds)

[Download RStudio - RStudio](#)

<https://www.rstudio.com/products/rstudio/download/> ▾

Feb 7, 2019 - Download the RStudio IDE or RStudio Server. ... RStudio 1.1.463 - Windows Vista/7/8/10, 85.8 MB, 2018-10-29 ...

[Download RStudio Server](#) · [Release Notes](#) · [Code Signing](#) · [IDE features](#)

Aquí ens oferirà diverses opcions; nosaltres escollirem la primera de totes, que és gratuïta i completament funcional.

RStudio Desktop
Open Source License

FREE

[DOWNLOAD](#)

[Learn More](#)

Quan cliquem el botó “**Download**”, ens portarà més avall a la mateixa pàgina, on podrem escollir el nostre sistema operatiu. És tan senzill com clicar la primera opció, en el cas de Windows.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.463 - Windows Vista/7/8/10	85.8 MB	2018-10-29	58b3d796d8cf96fb8580c62f46ab64d4

Se'ns descarregarà el fitxer que obrirà l'instal·lador. Caldrà anar clicant a següent fins que es completi la instal·lació.

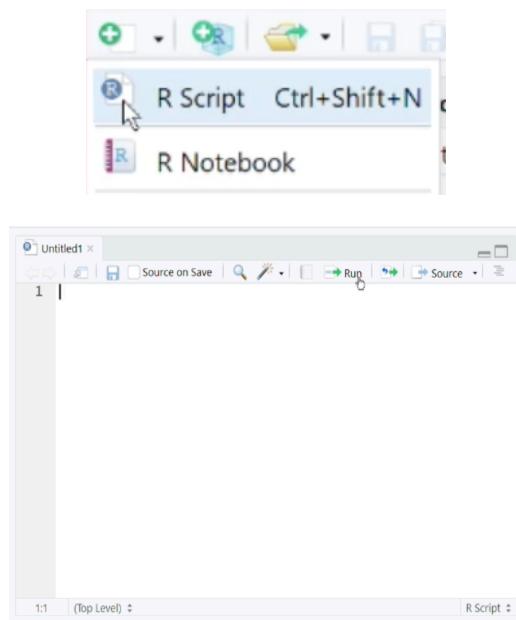
Continuem amb el curs!

1.4 INTRODUCCIÓ A L'ENTORN DE PROGRAMACIÓ RSTUDIO

Benvinguts i benvingudes!

En aquest vídeo presentarem la interfície gràfica RStudio, així com les funcionalitats més interessants que ens pot oferir. Abans de comentar una mica més en detall el que ens trobarem, hem de tenir en compte un consell. Si tenim un cert domini de l'anglès, el més pràctic és veure la interfície en aquest idioma. D'aquesta manera, en el cas de necessitar ajuda en algun moment, serà molt més senzill trobar-la a través de Google i aplicar-la.

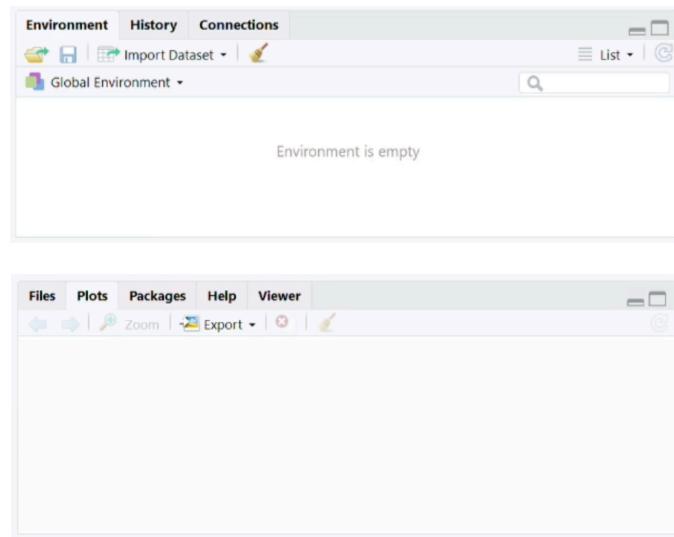
L'estructura d'RStudio es basa en una graella. És a dir, en quatre elements. Ara mateix, només en veiem tres. El que passa és que encara no hem creat un *script*. L'*script* el podem crear utilitzant aquesta icona d'aquí:



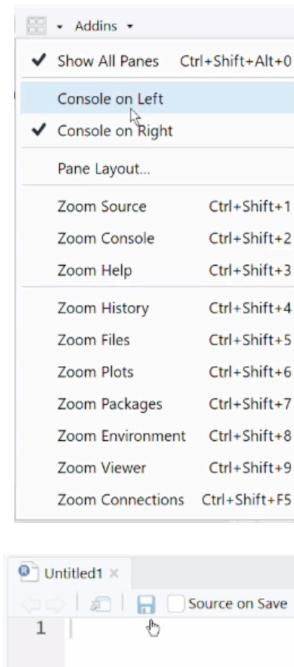
I obrint el document on escriurem el nostre codi. Tot el que escrivim aquí i executem utilitzant aquest botó d'aquí, ho veurem per la consola.



Aquí, a l'entorn, veurem els objectes que hem creat i aquí podrem veure, per exemple, els gràfics que estiguem generant.



Els elements més importants són aquests dos d'aquí dalt. Utilitzant aquest botó d'aquí, podem moure la consola de banda, per exemple. L'hem posat a l'esquerra i ara tindríem un espai ajustable, on podem veure el nostre *script*, la nostra consola i, en una altra banda, podríem veure el nostre entorn. Guardar un *script* és tan senzill com utilitzar aquest botó d'aquí.



Una altra funció molt important que necessitem saber si treballem amb RStudio és aquesta d'aquí: el Working Directory.



És a dir, on volem que se'ns guardin els arxius. Si cliquem aquí, el que estem aconseguint és que tots els gràfics que generem i objectes nous que vulguem guardar es guardin exactament a la mateixa carpeta on tenim l'*script* que acabem de generar. També podem escollir-lo utilitzant aquesta opció d'aquí. Aquí dalt tenim una gran quantitat de menús que permeten personalitzar com veiem el nostre RStudio. Tenim les opcions globals, on podem, per exemple, ajustar la visualització del nostre R.

Aquests ajustaments són purament estètics. El que és especialment útil a l'hora d'utilitzar RStudio, que no ens ofereix R, és aquesta visualització en format graella, on podem veure els nostres gràfics, els paquets que vulguem carregar, ajuda, un entorn on veurem les variables, un històric del que haguem fet, una consola, l'*script* i tindrem una ajuda en el cas de necessitar, per exemple, estilitzar el nostre codi utilitzant aquestes funcions d'aquí.

Seguim!

1.5 IDEES CLAU: INTRODUCCIÓ AL R I A L'ANÀLISI DE DADES

En aquest mòdul hem realitzat els primers passos amb R i RStudio. A continuació, farem un repàs de les idees més importants que han aparegut:

- R és un conegut llenguatge de programació que ha crescut enormement en els darrers anys. Es caracteritza per estar totalment orientat a la usabilitat, modelització i visualització, i està àmpliament estès en molts sectors.
- Hem instal·lat R i la seva interfície gràfica RStudio, a la vegada que hem explorat les funcionalitats principals d'RStudio, com per exemple:
 - Visualització dels objectes creats
 - Estructura i llegibilitat del codi
 - Assistents d'importació
 - Gestió de *scripts* i de projectes

2 PROGRAMACIÓ AMB R

En aquest mòdul veurem un resum molt breu dels fonaments d'R com a llenguatge de programació.

Començarem presentant els quatre grans tipus d'objectes amb els quals podem treballar, quines són les seves particularitats i per a què són útils.

A continuació, veurem com funcionen les estructures condicionals més senzilles i algunes de més complexes, el mateix amb bucles de diferents tipus, fent especial èmfasi en les seves condicions d'aturada.

Al llarg d'aquest mòdul veurem, de manera transversal, algunes de les funcions bàsiques d'R

2.1 DIFERENTS TIPUS D'OBJECTES EN R (I)

Hola de nou!

En aquest vídeo, veurem els diferents tipus d'objecte que podem crear en R, quina és la seva utilitat principal i com podem tractar-los adequadament. Veurem com crear variables unitàries, vectors i matrius.

El primer que farem és crear una variable unitària. Li assignem un nom i, utilitzant aquesta fletxeta, li assignarem un nombre.

```
num <- 3
```

Per executar això que acabo d'escriure, puc prémer a *Run* o puc prémer *Control+Enter*. El que podem veure és que ens apareix al nostre entorn aquesta variable que acabem de crear i la instrucció que hem executat a la nostra consola.

```
num <- 3.0
```

Una altra cosa que podem fer és crear un text. Tota la creació segueix la mateixa estructura: un nom i una fletxeta. Li poso aquestes quatre lletres, executo i ara he creat una variable que es diu “text” i que conté aquestes quatre.

```
text <- "asdf"
```

Un altre tipus d'estructura és el vector. Li diré de nom “*numeros*” i amb la fletxeta i aquesta instrucció d'aquí: *c*, obro parèntesi i li puc afegir els números que jo vulgui.

```
numeros <- c(1,2,3)
```

Executo i veig que aquí baix em diu el seu nom, el tipus de variable que és, és a dir numèrica, i que té tres posicions, 1, 2 i 3.

Una manera alternativa de crear aquest mateix objecte podria ser aquesta d'aquí.

```
numeros <- 1:5
```

Fem-lo de l'1 fins al 5. Executo i el que veig és que m'ha creat un objecte amb una seqüència de l'1 fins al 5. Una consideració que cal fer sobre els vectors és que tots els elements que el

composen han de ser del mateix tipus. M'explico: si jo creo aquest objecte, poso 1, 2, 3 i hi afegeixo un 4 però en format text, que és el que delimiten aquestes cometes d'aquí.

```
numeros <- c(1,2,3,"4")
```

Executo. El que haurà passat és que he creat un objecte amb quatre posicions on totes elles són variables textuals, és a dir, el tipus més flexible de variable del text.

Obté aquest nom de variable.

```
vector_variat <- c(5,TRUE,"Hola")
```

Els espais no estan permesos, així que afegeixo aquesta barra baixa. Puc posar-hi un nombre, una variable binaria i un text. Si executo aquesta instrucció, creo un nou vector on tot és format text, i tinc un 5 en format text, una variable binaria en format text i un text.

L'últim que veurem en aquest vídeo és com crear una matriu. Li diré de nom “matriu” i la crearem amb la instrucció *matrix*, de l'1 fins al 12, i li puc especificar el nombre de columnes que vull que tingui. Li estic dient: “posa'm els números de l'1 fins al 12 en tres columnes”.

```
matriu <- matrix(1:12,ncol = 3)
```

Executem i visualitzem el que acabem de crear. Executo i tinc aquesta estructura d'aquí: tres columnes, quatre files i els nombres me'ls ha omplert en vertical. Puc accedir als elements d'aquesta matriu, utilitzant els claudàtors. Per exemple, puc agafar l'objecte que ocupa la primera fila i la segona columna.

```
matriu[1,2]
```

A la primera fila i a la segona columna serà el número 5.

El mateix podríem fer dient-li que ens agafi la primera fila i la segona i tercera columna.

```
matriu[1,2:3]
```

També podem ometre un d'aquests dos elements i, si ho deixem en blanc, el que estaríem fent és escollir només la primera fila.

```
matriu[1,]
```

2.2 DIFERENTS TIPUS D'OBJECTES EN R (II)

Hola!

En aquest segon vídeo, explorarem com crear i manipular dataframes i llistes, els objectes més versàtils i utilitzats en R.

El primer que farem és carregar el paquet *datasets*.

```
require(datasets)
```

El que obtindrem dins d'aquest paquet són un conjunt de bases de dades d'exemple. Utilitzarem aquesta, que consisteix en un conjunt de cotxes amb les seves característiques.

```
mtcars
```

Guardarem aquesta base de dades com *df*, que és la notació clàssica per dataframe.

```
df<-mtcars
```

L'executem i el que podem veure és com accedim a la informació d'una de les columnes. La sintaxi que s'utilitza és el \$ i ens apareix una llista amb les columnes que té aquest objecte. Si executo aquesta instrucció, puc obtenir un vector amb cadascun dels elements de la columna de cilindres per a cada un d'aquests cotxes.

```
df$cyl
```

Una manera alternativa de fer-ho és utilitzant els claudàtors.

```
df["cyl"]
```

L'output que obtenim és lleugerament diferent, ja que aquí ens apareixen els noms de les files, cosa que és més pràctic. Si vull obtenir diverses columnes a la vegada, puc utilitzar aquesta sintaxi d'aquí.

```
df[1:3]
```

Si, en canvi, el que faig és afegir-hi una coma (,) i hi poso un nom de columna, com si fos un text, ara el que obtinc són els valors per a les tres primeres files de la columna “cilindres” (cyl).

```
df[1:3,"cyl"]
```

Aquesta mateixa estructura la puc utilitzar en format vectorial i el que puc aconseguir és seleccionar més d'una columna alhora. Per exemple seleccionarem la columna “mpg”, tres files i les dues columnes que li hem demanat.

```
df[1:3,c("cyl","mpg")]
```

Anem a veure un altre tipus d'estructura, que és la llista. Una llista es crea utilitzant aquesta funció.

```
llista <- list()
```

Si la creem buida, tenim una llista de zero elements. Per afegir elements a una llista, podem utilitzar també el \$. Per exemple, crearem un primer element d'aquesta llista i li direm “objecte1”, i aquí hi guardarem un vector amb tres nombres.

```
llista$objecte1 <- c(1,2,4)
```

Ara tenim una llista amb un element, no amb tres, ja que en aquesta llista el primer element és un vector. El que puc fer és crear-ne un altre. Aquí hi guardarem “Aixo es un text” i tenim una llista de dos elements.

```
llista$objecte2 <- "Aixo es un text"
```

Com pots comprovar, és un format molt flexible, ja que puc guardar un vector, un text i també hi podem guardar, per exemple, una fracció del nostre dataframe. Per exemple, les tres primeres files.

```
llista$objecte3 <- df[1:3,]
```

Per accedir a la informació que hem guardat, podem visualitzar-la així i veuriem el primer element, un vector; el segon element, un text; i el tercer element, una base de dades. Hi podem accedir o pels noms, com fèiem amb els dataframes, o utilitzant els dobles claudàtors.

```
llista[[1]]
```

Podríem obtenir el primer element així i, d'aquest primer element, podríem obtenir la segona posició, utilitzant aquesta sintaxi d'aquí.

```
llista[[1]][2]
```

Així doncs, si volguéssim explorar el dataframe que hem guardat aquí, hauríem de fer “*llista*”, “\$”, “*objecte3*”

```
llista$objecte3
```

que seria aquesta estructura d'aquí. I ara podríem fer, per exemple, selecciona'm la columna “*cyl*”.

```
llista$objecte3[,"cyl"]
```

I obtindríem aquests tres valors d'aquí en format vectorial.

2.3 CONDICIONALS

Hola a totes i a tots!

A continuació, veurem què són les condicions en un llenguatge de programació i com podem crear les nostres pròpies amb R.

Una condició és aquella estructura de control que utilitzem per decidir, basant-nos en un cert criteri, si volem realitzar alguna acció o no. La manera més senzilla de veure com funcionen les condicions és amb un exemple:

El primer que farem és crear una nova variable que li direm “*edat*”, que representarà la nostra edat.

```
(Edat <- 26)
```

L'estructura condicional que utilitzarem és aquesta d'aquí.

```
if(edat >= 18){
```

On, aquí a dins del parèntesi, especificarem una condició, per exemple, si la meva edat és més gran o igual a 18 anys, i aquí dins d'aquests claudàtors quina acció volem realitzar.

```
print("Major d'edat")  
}
```

Si executem aquesta variable i l'estructura condicional, obtenim que l'individu és major d'edat; si, en canvi, modifiquem aquesta variable, aquesta estructura no ens retorna absolutament res.

```
Edat <- 16  
if(edat >= 18){  
  print("Major d'edat")  
}
```

El que podem fer, però, és millorar aquesta estructura, dient-li què volem que faci en el cas contrari, “en cas que aquesta condició no es compleixi, fes-me això d'aquí”.

```
else{  
  print("Menor d'edat")  
}
```

Si executem tota aquesta estructura, el que ens retorna és que l'individu és menor d'edat.

Una manera alternativa de realitzar aquestes estructures és utilitzant la funció “*ifelse*”. La funció “*ifelse*” depèn de tres paràmetres: la nostra condició, que serà l'edat.

```
ifelse(test=edat>=18)
```

Què volem que faci, si es compleix, això d'aquí.

```
ifelse(test=edat>=18, yes=print("Major d'edat"))
```

I, què volem que faci, quan no es compleix.

```
ifelse(test=edat>=18, yes=print("Major d'edat"), no=print("Menor d'edat"))
```

Si l'executem, veiem que ens està donant la segona resposta.

Aquesta és una introducció molt breu al funcionament de les condicions amb R. Aquí hem proposat una estructura basada en una sola condició, que, si es complia, imprimia un resultat i, si no es complia, n'imprimia un altre. Evidentment, podem definir condicions múltiples, com per exemple creant una nova variable que sigui la nostra edat.

```
edad <- 20
```

I si l'individu és home o dona.

```
sexe <- "H".
```

Utilitzant l'estructura “*if*”, podem preguntar si l'edad és més gran o igual a 18 anys i utilitzant “&” el sexe és igual a “Home”. Aquesta estructura d'aquí pregunta si es compleix aquesta condició i aquesta altra.

```
if(edat >= 18 & sexe == "H")
```

Aquí també hem introduït la comparació d'igualtat. Aquí estàvem fent una desigualtat en la qual incloíem l'igual (\geq) i aquí estem fent exactament una igualtat ($=$). Volem que ens digui “Home adult”.

```
if(edat >= 18 & sexe == "H"){
  print("Home adult")
}
```

Així doncs, executem aquestes variables i, ara, quan executem aquesta condició múltiple, ens dirà que les compleix ambdues, ja que és el que li estem preguntant. Per exemple, si canviéssim “Home” a “Dona”, ara executem aquesta condició i ja no es compleix.

```
sexe <- "D".
```

Podem modificar aquestes condicions perquè s'adaptin a tot tipus de circumstàncies. Per exemple, aquest símbol d'aquí, ($|$) la ratlla en vertical, indica una condició o l'altra condició.

```
if(edat >= 18 | sexe == "H"){
print("Home adult")
}
```

Si l'executem, ens sortirà “*Home adult*”. Per què? Perquè estàavaluant si l'edat és més gran que 18, que en aquest cas és veritat, o si és home. En aquest cas no és veritat, però com que ja es complia la primera, ens imprimeix el que tenim aquí dins.

Una última cosa és que podem modificar aquesta doble igualtat per una desigualtat. Aquí, el que li estem dient a R és que ens comprovi si el sexe no és igual,(“!=”) a home.

```
if(edat >= 18 | sexe != "H"){
print("Home adult")
}
```

I fins aquí aquesta introducció a les condicions. Continuem!

A continuació us proposem uns exercicis on haurem d'utilitzar múltiples vegades la instrucció `if()` per generar estructures condicionals més complexes. La dificultat d'aquestes estructures rau en on definim les diferents condicions, i en quin ordre ho fem!

Exercicis:

1. Crea el codi que, facilitant un any de naixement concret, et digui si has nascut abans del 80, entre el 1980 i el 1999, o a partir de l'any 2000.
2. La instrucció `ifelse()` també funciona quan l'apliquem en vectors. Crea el vector `c(-5, 4, 8, -1)`, utilitzant-la, aconsegueix el vector `c("Negatiu", "Positiu", "Positiu", "Negatiu")`.
3. Utilitzant els parèntesis adequats, que determinen la importància de les operacions, i una sola condició, troba una manera de detectar si un número és inferior a 18 o superior a 99, i al mateix temps és parell! Hauràs d'utilitzar la condició següent: `numero %% 2 == 0`. Explora'n el funcionament abans de començar l'exercici! Què fa `numero %% 2`?

Per resoldre els exercicis, trobarem la solució a continuació.

2.4 RESOLUCIÓ DELS EXERCICIS CONDICIONALS

Hola de nou!

A continuació veurem com resoldre els exercicis plantejats.

El primer que demanava era fer un filtre que classifiqués un any en funció d'altres categories.

`Any <- 1993`

Partint d'aquest any, mirarem si és més gran o igual a l'any 2000. Si ens trobem en els 80 o 90 o abans dels 80. El primer que fem és comprovar si aquest any és més gran o igual que l'any 2000. En cas afirmatiu, imprimim aquesta instrucció.

```
if(any >=2000){  
print("A partir dels 2000")}
```

En cas contrari, realitzem tota aquesta estructura d'aquí, que és un condicional en si.

En el cas que aquesta condició no s'hagi complert, mirarem si l'any és més gran o igual que el 1980. En cas afirmatiu, ens trobem en els 80 i 90.

```
} else{  
if (any >= 1980){  
print("80s-90s")}
```

En cas contrari, per força, ens trobarem abans dels 80.

```
} else{  
print("Abans dels 80")  
}
```

Si executem tota aquesta instrucció, obtenim el resultat desitjat.

```
if(any >=2000){  
print("A partir dels 2000")}  
} else{  
if (any >= 1980){  
print("80s-90s")}  
} else{  
print("Abans dels 80")  
}
```

A continuació, mostrarem una solució per al segon exercici, que bàsicament et demanava que utilitzant aquest vector d'aquí “c(-5,4,8,-1)” classificuessis els nombres, segons si eren negatius o positius. Això és tan senzill com utilitzar la funció *ifelse* aplicant la condició que, si cadascun dels nombres és més petit que 0, executi aquesta part d'aquí (“Negatiu”) i, en cas contrari, aquesta d'aquí (“Positiu”).

```
numeros<-c(-5,4,8,-1)
ifelse(numeros < 0, "Negatiu", "Positiu")
```

Trio el vector, executo i obtinc exactament el que m'interessava. Aquest resultat d'aquí el puc guardar com un nou vector i així tinc els resultats en un objecte independent.

```
nouvector <- ifelse(numeros < 0, "Negatiu", "Positiu")
```

Aquí he omès la part *yes* i *no*, ja que, per posició, la primera sempre ocupa la part del *yes* i la segona la part del *no*.

```
nouvector <- ifelse(numeros < 0, yes="Negatiu", no="Positiu")
```

I, en aquest últim exercici, veiem com aplicar una condició múltiple. El que necessitem és crear una variable, que serà el nostre número .

```
numero <- 12
```

I, dins de la condició, li demanem dues coses, en realitat tres. Per una banda, que es compleixi una d'aquestes dues (*numero < 18 | numero > 99*) i per altra banda, que es compleixi aquesta (*numero %% 2 == 0*).

```
if((numero < 18 | numero > 99) & numero %% 2 == 0){
```

El primer que fem és dir-li que el meu nombre sigui més petit que 18 o més gran que 99. Això d'aquí es complirà, és a dir serà *true*, quan el meu nombre compleixi o aquesta (*numero < 18*) o aquesta (*numero > 99*). Per altra banda, buscaré que el residu de la divisió del meu nombre per 2 sigui igual a 0. També és cert.

Si miro les dues condicions conjuntament, s'ha de complir tant aquesta (*numero < 18 | numero > 99*) com aquesta (*numero %% 2 == 0*), ja que ho hem especificat amb el símbol *&*.

Si executem tota aquesta instrucció:

```

numero <- 12
if((numero < 18 | numero > 99) & numero %% 2 == 0){
print("Compleix totes les condicions")
}

```

Ens imprimeix per pantalla que el nombre compleix totes les condicions.

2.5 FUNCIONS (I)

Benvingudes i benvinguts de nou!

En aquest vídeo presentarem què són i com s'utilitzen les funcions a R. Podem simplificar la definició de què és una funció amb la frase següent: “és tota aquella instrucció a R que va seguida de dos parèntesis”. Normalment, les funcions es caracteritzen per dur a terme un procés molt determinat sobre algun dels objectes amb els quals estem treballant, tot i que no sempre és així.

Els exemples més clàssics de funcions són aquells que calculen estadístics sobre un vector. Per exemple, si tenim aquestes valoracions d'un producte, per exemple, i els apliquem una funció.

```
valoracions<- c(7,8,6,10,5,7,4,6,10)
```

La més famosa de totes és la mitjana.

```
mean(valoracions)
```

La mitjana, senzillament, suma tots aquests valors i els divideix entre la llargada.

Una altra funció seria, per exemple, la suma.

```
sum(valoracions)
```

I una altra, la llargada.

```
length(valoracions)
```

Utilitzant la combinació d'aquestes dues funcions,

`sum(valoracions) / length(valoracions)`

podríem obtenir exactament el mateix que en la funció *mean*.

Altres funcions molt interessants són, per exemple, la variància;

`var(valoracions)`

la desviació típica, que bàsicament és l'arrel de la variància;

`sd(valoracions)`

el mínim;

`min(valoracions)`

el màxim;

`max(valoracions)`

la mediana, que separa el 50 % de les observacions superiors del 50 % de les observacions inferiors;

`median(valoracions)`

i, una de les meves preferides, la funció *summary*,

`summary(valoracions)`

que segons l'objecte sobre el qual l'apliquem, obtenim un resultat o un altre. Sobre un vector, ens donarà el mínim, el primer quartil, la mediana, la mitjana, el tercer quartil i el màxim.

El primer i el tercer quartil són els punts que separen els 25 % de les observacions per sota i el 25 % de les observacions per sobre. Si volguéssim obtenir informació més detallada dels quantils, podríem utilitzar la funció *quantile*.

`quantile(valoracions)`

Si l'executem, ens donarà el mínim, el 25, el 50, el 75 i el 100. Exactament els mateixos que ens estava donant la funció *summary*. Però podem especificar-li, per exemple, quines probabilitats volem. Si volem el quantil 40 %, ho podem fer amb aquesta instrucció.

```
quantile(valoracions,probs=.4)
```

2.6 FUNCIONS (II)

Hem començat explicant com aplicar funcions ja existents a R, ja que amb la paciència suficient i amb l'ajuda de l'R o Google, segurament siguem capaços de trobar una funció que realitzi exactament el càlcul o procés que volem en pràcticament totes les situacions.

Tot i així, sempre podrem definir les nostres pròpies funcions. De què ens servirà això? Doncs permetrà guardar un codi que vulguem aplicar moltes vegades d'una manera molt compacta i definir el tipus de sortida o resultat que desitgem.

Vegem un parell d'exemples molt senzills:

Primer, crearem la nostra pròpia funció per calcular la mitjana. Definim un nom de funció, li direm “mitjana”, i, utilitzant la instrucció *function*, que dependrà d'uns paràmetres que posarem aquí dins, realitzarem un càlcul.

```
mitjana <- function(){
}
```

El càlcul que realitzarem és el que us hem mostrat abans, la suma de X dividit per la llargada. Així doncs, la meva funció mitjana dependrà d'un valor X, un paràmetre X, i el que farà és sumar-ho tot i dividir-ho per la seva llargada, exactament el càlcul de la seva mitjana.

```
mitjana <- function(x){
  sum(x) / length(x)
}
```

Si executo aquesta funció, diguem que m'apareix com una nova funció, la que acabo de definir ara mateix. Si creo un objecte i li dic “nombres”, de l'1 al 5, per exemple.

```
numeros<- c(1,2,3,4,5)
```

Puc executar-la directament i em calcula la mitjana d'aquests nombres.

```
mitjana(numeros)
```

Això que acabo d'escriure és equivalent a utilitzar la instrucció *return*.

```
mitjana <- function(x){  
  return(sum(x) / length(x))  
}
```

Això d'aquí, bàsicament, el que especifica és el que vull que em retorni la funció. Si no ho especificuem, senzillament retorna l'últim que troba, i en aquest cas era equivalent.

En segon lloc, utilitzant les condicions que ja hem vist anteriorment, detectarem si ens trobem en un any de traspàs (bàsicament si l'any és divisible per 4, per simplificar) o no.

Així doncs, creem aquesta funció, que dependrà d'un any.

```
anytraspas <- function(any)
```

I ara, aquí dins, crearem una estructura condicional: si l'any dividit per 4 dona exacte, ens retornarà “L'any té 366 dies”.

```
if( any %% 4 == 0){  
  return ("L'any té 366 dies")
```

En cas contrari, ens retornarà “L'any té 365 dies”.

```
else{  
  return("L'any té 365 dies")}
```

Com pots comprovar, podem utilitzar la instrucció *return* tantes vegades com vulguem. I sempre acaba el que fa la funció. Així doncs, si li afegíssim un *return* aquí baix, com que hauria trobat un d'aquests dos, aquest que haguéssim posat aquí ja no s'executaria.

```
anytraspas <- function(any){  
  if( any %% 4 == 0){  
    return ("L'any té 366 dies")  
  }else{  
    return("L'any té 365 dies")  
  }}
```

Executem la funció i ara veurem el que ens retorna si li posem 2004, per exemple, era un any de traspàs.

```
anytraspas(2004)
```

O el 2006, que no ho era.

```
anytraspas(2006)
```

En ser una funció, podem guardar aquest resultat en una variable. “Resultat 2006”, per exemple.

```
resultat2006 <- anytraspas(2006)
```

Executem i ara veiem que se'ns ha executat una variable textual que ens diu que l'any té 365 dies.

A continuació, proposem alguns exercicis sobre funcions. Trobarem la solució de com fer-ho més endavant:

1. Defineix una funció que imprimeixi en pantalla la diferència entre el màxim i el mínim d'un vector.
2. Modifica la funció anterior perquè, en el cas de rebre un sol nombre en lloc d'un vector de nombre, imprimeixi en pantalla una advertència. Pots fer-ho amb la funció `length()`.
3. Si definim una funció com multiplicar `<- function(a,b){return(a*b)}` i ho cridem com multiplicar(2,3), obtindrem el número 6. Així, podem especificar funcions que depenguin de més d'una variable! Crea una funció que depengui de dos números. La funció ha de retornar el número més gran i imprimir en pantalla si algun dels números és negatiu.

2.7 RESOLUCIÓ DELS EXERCICIS FUNCIONS

Hola de nou!

Vegem com resoldre els exercicis de funcions.

El primer que he fet és definir un vector i un número, que és el que utilitzarem per veure com funcionen les nostres funcions.

```
vector <- c(1,4,5,7,4,2,4,6,8,9,10,2)  
numero <- 7
```

La primera que he demanat crear és la funció *rang*, és a dir, la diferència entre el màxim i el mínim.

```
rang <- function(x){
  max(x) - min(x)
}
```

Aquí, com que senzillament estem aplicant aquesta diferència, no fa falta utilitzar la funció *return*. Si executem aquesta funció i ara la cridem sobre el vector, ens dirà la diferència entre el número més gran, 10, i el número més petit, 1.

```
rang(vector)
```

Si executem aquesta mateixa funció sobre el número, ens dona *rang 0*.

```
rang(numero)
```

El segon exercici busca corregir això, és a dir, busca que la nostra funció entengui que, si no li estem entrant un vector, és a dir, si la llargada de l'objecte que entra és igual a 1, és a dir, un número, no un vector, ens dirà: "ep, has introduït un número, no un vector". En cas contrari, serà exactament el que li hem demanat. Fixa't que en aquesta primera condició no retorna absolutament res. Senzillament avisa per pantalla.

```
rang <- function(x){
  if(length(x) == 1) {
    print("Has introduït un número, no un vector")
  }else{
    return(max(x) - min(x))
  }
}
```

Si executem aquesta funció, que estem sobreescrivint l'anterior, ara podem provar què fa quan li executem rang del número. Ja no ens dona 0, com anteriorment, sinó que ens diu: "Has introduït un número, no un vector".

En l'últim exercici et demanava una funció que et digués quin dels dos nombres que li donaves és més petit que l'altre, i que t'avisés si almenys un dels dos és negatiu.

```
mesgran <- function(numero1, numero2){
  if(numero1 < 0 | numero2 < 0){
    print("Almenys un dels nombres és negatiu")
```

```

}
if(numero1 > numero2){
return(numero1)
}
return(numero2)
}

```

Aquestes dues condicions no són excloents, és a dir, primer el que farem és comprovar si un dels dos nombres és més petit que 0, utilitzant aquesta condició (`numero1 < 0`) o aquesta altra (`numero2 < 0`); que ens avisarà que almenys un dels dos és negatiu i, independentment del que hagi passat fins ara, mirarà si el número 1 és més gran que el número 2: ens retornarà el número 1, és a dir, el més gran; i, en cas contrari, ens retornarà el número 2. Aquí no és necessària l'estructura `else`, ja que, si utilitzem aquest `return(return(numero1))`, sortim de la funció i, si aquesta condició no s'ha complert, arribem aquí (`return(numero2)`) i retornarà el número 2.

Anem a veure com funciona aquesta funció. L'executem i l'apliquem sobre el número 7 i el -8, per exemple.

```
mesgran(7,-8)
```

Executem i ens dona dos resultats: per una banda, el que ens està retornant, és a dir el número més gran i, per altra banda, ens avisa de què almenys un dels dos nombres és negatiu.

Si guardéssim el resultat d'aquesta funció en un objecte, per exemple, li direm “`obj`”,

```
obj <- mesgran(7,-8)
```

si mirem el que estem fent és mostrar un resultat per pantalla que no es guarda

```
obj <- mesgran(7,-8)
obj
```

i, dins de l'objecte, hem guardat el número 7, com podíem veure al nostre entorn.

2.8 BUCLES (I)

Benvingudes i benvinguts a aquest nou vídeo!

A continuació exposarem de manera molt breu què són els bucles i la seva utilitat pràctica, ja que són una de les estructures més importants dels llenguatges de programació, juntament amb els condicionals i els objectes.

Un bucle, bàsicament, és un procés que repetirem tantes vegades com definim, i que normalment s'utilitza per explorar els diferents elements d'un objecte, o realitzar alguna operació fins que es satisfaci alguna condició predefinida.

Vegem-ne un exemple pràctic. Si volem explorar aquest vector, aquesta llista de nombres de l'1 fins al 20, el que farà el bucle *FOR* que acabem de definir, que és segurament el més popular de tots, és anar element per element i aplicar el procés que haguem definit a dins seu.

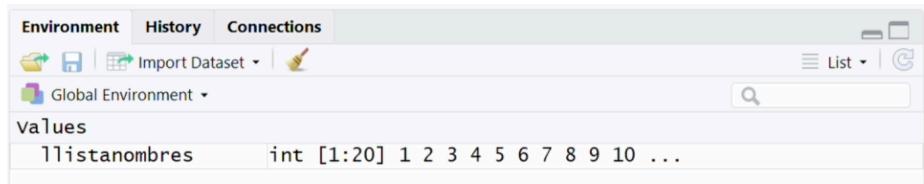
llistanombres <- 1:20

Així doncs, defineixo el bucle amb uns parèntesis.

Dins dels parèntesis, li dic un nom d'una variable, que és arbitrari. Aquí hi hagués pogut posar-li qualsevol nom que jo hagués volgut i, per cada element que trobi dins d'aquest vector o llista, anirà anomenant-lo *i* i aplicarà aquesta operació que li estic demanant. En aquest cas, li estic que m'imprimeixi cadascun d'aquests elements que trobi a la llista de nombres elevat al quadrat. Aquesta és la sintaxi per elevar un nombre al quadrat.

```
for(i in llistanombres){
  print(i ^2)
}
```

Així doncs, abans d'executar, el que interessa veure és que agafarà el primer nombre, és a dir l'1. Aquest d'aquí.



Aquest nombre el guardarà com a *i* i l'elevarà al quadrat i l'imprimirà per pantalla. Arribarà aquí i seguirà, ja que encara queden nombres a la llista per explorar. El següent nombre és el 2. Guardarà amb el nom de *i* aquest valor i l'imprimirà elevat al quadrat. Arribarà aquí *i*, com que encara quedaran nombres per explorar, agafarà el 3, el 4, etc. fins arribar al 20, per a cadascun dels elements que hagi trobat en aquesta llista. Si executem aquest vector, veiem que executa 20 vegades la instrucció *print*.

I continuem!

2.9 BUCLES (II)

Hola de nou!

Una altra manera molt clàssica de definir els loops o bucles és, en comptes d'iterar per cadascun dels elements d'una llista, com hem fet anteriorment, fer-ho per a cadascun dels índexs.

```
llistanombres <- c(7,8,9,2,4,5,6,1,7,8,9,10)
```

```
for(i in llistanombres){  
  print (i^2)  
}
```

Així doncs, podem transformar el bucle que teníem, utilitzant no els valors en si, és a dir 7, 8, 9, 2... sinó la posició que ocupen en el vector, és a dir, 1, 2, 3, 4, 5, etc. Això és tan senzill com transformar la llista sobre la qual iterem en aquesta llista de posicions, de l'1 fins a la llargada del nostre vector.

```
llistanombres <- c(7,8,9,2,4,5,6,1,7,8,9,10)
```

```
for(i in 1:length(llistanombres)){  
  print (i^2)  
}
```

Si executem aquest bucle, obtindrem els nombres de l'1 fins al 12 elevats al quadrat.

Però això no és el que ens interessa, sinó que ens interessa que el nombre que ocupa la primera posició l'elevem al quadrat. Així doncs, hem de repensar aquesta instrucció d'aquí i transformar-la.

```
print( llisanombres[i] ^2 )
```

De la llista de nombres, agafarem, a cada volta, la posició “*i*”, i acabem d'aconseguir el mateix bucle que teníem abans.

```
llisanombres <- c(7,8,9,2,4,5,6,1,7,8,9,10)
```

```
for(i in 1:length(llisanombres)){  
  print( llisanombres[i] ^2 )
```

}

Finalment, una altra estructura molt útil és el *while*, que funciona de manera bastant diferent, ja que el criteri d'aturada és més complex. Per exemple, podem definir un nombre:

```
numero<- 1
```

L'1, per exemple, i, amb aquesta funció d'aquí, que tindrà una condició d'aturada aquí dins, i les accions que vulguem fer, i després hi posarem que, mentre el número sigui més petit que 1000, mentre això es compleixi, ens imprimeixi el número i, cada vegada que ho faci, aquest número es guardi com ell mateix multiplicat per 2.

```
while( numero < 1000){
  print(numero)
  numero <- numero * 2
}
```

Així doncs, comencem amb un 1 i, a cada volta, anem doblant aquest resultat. Quan aquest resultat excedeixi el número que hem fixat aquí, és a dir el 1000, sortirem del *while* i seguiríem amb el nostre programa.

Executem i el que veiem és que ens ha fet 1, 2, 4, 8... Ens ho va imprimint, perquè li hem demanat, i ens imprimeix fins al 512, ja que tenim un 256, que multiplicat per 2 és 512. El 512 compleix aquesta propietat. Imprimeix el 512, el multiplica per 2 i el guarda. Aquest valor d'aquí ara val 1024. Com que a 1024 ja no compleix aquesta condició, no entra dins d'aquest espai d'aquí dins i surt del bucle.

Continuem!

A continuació proposem alguns exercicis sobre bucles. Trobaràs la solució de com fer-ho més endavant:

1. Explora l'ús de la instrucció *break*. Programa un bucle que recorri els números de l'1 al 100 mentre els imprimeix en pantalla. Mitjançant una condició, digues-li que executi *break* quan el número en el qual es trobi sigui el 30. Què ha passat?
2. Crea un bucle que iteri sobre un vector que hagis definit tu, que tingui tant nombres positius com negatius. Per als nombres positius, volem que digui si són parells o no. Per als negatius, que els imprimeixi en pantalla en sentit positiu, si són superiors a -10. Si el número és més petit que -10, volem que el bucle pari.
3. Crea dos números, els que vulguis, i, mitjançant un bucle *while*, ves doblant-los tots dos a cada iteració. Atura el bucle, quan el gran sigui almenys 1000 unitats més gran que el petit. Pensa detingudament en com pots definir aquesta condició d'una manera senzilla, perquè el bucle s'executi mentre no es compleixi.

2.10 RESOLUCIÓ D'EXERCICIS: BUCLES

Benvinguts i benvingudes de nou!

Vegem com es resolen els exercicis de bucles.

El primer de tots, que és el més senzill, es basa en crear una estructura de bucle `for simple`, en la qual li demanem que, quan trobi el valor `i = 30`, surti del bucle. Què passarà quan executem això? Que anirà imprimint tots els nombres de l'1 al 100 fins que trobi el 30. A partir d'aquí, sortirà del bucle. Això implica que imprimirà de l'1 fins al 29 i el 30 no arribarà a imprimir-lo, perquè sortirà en aquest moment d'aquí.

```
for(i in 1:100){
  if(i == 30){
    break
  }
  print(i)
}
```

Si ho executem, podrem veure efectivament que arribem fins al número 29.

El segon exercici és més rebuscat. El que busquem és definir un vector de nombres positius i negatius, i aplicar-los una certa acció o una certa altra, en funció del seu valor. El primer que busquem és si el nombre és positiu. En aquest cas, realitzem totes aquestes accions d'aquí dins.

```
vectordefinit<- c(1,-1,3,4,-5,20,-12,4,5)

for(i in vectordefinit){
  if(i >= 0){
    if(i %% 2 == 0){
      print("Parell")
    }else{
      print("Imparell")
    }
  }else{
    if(i<(-10)){
      break
    }
    print(-i)
  }
}
```

En funció de si és parell, utilitzant aquesta sintaxi d'aquí ($i \% 2 == 0$), o si no ho és, és a dir, en cas contrari. Aquí l'únic que fem és imprimir, si és parell, si compleix la primera condició, estant dins dels nombres positius; o, en cas contrari, imprimim que és imparell.

Si no estem en el cas de nombres positius, entrem aquesta part d'aquí al final.

```
else{
if(i<(-10)){
break
}
print(-i)
}
```

Aquí, el que fem és comprovar si el nombre és més petit que -10. Si el nombre és més petit que -10, és a dir, -11, -12, etc., sortim del bucle. En cas contrari, mostrem el nombre canviat de signe, que es pot fer senzillament afegint un menys ("–") davant del valor sobre el qual iterem. Executem el vector, el bucle, i veiem que s'executa correctament, ja que obtenim "Imparell", el nombre en positiu, "Imparell", "Parell", el nombre en positiu, "Parell"; i aquest nombre d'aquí i els subseqüents (-12,4,5) ja no apareixen, ja que hem accedit aquí (*break*) i hem trencat el bucle.

I, ja per acabar, el que busquem és crear dos nombres

```
a <- 2
b <- 1
```

i, utilitzant un bucle *while*, busquem si la diferència entre aquests nombres (jo ho he fet utilitzant la funció "absolut" (*abs*), que el que fa és que ens calcula la diferència sense importar si el més gran és el primer o el segon), si aquesta diferència és més petita que 1000.

```
while(abs(a-b) < 1000)
```

Mentre això es compleixi, és a dir, mentre la diferència entre a i b sigui més petita que 1000, anem doblant els nombres utilitzant aquestes instruccions d'aquí.

```
a <- a * 2
b <- b * 2
```

Un cop es compleixi, sortim del bucle i imprimim els resultats.

```
print(a)
print(b)
```

Ho executem

```
while(abs(a-b) < 1000){
  a <- a * 2
  b <- b * 2
}
print(a)
print(b)
```

i veiem que, en ser potències de 2, el que estem obtenint és 2048 i 1024.

Podem fer les proves amb altres valors i veuríem quan es compleix que aquesta diferència, és a dir, que la resta entre a i b sigui més gran que 1000.

L'avantatge d'utilitzar això d'aquí ($\text{abs}(a-b)$) és que, si jo poso un 1 i poso un 5, executo aquestes variables, obtinc els mateixos resultats que no hagués obtingut si hagués fet $a - b$ directament.

I això és tot, seguim!

2.11 IDEES CLAU: PROGRAMACIÓ AMB R

Ja som al final del segon mòdul. A continuació veurem les idees i els conceptes més importants que hem desenvolupat:

- Els diferents tipus d'objecte que es poden crear en R són:
 - **Variables unitàries**: utilitzades per guardar valors únics del tipus que sigui.
 - **Vectors**: conjunts de variables unitàries del mateix tipus.
 - **Dataframes**: estructures similars als fulls de càcul d'Excel.
 - **Llistes**: semblants als vectors, però més flexibles.
- Hem après com funcionen les estructures condicionals i les comparacions en R.
 - D'una banda, podem utilitzar *if* *else*, els condicionals més clàssics per definir processos diferenciats.
 - De l'altra banda, la funció compacta *ifelse*, que permet condensar aquest mateix procés binari en una única funció.
- Hem vist com utilitzar les funcions ja existents en R i quina estructura general segueixen, així com una breu introducció a la creació de funcions pròpies, la qual cosa ens permetrà reproduir processos repetitius d'anàlisi molt fàcilment.

- Finalment, hem après a dissenyar i aprofitar estructures en bucle, és a dir, processos iteratius (com el *foro* el *while*) sobre els nostres objectes (llistes, *dataframes* o vectors) i com podem especificar les seves condicions d'aturada.

3 TREBALLAR AMB BASES DE DADES

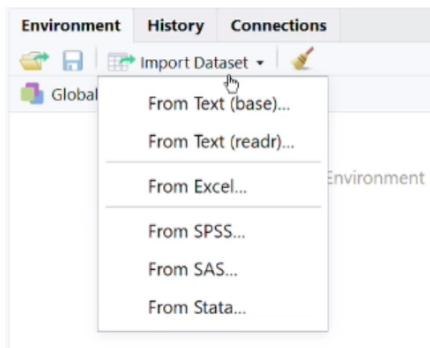
En aquest mòdul veurem com utilitzar l'importador de bases de dades de RStudio per a diferents formats, i com podem guardar els objectes que haguem creat, com per exemple, les bases de dades en formats propis d'R o Excel. També veurem els principals tipus de filtre i transformació que podem aplicar als nostres *dataframes*, per obtenir aquelles dades que ens interessen en el format desitjat.

3.1 IMPORTAR I EXPORTAR BASES DE DADES

Hola de nou!

A continuació, veurem com utilitzar l'importador de bases de dades d'RStudio, que simplifica enormement les tasques d'importació en el format correcte, i com podem guardar els objectes amb els quals treballem, ja sigui en un format llegible per un altre programa com pot ser Excel o OpenOffice, o en el format nadiu d'R.

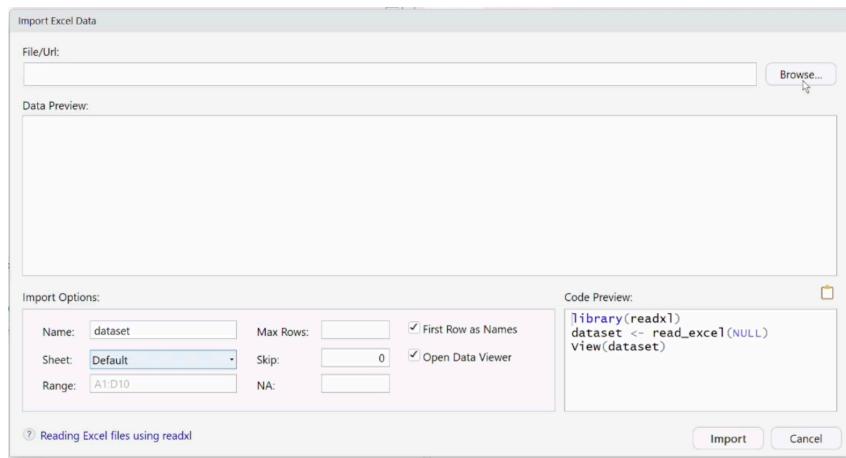
L'importador de bases de dades es troba a l'entorn. Hem de fer clic aquí i tenim diverses opcions.



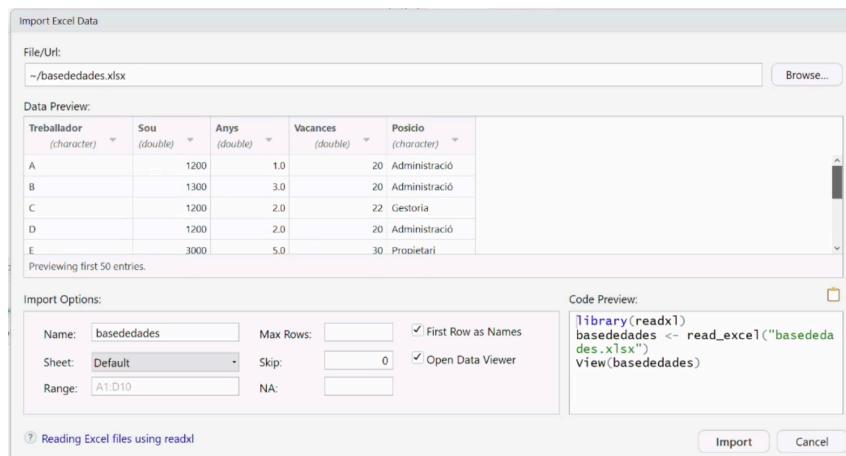
Aquests tres d'aquí són programari estadístic.

Tenim Excel i dos importadors de text. El meu preferit és aquest segon (*From Text (readr)*). Nosaltres el que farem és utilitzar l'importador d'Excel.

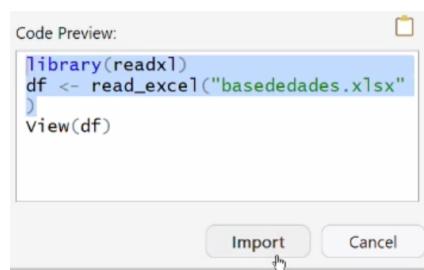
Haurem de seleccionar el fitxer que ens interessa, clicant aquí.



Aquí, ja hi tinc uns quants fitxers. Agafaré el primer i ens mostrerà, en aquest quadre d'aquí, una primera visualització.



Puc modificar el seu nom, per exemple, posar-li la notació estàndard *df*; seleccionar el nombre màxim de files que vull importar, i aquí em dona un codi molt útil que podré copiar i enganxar, per després estalviar-me fer aquest procés.



Se'ns obre una visualització de la base de dades, s'ha executat el codi automàticament i ja podem anar al nostre script i enganxar el codi que acabem de copiar.

Guardar objectes

Si el que volem és guardar un objecte (normalment un dataframe, malgrat no sempre sigui així) per usar-lo en R posteriorment, podem utilitzar la següent instrucció.

```
save(df,file="dades.Robj")
```

On hi haurem d'especificar l'objecte que volem guardar, en aquest cas el nostre dataframe, i on volem que el guardi.

El format *Robject* és un format nadiu d'R, que permet la importació i l'exportació de manera molt senzilla. Per veure com funciona, l'executem. Se'n guarda on tinguem especificat el nostre directori de treball. Esborrem l'objecte que nosaltres teníem i, ara, utilitzant la funció *load* i el nom del fitxer amb el qual estiguem treballant, tornem a recuperar l'objecte que teníem inicialment.

```
load("dades.Robj")
```

Si el que volem és exportar-ho en format .csv, que és perfectament llegible amb Excel, senzillament haurem d'utilitzar la funció *write_csv*, que forma part del paquet *readr*.

```
require(readr)
```

Carreguem aquest paquet i, utilitzant aquesta funció, on li haurem d'especificar l'objecte que vulguem guardar una altra vegada, i on ho volem guardar.

```
write_csv(df,path = "df.csv")
```

Estem guardant aquest fitxer en un fitxer llegible amb processadors de càlcul com Excel.

3.2 FILTRAR BASES DE DADES

Hola a tots i a totes!

En aquest vídeo veurem com podem tractar els nostres dataframes de manera que obtinguem només aquelles dades que desitgem, ja sigui des d'un punt de vista purament estètic, com pot ser ordenant-les o visualitzant-ne una petita part, o aplicant filtres als valors de les nostres variables.

Una primera funció que veurem és la funció *head*.

```
head(df)
```

Aquesta funció permet seleccionar només les primeres files de la nostra base de dades. De la mateixa manera, podem utilitzar la funció *tail*, per veure'n les últimes.

```
tail(df)
```

Si el que volem és seleccionar només les files que compleixin una certa propietat, per exemple, els treballadors i treballadores amb més de dos anys a l'empresa, podríem fer això: aplicar un filtre on volem que totes les files compleixin una propietat d'una de les columnes; que el valor d'aquesta columna sigui més gran o igual a dos anys.

```
df[, df$Anys >= 2,]
```

Així doncs, el que obtindrem és tota la nostra base de dades, on aquesta columna tingui un valor 2 o superior. Podríem guardar aquest objecte amb el nom “treballantic”.

```
treballantic <- df[, df$Anys >= 2,]
```

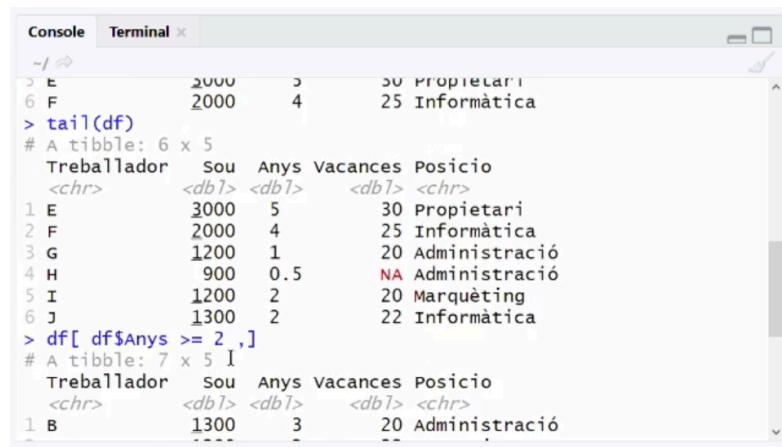
Aquí veiem que tres de les observacions que teníem, ara ja no disposem d'elles.

Els filters que podem aplicar a una base de dades són de tot tipus. Per exemple, el que podrírem fer és buscar únicament el treballador amb un cert nom, per exemple, anem a buscar el treballador “F”.

Ho podríem fer utilitzant la base de dades original, seleccionant la fila que compleixi que el nom del treballador, que és una columna (estic utilitzant el tabulador per autocompletar, quan em surt l'ajuda en posar el dòlar), sigui igual a exactament el valor que apareix aquí. Una bona pràctica és copiar-ho exactament igual.

```
df[df$Treballador == "F",]
```

Obtinc la informació, és a dir, totes les columnes, d'aquest treballador. Si us hi heu fixat, en aquesta base de dades, tinc una dada faltant, que apareix ressaltada aquí.



```

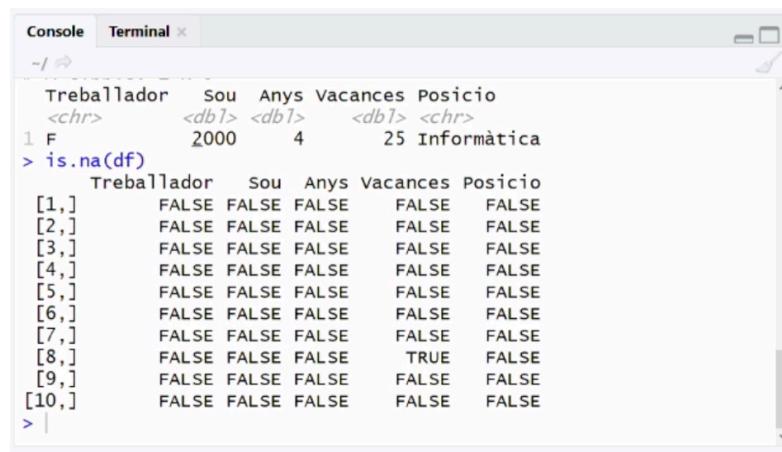
Console Terminal ×
~/🔗
> E      2000  3    30 Propietari
6 F      2000  4    25 Informàtica
> tail(df)
# A tibble: 6 x 5
  Treballador   Sou  Anys Vacances Posicio
  <chr>        <dbl> <dbl>    <dbl> <chr>
1 E            3000  5       30 Propietari
2 F            2000  4       25 Informàtica
3 G            1200  1       20 Administració
4 H             900  0.5     NA Administració
5 I            1200  2       20 Marquèting
6 J            1300  2       22 Informàtica
> df[ df$Anys >= 2 , ]
# A tibble: 7 x 5
  Treballador   Sou  Anys Vacances Posicio
  <chr>        <dbl> <dbl>    <dbl> <chr>
1 B            1300  3       20 Administració

```

Puc detectar-la, utilitzant aquesta funció.

`is.na(df)`

Si l'aplico, sobretot a la base de dades, el que obtinc és això d'aquí, una matriu de falsos i veritats, quan troba una dada faltant.



```

Console Terminal ×
~/🔗
> E      2000  3    30 Propietari
6 F      2000  4    25 Informàtica
> is.na(df)
  Treballador   Sou  Anys Vacances Posicio
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE
[4,] FALSE FALSE FALSE FALSE FALSE
[5,] FALSE FALSE FALSE FALSE FALSE
[6,] FALSE FALSE FALSE FALSE FALSE
[7,] FALSE FALSE FALSE FALSE FALSE
[8,] FALSE FALSE FALSE TRUE FALSE
[9,] FALSE FALSE FALSE FALSE FALSE
[10,] FALSE FALSE FALSE FALSE FALSE
> |

```

T'animo a què exploris com utilitzar aquest tipus d'estructura per eliminar les files que tinguin valors faltants.

Una altra cosa que podem fer és aplicar filtres múltiples. Per exemple, podríem filtrar els treballadors i treballadores, segons el seu sou, més gran a 1200, per exemple, i el nombre de dies de vacances, superior a 20. Si no afegim aquesta coma d'aquí, ens donaria un error.

`df[,df$Sou > 1200 & df$Vacances > 20,]`

Per acabar aquest vídeo, anem a veure com podem ordenar una base de dades. El que farem és utilitzar la funció `order` i li hem d'especificar una columna de la nostra base de dades que sigui numèrica, per exemple el sou.

```
df[order(df$Sou),]
```

Si executem aquesta instrucció, obtenim el sou ordenat de menor a major. Això ho podem corregir, utilitzant un dels paràmetres de la funció. Volem que sigui decreixent? Sí. Li especificuem que volem que l'ordre sigui decreixent.

```
df[order(df$Sou,decreasing = TRUE),]
```

Executem i ja tenim els nostres treballadors ordenats en funció del seu salari.

Seguim!

A continuació proposem alguns exercicis sobre com filtrar bases de dades. Trobarem la solució de com fer-ho en aquest vídeo:

1. Com utilitzaries les funcions `order()` i `tail()` per obtenir únicament el nom de la persona treballadora amb un sou més alt? Prova d'aconseguir tota la seva informació amb la funció `max()`.
2. Com obtindries tots i totes les treballadores amb un sou d'entre 1.250 i 2.500 €?
3. Com trobaries si la persona treballadora amb un sou més baix també és qui té menys vacances?

3.3 RESOLUCIÓ DE L'EXERCICI PER FILTRAR BASES DE DADES

Hola de nou!

Anem a veure com podem resoldre aquests exercicis.

El primer que demanava era, utilitzant la funció `tail`, obtenir el treballador o treballadora amb un sou més elevat. Per començar, el que hem de fer és ordenar la base de dades.

```
tail(df[order(df$Sou),])
```

El que estem fent és aplicar un filtre que el que ens dona és tota la base de dades amb el sou ordenat de manera creixent. Utilitzant la funció `tail`, el que obtindríem són els últims registres d'aquesta taula. Podem especificar-li que només volem l'últim de tots, amb aquest número d'aquí.

```
tail(df[order(df$Sou),],1)
```

Si executem això, obtenim efectivament l'últim registre de tots, és a dir, el propietari.

Una manera alternativa de fer-ho és utilitzant aquest filtre per files, és a dir, seleccionant totes les files on el sou sigui exactament igual al màxim del sou.

```
df[df$Sou == max(df$Sou),]
```

Aquesta sintaxi d'aquí, podem simplificar-la utilitzant aquesta funció d'R, que bàsicament ens dona la posició on es troba el màxim.

```
df[which.max(df$Sou),]
```

Mirem-ho i, si ho executem, aquest valor d'aquí no és el sou màxim, evidentment, sinó que és, en l'ordenació original, quina fila conté el sou màxim. Si executem, veiem que obtenim el mateix resultat que al principi.

El segon exercici que proposo és buscar tots aquells treballadors i treballadores que compleixin que el seu sou es trobi entre 1.250 i 2.500 euros. El que hem d'aplicar aquí, doncs, és un filtre múltiple. Hi ha altre programari on això es pot fer més directament. R també contempla algunes opcions per fer-ho amb un interval directament, tot i que, per començar, és recomanable utilitzar aquestes condicions múltiples.

```
df[df$Sou > 1250 & df$Sou < 2500,]
```

Si executo aquesta part d'aquí (`df$Sou > 1250 & df$Sou < 2500`), el que veiem és que ens dona un conjunt de falsos i veritats en funció de si cadascun dels treballadors i treballadores compleixen aquestes propietats.

Si executo, obtinc tota la informació de la base de dades que compleix les dues propietats simultàniament, és a dir, si estem dins d'aquest interval.

Si volguéssim seleccionar algunes columnes en concret, podríem fer-ho de la següent manera.

```
df[df$Sou > 1250 & df$Sou < 2500, c("Posicio", "Treballador")]
```

Així doncs, obtindríem els mateixos tres registres però ara només de la informació que ens interessés, per exemple: la posició i el seu nom.

I, per acabar, l'últim exercici, demanava si el treballador o treballadora amb un menor sou coincidia amb el treballador o treballadora amb menys dies de vacances. Aquí, la meva proposta és utilitzar la funció `which.min`, que bàsicament troba la posició que ocupa la persona amb el sou menor.

```
df[which.min(df$Sou),]$Treballador
```

I fer el mateix amb la persona que ocupa l'última posició en nombre de dies de vacances, i seleccionant la columna rellevant utilitzant la sintaxi típica per als dataframes, és a dir, el \$ (dòlar).

```
df[which.min(df$Vacances),]$Treballador
```

Aquí, el que estem fent és obtenir, per una banda, el nom del treballador o treballadora; per altra banda, el segon nom, que ja veiem que no coincideix.

```
df[which.min(df$Sou),]$Treballador == df[which.min(df$Vacances),]$Treballador
```

I, si executem les dues instruccions conjuntament, el que obtenim és si coincideixen o no. En aquest cas, no coincideixen.

Una manera més ràpida de fer això és senzillament utilitzant les funcions *which.min*. Ja que, si la posició no coincideix, és que no són el mateix treballador.

```
which.min(df$Sou) == which.min(df$Vacances)
```

I fins aquí el vídeo. Ens veiem a la propera!

3.4 TRANSFORMAR BASES DE DADES

Hola!

La transformació d'una base de dades pot anar des de l'ordenació de les seves columnes, com ja sabem, fins a la creació de noves files o columnes. Aquí ens centrarem en aquests aspectes. Quan decidim afegir nova informació a les bases de dades que ja teníem, una cosa que hem de tenir sempre present és les dimensions de l'objecte que tenim. Si volem afegir una nova fila, per exemple, un treballador o treballadora, les noves dades que hi incorporem hauran d'estar exactament en el mateix format original, o podríem obtenir errors o dades errònies. El mateix passa amb les columnes noves, hem de vigilar sempre que tinguin la mateixa longitud que les prèvies. Una bona manera de procedir és generant-les a partir de les ja existents!

Anem a veure'n un exemple.

El primer que farem és seleccionar un sol treballador de la nostra base de dades. El guardarem amb aquest nom.

```
noutreballador <- df[]
```

I, utilitzant un filtre per files, seleccionarem el treballador o treballadora número o nom “c”. Acabem de crear un objecte de tipus dataframe amb una observació i 5 columnes.

```
noutreballador <- df[df$Treballador == "C"]
```

Ara el que podem fer és modificar les característiques d'aquest objecte. Per exemple, li canviarem el nom i li direm treballador “k”.

```
noutreballador$Treballador <- "K"
```

I, per exemple, també li podem canviar el nombre de dies de vacances que té assignats. I n'hi assignarem 20.

```
noutreballador$Vacances <- 20
```

Si ara mirem el nou treballador, veiem que té el nom que li hem donat i el nombre de dies de vacances que tenia. El que podem fer és afegir aquest nou treballador, que té les dimensions que ens interessen a la base de dades original que teníem.

Això ho podem fer, utilitzant aquesta instrucció d'aquí: raw bind (*rbind*). Estem afegint files noves a la base de dades. Posem: el nom de la base de dades i la nova fila.

```
rbind(df,noutreballador)
```

Aquesta instrucció evidentment funciona també si volem afegir moltes files. Estem obtenint la concatenació d'aquesta fila.

Si ara mirem l'objecte *df*, veurem que les dades no s'han afegit.

```
rbind(df,noutreballador)  
df
```

No s'han afegit perquè aquesta instrucció d'aquí no s'afegeix directament, sinó que el que hem de fer és guardar-ho.

Ara hem sobreescrit el nostre objecte *df*, afegint-li aquesta nova fila. Podem comprovar que ara sí que ha funcionat.

Si el que volem, per altra banda, és afegir noves columnes, ho podem fer de dues maneres. Vegem primer la més rudimentària. Podem utilitzar una instrucció molt similar a aquesta, serà colum bind (cbind) i un vector de la mida correcta.

El que farem és crear una nova variable i li direm “sexe”, que haurà de tenir les dimensions de la nostra base de dades, és a dir, 11. Ho faré de manera més o menys automàtica. Li diré que em repeteixi 5 homes i 6 dones. Aquestes instruccions d'aquí el que em creen són un vector de 5 H i aquestes d'aquí, un vector de 6 D.

```
sexe <- c(rep("H",5),rep("D",6))
```

Si executem, el que acabem d'obtenir és un vector de caràcters amb 5 homes i 6 dones. Ara, aquest objecte d'aquí té les mateixes dimensions que el nombre de files que té la nostra base de dades. Així que el que podem fer és crear una nova columna utilitzant el mateix que acabàvem de veure.

```
df<- cbind(df,sexe)
```

Si ara mirem la nostra base de dades, veurem que tenim la nova columna que li acabem de demanar.

Per altra banda, i això és el més interessant, podem crear una columna a partir d'una o més columnes. Per exemple, si paguem les vacances a 50 € el dia + 5 € per cada any d'antiguitat, podríem fer l'operació següent.

```
df$CostVacances <- df$Vacances * ( 50 + 5*df$Anys)
```

Creo una nova columna i li diré “cost de les vacances” i diré que és el nombre de dies de vacances, que està guardat en aquesta columna, multiplicat per 50, el cost base, més 5 euros addicionals per cada any d'antiguitat. Si executo aquesta instrucció, no apareix res.

Però, si vaig a veure què ha passat amb la meva base de dades, obtinc una nova columna, que és “cost de les vacances”, que, ha aplicat exactament aquest càlcul d'aquí.

```
df$CostVacances <- df$Vacances * ( 50 + 5*df$Anys)
df
```

Tinc una dada faltant, que no és res més que producte de què estic fent una operació amb una dada que no existeix. I, per tant, no m'ho pot calcular.

Seguim!

A continuació proposem alguns exercicis sobre com transformar bases de dades. Trobarem la solució de com fer-ho en aquest vídeo:

1. Inventa una columna que doni valors numèrics al rendiment dels treballadors i treballadores. Posteriorment, crea una segona columna on es descrigui el rendiment en relació amb el seu sou.
2. Descobreix com podem esborrar files d'un dataframe. Hauràs d'utilitzar el signe “-”. Funciona en el cas de les columnes?
3. Explora com funciona la instrucció *table()* sobre una columna qualitativa del dataframe.

3.5 RESOLUCIÓ DE L'EXERCICI PER TRANSFORMAR BASES DE DADES

Benvingudes i benvinguts de nou!

A continuació resoldrem els tres exercicis proposats.

El primer que fem és crear una nova columna a la nostra base de dades que es basa en el rendiment dels nostres treballadors i treballadores. Així doncs, ho fem amb aquesta instrucció i el que podem veure és que efectivament hem afegit una nova columna a la base de dades.

```
df$Rendiment <- c(7,8,9,10,7,6,8,9,5,9)
```

A partir d'aquesta, crearem una nova columna que anomenarem “PreuRendiment”, que bàsicament serà el rendiment que estem suposant que tenen els nostres treballadors i treballadores, dividit pel sou, per veure quins són els treballadors i treballadores que més rendeixen en funció del que guanyen.

```
df$PreuRendiment <- df$Rendiment / df$Sou
```

Executem aquesta instrucció i, si explorem l'objecte, veiem que no és especialment llegible.

Així doncs, el que jo proposo és multiplicar aquest resultat per 1000, per exemple, i sobreescriure aquesta columna (*df\$PreuRendiment*) i veure què queda.

```
df$PreuRendiment <- df$Rendiment / df$Sou * 1000
```

Queda una variable bastant més llegible, però encara ho podem millorar, utilitzant la instrucció *round*, que arrodoneix els nostres nombres, per exemple a dos decimals.

```
df$PreuRendiment <- round(df$Rendiment / df$Sou * 1000, 2)
```

Si ara mirem els resultats, efectivament les puntuacions són més atractives.

A continuació, proposava utilitzar el signe menys (-) per esborrar columnes i files de la nostra base de dades. Esborrar una fila és tan senzill com utilitzar aquesta instrucció d'aquí.

```
df <- df[-5,]
df$PreuRendiment <- NULL
```

Si l'executem, el que acabem de fer és esborrar la fila número 5 de la nostra base de dades. Veiem que ara només arriba fins al 9. Si volem utilitzar aquesta mateixa estructura per a les columnes, també ho podem fer. Ja que és tan senzill com esborrar, per exemple, el nom.

```
df <- df[-5,]
df <- df[,-1]
df$PreuRendiment <- NULL
```

Tot i així, una millor pràctica és esborrar les columnes pel seu nom. El que mostra aquí, per exemple, és com esborrar la columna nova que acabàvem de crear. Si l'executem, en aquesta línia d'aquí li estem dient que la columna preu-rendiment l'esborri, la faci desapareixer.

```
df <- df[-5,]
df <- df[,-1]
df$PreuRendiment <- NULL
df
```

Hem d'anar en compte ja que el que estem fent és esborrar i guardar l'objecte amb el nom original. Podria ser interessant que els objectes en els quals els esborrem una columna o unes files, els guardessim amb un nom diferent.

I ja per acabar proposava que exploressis la funció *table* per veure un resum qualitatiu d'algunes de les columnes de la nostra base de dades. Per exemple, la posició.

```
table(df$Posicio)
```

Aquesta és una funció molt útil que el que ens permet és generar un resum de quants elements troba de cada categoria. Això d'aquí que podrem guardar, per exemple, en un objecte anomenat taula, serà molt útil per fer diagrames de barres i diagrames de sectors.

```
taula<- table(df$Posicio)
```

Seguim!

3.6 IDEES CLAU: TREBALLAR AMB BASES DE DADES

Vegem els conceptes més importants del mòdul, que tenen a veure amb la realització de diversos processos amb les nostres pròpies bases de dades:

- Importar les dades a R, en diferents formats, mitjançant l'assistent d'importació d'RStudio.
- Guardar els nostres objectes, ja sigui en el format nadiu d'R o formats llegibles per a Excel.
- Obtenir dades dels nostres *dataframes* mitjançant filters simples i múltiples, ja sigui per columnes o files.
- Modificar les dimensions de la nostra base de dades, afegint-li o esborrant-li dades, així com crear noves variables a partir de la informació de la qual disposavem al principi.

4 VISUALITZACIÓ DE DADES

En aquest darrer mòdul veurem com realitzar alguns dels gràfics més populars amb R, i com podem personalitzar-los i millorar-los per fer-los més atractius. Atès que la comunicació de resultats en format gràfic és una de les competències més necessàries actualment, posarem especial èmfasi en la interpretació dels resultats. Els gràfics que explorarem en aquest mòdul són el diagrama de sectors, el gràfic de barres, l'histograma, el diagrama de caixa i el núvol de punts, així com algunes eines per afegir-hi informació addicional.

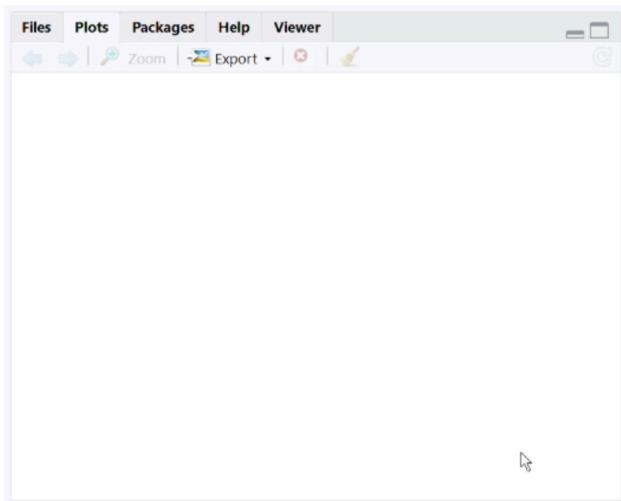
4.1 INTRODUCCIÓ A LA VISUALITZACIÓ DE DADES

Hola a totes i a tots!

En aquest vídeo explorarem les bases de la visualització de dades en R, com podem aprofitar RStudio per a tal finalitat i algunes consideracions prèvies a l'hora de treballar amb visualitzacions.

Primer de tot, hem d'entendre l'espai que ens ofereix RStudio per a la visualització:

Els gràfics a RStudio apareixen a la pestanya *Plots*, en aquest espai d'aquí.

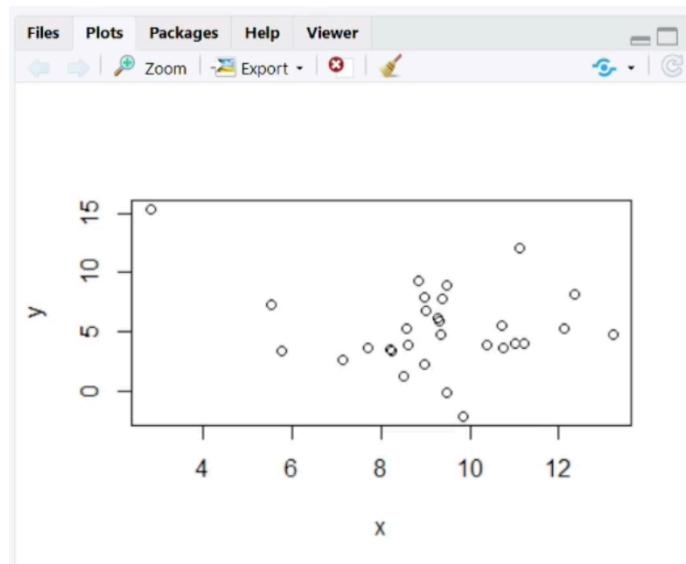


És important que sempre el tinguem amb una mida suficientment gran perquè puguem encabir-hi un gràfic. Si tinc la pestanya en una dimensió similar a aquesta, quan executem una funció gràfica, per exemple `plot`, R es queixarà. Així doncs, la mantenim més o menys maximitzada.

Aquí, he generat dos vectors de dades aleatòries, és a dir 30 punts, de mitjana 10 i desviació 2, d'una variable aleatòria normal. I, utilitzant una instrucció `plot x, y`, obtinc un núvol de punts per x i per y.

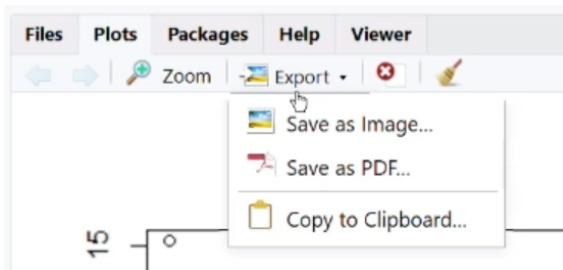
```
x <- rnorm(30,10,2)
y <- rnorm(30,5,3)
```

```
plot(x,y)
```



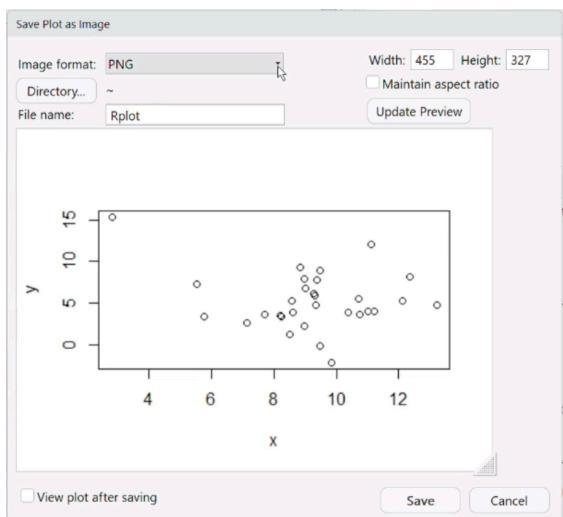
És important entendre que quan estem generant un gràfic, estem com creant una fulla de paper, i tot el que vulguem afegir-hi, llegendes, punts, línies... ho estem afegint a sobre. I això implicarà que podem haver-hi superposicions.

En segon lloc, hem de remarcar la importància de l'exportació.



Els gràfics exporten utilitzant aquesta pestanya d'aquí.

Podem copiar-lo directament utilitzant aquesta instrucció, però el més habitual és guardar-ho com una imatge. Aquí, podem escollir el format que vulguem.



Els més habituals són els dos primers i el directori on ho volem guardar, així com el nom. Podem fixar l'amplada i l'altura de la imatge, i és important considerar que, com l'haguem generat, influirà en com l'estem guardant.

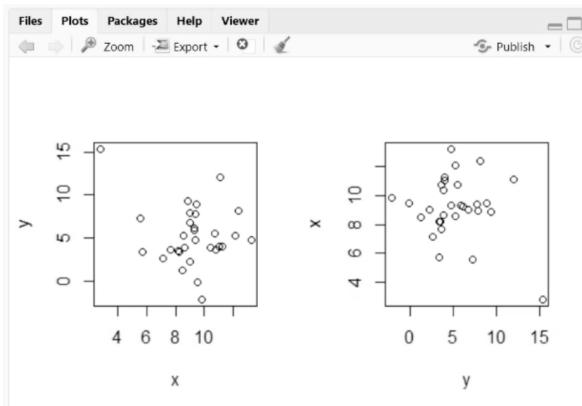
En tercer lloc, mostrarem com ajuntar dos gràfics en una sola visualització i la importància que té la seva disposició en la comunicació d'informació.

Així doncs, utilitzant la instrucció *par* i aquest paràmetre d'aquí, *mfrow*, estem aconseguint partir el nostre espai de gràfics en una fila, dues columnes.

```
par(mfrow=c(1,2))
```

Per exemple, si ara executem aquest gràfic d'aquí, ens el col·loca en la primera posició, deixant-nos espai per un altre gràfic. Per exemple, podríem imprimir el gràfic invers, és a dir, y amb x, i obtindríem aquesta disposició d'aquí.

```
plot(x,y)
```



```
plot(y,x)
```

La comparació que podríem fer seria bastant pràctica. Per altra banda, si el que fem és crear un espai on tenim dues files i una columna.

```
par(mfrow=c(2,1))
```

L'executem i ara tornem a executar aquests dos gràfics d'aquí. Es queixa de què l'espai és massa petit, perquè els dos gràfics no caben gaire bé en vertical. Ho amplifiquem una mica més, executem i veiem que la visualització no és gens atractiva. Així doncs, és molt important que, a l'hora de partir els gràfics en dos espais, pensem, abans de fer-ho, quina és la disposició que comunicarà millor la nostra informació.

És important destacar que en aquest mòdul veurem com generar i modificar gràfics utilitzant les funcions bàsiques d'R, per la seva senzillesa i potencial. Si ens interessen visualitzacions més avançades, atractives o interactives, hauríem de considerar paquets com “ggplot2” o “Plotly”.

Així que ja sabem una mica més d'RStudio.

Abans d'acabar aquest vídeo, per recuperar la visualització en un sol gràfic, el que necessitem és tornar a executar aquesta instrucció dient-li que volem una sola fila i una sola columna.

```
par(mfrow=c(1,1))
```

4.2 GRÀFICS DE BARRES I DIAGRAMES DE SECTORS

Hola de nou!

A continuació, veurem uns exemples de codi molt senzills de com podem fer dues de les visualitzacions més populars amb R.

El codi, com veuràs, per a cadascuna de les visualitzacions és molt simple, però les seves opcions de personalització són molt interessants. Hem de destacar que a l'hora de crear un gràfic o visualització, el més important sempre és l'assaig i l'error, ja que molts dels paràmetres que podrem modificar no tenen una fórmula que ens digui quin és un bon valor en tots els escenaris, sinó que hem d'explorar quin és el més adequat per a cadascuna de les circumstàncies en les quals ens trobem.

Comencem creant una taula. Crearem aquest objecte, utilitzant la funció *table*, d'una de les columnes categòriques de la nostra base de dades. Per exemple, la posició.

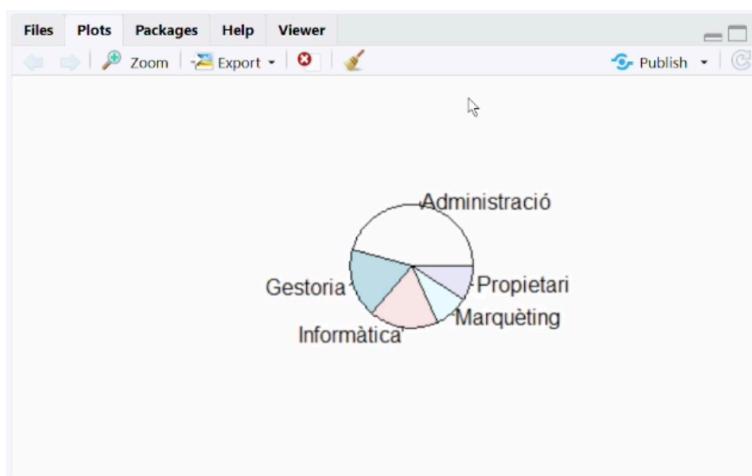
```
taula <- table(df$Posicio)
```

Executem. Se'ns crea aquí un objecte de tipus taula i anem a veure què conté.

Conté un recompte de les posicions que tenim per a cada un dels individus. Un cop tenim aquest tipus d'objecte, crear un diagrama circular, un diagrama de formatget, és tan senzill com utilitzar la funció *pie*.

```
pie(taula)
```

Aquesta és la visualització per defecte, però modificar-ho, com ja he comentat, té molt potencial.



El primer que podem fer és afegir-li un títol, utilitzant la funció *main*.

```
pie(taula, main="Titol")
```

Una altra propietat que podem modificar és aquesta d'aquí, que el que permetrà, com ara veurem, és que comenci exactament a les 12 del migdia.

```
pie(taula, main="Titol", clockwise = TRUE)
```

Amb això, el que aconseguim és que sigui més interpretable aquesta primera categoria. Ja que percebem millor les àrees, començant exactament des del centre.

També podem modificar els colors, per exemple, utilitzant una funció molt interessant, que és la funció *rainbow* i li hem d'especificar la llargada de la taula.

```
pie(taula, main="Titol", clockwise = TRUE, col = rainbow(length(taula)))
```

Si executem només aquesta part d'aquí, podem veure que el que obtenim són 5 colors diferents en la nomenclatura tècnica. Si executem tot el gràfic, obtenim la mateixa estructura però amb colors de l'arc de Sant Martí. Una última modificació seria modificar el radi del nostre gràfic. Aquí sí que haurem de fer assaig i error, i podem provar amb radi 1. Radi 1 és més gran, podríem provar amb radi 5, per exemple; i veure que ens hem passat.

```
pie(taula, main="Titol", clockwise = TRUE, col = rainbow(length(taula)), radius=1)
```

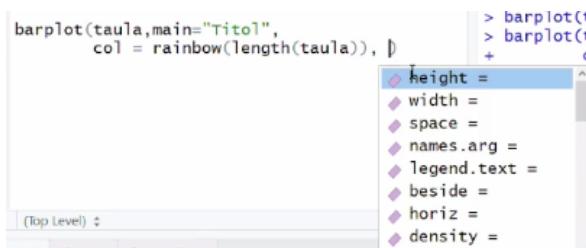
Això, com ja he comentat, s'ha d'anar ajustant en funció de la mida que tinguem de la nostra pestanya.

Per acabar aquest vídeo, el que farem és un diagrama de barres. La funció és *barplot* i podem utilitzar una altra vegada el mateix objecte "*taula*".

```
barplot(taula)
```

Si ho executem, veiem aquesta visualització d'aquí. Sembla que no hagi funcionat correctament. L'únic que passa és que no té espai suficient per mostrar-nos totes les etiquetes correctament. I, si desplaçem el nostre espai, podem visualitzar-les. Alguns dels paràmetres que podem modificar són, per exemple, *main*. Li afegeix un títol. També podem afegir-li colors, de manera idèntica a com hem fet en l'exemple anterior. Copiaré directament.

```
barplot(taula, main="Títol", col = rainbow(length(taula)))
```



I podríem comprovar quins paràmetres podem moure, mirant tota aquesta llista d'aquí.

T'animo a què ho exploris i miris què és el que necessites per millorar o adequar el teu gràfic a un context concret.

Seguim amb el curs!

4.3 HISTOGRAMES I DIAGRAMES DE CAIXA

Hola!

En aquest vídeo veurem com es generen els dos tipus de gràfics més estesos a l'hora d'estudiar la distribució i dispersió de les nostres dades. Els diagrames de caixa o boxplots i els histogrames. Posarem un èmfasi especial en la seva interpretació correcta.

Un boxplot serveix fonamentalment per descriure els quartils i els outliers d'una variable numèrica i, sobretot, per comparar diverses categories que comparteixin una mateixa escala.

El que podem fer, per exemple, és carregar la nostra base de dades *mtcars* i fer un *boxplot* d'una de les columnes d'aquesta base de dades. Per exemple, aquesta d'aquí.

```
data(mtcars)
boxplot(mtcars$wt)
```

Si executem, podem veure un diagrama de caixa molt senzill, però que ja conté tota la informació que ens interessa. Un diagrama de caixa ens mostra, en primer lloc, la mediana, aquesta recta d'aquí, que separa entre el 50 % inferior dels cotxes i el 50 % superior dels cotxes. La caixa inclou des del 75 % superior dels cotxes, és a dir, el tercer quartil que va avall i el 25 % dels cotxes cap avall, que separa aquesta línia d'aquí, que és el primer quartil.

Aquest valor d'aquí és el mínim; i l'últim punt que trobem és el màxim. Què vol dir que apareguin en aquest format? Que estan prou lluny de la caixa com per considerar-les dades anòmals, és a dir, outliers. Aquests outliers en aquesta base de dades apareixen per sobre, però també podrien aparèixer per sota.

Així repassant, podem veure els extrems de les nostres dades, la mediana, el tercer quartil i el primer quartil.

Com he comentat, una de les utilitats principals dels boxplots és comparar dos grups. El que podem fer, per exemple, és comparar dues vegades aquesta variable.

```
boxplot(mtcars$wt, mtcars$wt)
```

Si ho executéssim, veuríem dues vegades el mateix boxplot, però que compleixin una propietat en concret. Per exemple, que la variable *vs* sigui igual a 1, i que, per al segon grup, sigui igual a 0.

```
boxplot(mtcars$wt[mtcars$vs==1],mtcars$wt[mtcars$vs==0])
```

Ara executem i podem veure com es distribueix la variable *wt*, en funció de la categoria 1 o la categoria 0. Un dels paràmetres que podem modificar, en aquest cas, és el color. Li haurem d'especificar dos colors, per exemple, verd i vermell.

```
col = c("green", "red")
```

I podríem especificar-li també, aquest és un paràmetre molt important, els noms. En aquest cas li especificaríem *grup 1* i *grup 0*.

```
names = c("Grup 1","Grup 0")
```

Executem i veiem que hem aconseguit el grup 1, el grup 0, per ordre d'aparició, i els colors en el mateix ordre en el qual els hem anat introduint.

I, si fem referència als histogrames, aquests ens permeten visualitzar, amb molt més detall i d'una manera més intuïtiva, com es distribueixen les dades.

Així doncs, la funció que utilitzem és *hist*. Utilitzem la mateixa variable.

```
hist(mtcars$wt)
```

I el que podem veure és el perfil de com es distribueix aquesta variable d'aquí, *wt*, i quantes vegades apareix, és a dir, la seva freqüència, per a cadascun dels valors de la variable.

Una manera molt interessant de millorar aquest gràfic és afegint-li nous talls. Per exemple, podem afegir-n'hi 15.

```
hist(mtcars$wt, breaks=15)
```

La qual cosa permet comprovar que tenim una categoria molt més poblada que les altres i veure un perfil molt més detallat de com es distribueix. Si el que ens interessa és una freqüència relativa, el que podríem fer és dir-li la freqüència *FALSE*.

```
hist(mtcars$wt, breaks=15, freq=FALSE)
```

Una altra opció que pot ser interessant és dir-li *plot FALSE*.

```
hist(mtcars$wt, breaks=15, freq=FALSE, plot=FALSE)
```

Amb la qual cosa estem obtenint *warning*, que no ens ha de preocupar perquè no és un error. Estaríem obtenint tots els càlculs que ha realitzat i totes les dades numèriques que s'estreuen d'un histograma, que això pot ser especialment interessant, si el que volem veure és quins són aquests talls i quina és la seva densitat.

4.4 NÚVOL DE PUNTS I RECTA DE REGRESIÓ (I)

Hola!

Ha arribat el moment d'introduir un dels models fonamentals de l'estadística i l'anàlisi de dades: i aquest és el model de regressió lineal, també anomenat “recta de regressió”. Per fer-ho, utilitzarem la funció *plot*, la més bàsica de totes, per dibuixar el núvol de punts en el format que desitgem, i veurem com afegir-li, de manera gràfica, un model a sobre.

La funció *plot* requereix dues coordenades: la coordenada x i la coordenada y.

```
plot(x,y)
```

Utilitzarem dues columnes de la nostra base de dades: *mtcars*, *wt* i *mtcars mpg*.

```
plot(x=mtcars$wt, y=mtcars$mpg)
```

Amb aquesta instrucció, podem veure directament un núvol de punts molt senzill. El que veurem ara és com millorar-lo, acolorir aquests punts basant-nos en alguna característica i com afegir-li la recta de regressió.

El primer que farem és crear un filtre. Un filtre que decidirà si aquests cotxes d'aquí pertanyen a un grup o a un altre. Utilitzarem la columna *vs* i agafarem els valors que siguin igual a 0 per a una categoria i igual a 1 per a una altra.

```
filtrat <- mtcars$vs == 0
```

Obtenim aquest tipus de filtre, que utilitzarem per seleccionar quins casos pintem d'un color i quins casos pintem d'un altre. Així doncs, el que voldrem és afegir-li uns punts a sobre d'aquest gràfic. Aquests punts els podem afegir amb la funció *points* i el que farem és dir-li "agafa aquests valors d'aquí, però ara pinta'm d'un color especial els que tinguin un valor *TRUE* del filtre que acabem de crear".

```
points(x=mtcars$wt[filtre],y=mtcars$mpg[filtre])
```

Hi ha moltes maneres de fer això que estic fent, però aquesta és una manera molt interessant perquè permet aprendre a crear filters i manipular-los posteriorment. Aquests punts d'aquí els voldrem d'un altre color, per exemple, de color vermell.

```
points(x=mtcars$wt[filtre],y=mtcars$mpg[filtre], col="red")
```

Així doncs, el que hem fet és superposar punts de color vermell als punts que teníem abans.

Afegir-li la recta de regressió serà tan senzill com definir un model de regressió lineal. La funció és "linear model" (*lm*) i hem de definir la meva variable de l'eix vertical, una titlla (~) i "wt". Hem d'especificar-li també de quina base de dades ho hem extret, en aquest cas "mtcars".

```
model <- lm(mpg ~ wt, data = mtcars)
```

I això que acabem d'executar és un model de regressió lineal. No entrarem a interpretar-lo, i el que farem és afegir-li una recta. La recta s'afegeix amb aquesta funció d'aquí, *abline()*, que permet afegir rectes verticals, horitzontals o rectes amb pendent, com és aquest cas.

```
abline(model)
```

Si executem, veiem que acabem d'afegir la recta a sobre dels nostres punts. Podem millorar una mica aquest gràfic, modificant alguns paràmetres. Per exemple, canviant el tipus de punt, fent-lo més gruixut o afegint-li un títol.

```
plot(x=mtcars$wt,y=mtcars$mpg, pch = 4, lwd=1.5, main = "Recta de regressio")
```

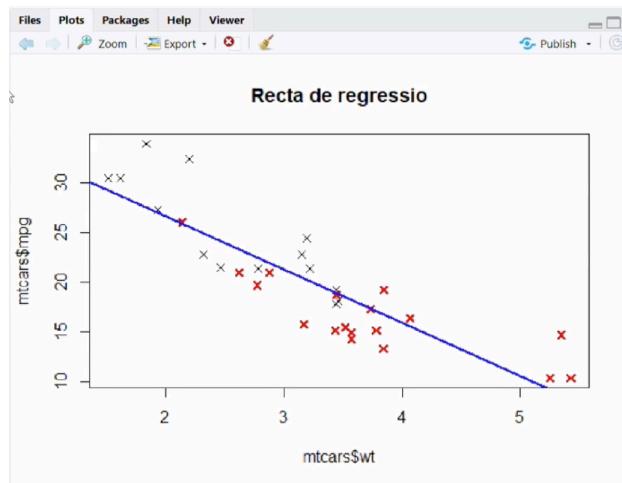
El mateix podem fer amb la funció *points*. Ja hem vist que li hem canviat el color. Ara, perquè no desentoni, els posarem d'una grossor una mica superior als altres i del mateix tipus.

```
points(x=mtcars$wt[filtre],y=mtcars$mpg[filtre], col="red", lwd=2, pch=4)
```

I aquesta recta que estem afegint, la canviarem de color també, i també la farem més gruixuda. Podríem canviar-li el tipus, però això ho veurem més endavant.

```
abline(model,col="blue",lwd=2)
```

Si ara executem totes aquestes instruccions, obtenim la recta de regressió sobre les nostres dades filtrades basant-nos en dues categories i, al final, acaba quedant amb aquest format.



I ja ho tindríem. Continuem!

4.5 NÚVOL DE PUNTS I RECTA DE REGRESSIÓ (II)

Hola de nou!

En aquest segon vídeo veurem com interpretar l'output que ens genera la funció *lm*, quins són els elements més importants a considerar i com podem incorporar-los a la comunicació dels nostres processos d'anàlisi.

Aquí veiem el codi que genera el model de regressió simple, és a dir, estem ajustant el consum per galó, utilitzant el pes de tots aquests cotxes.

```
model<- lm(mpg ~ wt, data = mtcars)
```

Si ara mirem què tenim dins de l'objecte *model*, tenim aquesta fórmula d'aquí, és a dir, un coeficient que intercepta un dels eixos i un coeficient associat a la variable.

Aquest valor d'aquí és el pendent de la recta. Un valor negatiu indica que com més valor d'aquesta variable, és a dir, com més pes, menor consum.

Si volem veure més informació, el que podem fer és *summary* del nostre model.

```
summary(model)
```

Una primera ullada a aquest resum ens permet veure que tenim molta més informació de la que segurament necessitem a nivell d'usuari. Tenim un estudi dels residus, un estudi estadístic dels coeficients, que, com podem veure, són els mateixos que teníem executant només el model, i un resum en text de les especificacions del model.

Segurament, l'element més important de tots és aquest d'aquí. Aquest nombre ens indica en tant per 1, és a dir, molt fàcilment convertible en un tant per cent, quin percentatge de variabilitat explica aquesta variable amb relació a aquesta. Dit d'una altra manera, en quin percentatge aquestes variables estan relacionades. En aquest cas, en un 74 %, si movem la coma dues unitats cap enrere. Un 74 % de relació lineal entre les variables és una relació molt elevada. Normalment les dades no presenten una relació tan forta, però en aquest cas, en tractar-se de cotxes, i variables tan relacionades com el pes i el consum, podem obtenir una interpretació molt forta de com es relacionen.

En termes estadístics, també és molt important aquesta regió d'aquí. Si aquests valors d'aquí són molt petits, en aquests casos ho són, ja que és 1,29 per 10 elevat a la -10, un nombre molt petit, el que estem dient és que aquesta variable d'aquí és molt rellevant. Va molt associat a aquesta variable d'aquí, però si estem ajustant un model de regressió múltiple, per exemple afegint-li aquí més variables, podríem veure quines d'aquestes són més importants.

Continuem!

4.6 PERSONALITZACIÓ DE GRÀFICS (I)

Hola de nou!

En aquesta lliçó veurem com podem afegir elements addicionals als nostres gràfics, que tant poden aportar nova informació al lector com fer-lo més atractiu. Per veure-ho, treballarem en els gràfics que hem generat anteriorment.

Una primera millora que podem aplicar, i és recomanable fer-ho sempre, és modificar els noms dels eixos amb els paràmetres *xlab* i *ylab*.

Així doncs, el que utilitzarem són aquestes dues instruccions d'aquí. Per exemple, pes i consum.

```
plot(mtcars$wt, mtcars$mpg, xlab="Pes", ylab="Consum")
```

Ja havíem vist anteriorment que podem modificar el títol d'un gràfic utilitzant la instrucció *main*.

```
main = "Pes vs Consum"
```

Si el que ens interessés és centrar-nos en una regió concreta d'aquest gràfic, el que podríem fer és dir, per exemple, que volem mirar només els vehicles que estan entre 3 i 4 tones de pes. Això ho podríem fer, utilitzant aquesta instrucció. Això dependrà d'un vector, on li haurem d'especificar el mínim i el màxim.

```
xlim = c(3,4)
```

El que acabem de fer és aplicar un zoom sobre el nostre gràfic.

Ara estem veient que no necessitem tot aquest espai d'aquí dalt, així que podríem aplicar exactament el mateix per a l'eix vertical.

```
ylim = c(10,25)
```

I suggerim afegir un codi de colors més atractiu. A diferents paquets podem trobar gammes cromàtiques bastant més atractives que els colors bàsics d'R o la instrucció *rainbow*, que ja hem vist abans, que genera gràfics molt infantils. Uns bons exemples són els que veurem a continuació.

Utilitzarem la funció, per exemple, *terrain.colors*, i li haurem d'especificar un número, que serà el nombre de colors que ens estarà generant la funció.

```
col=terrain.colors(20)
```

Per exemple, podem utilitzar-ne 20 i, si ho executem, veurem aquesta seqüència de colors. Què tenen en comú? Que pertanyen tots a una gamma cromàtica concreta, que fa que l'estil sigui molt més atractiu.

Anem a fer els punts una mica més visibles i el que veiem és que tots els colors mantenen una certa tonalitat.

```
lwd=4
```

Una altra gamma cromàtica bastant interessant és *heat.colors*. És una gamma més basada en els vermells i, per la impressió en blanc i negre, una de molt interessant és *gray.colors*, que genera una impressió on està utilitzant només la gamma de blanc a negre. És important

considerar que aquí el que estem fent és acolorir els nostres punts, basant-nos en un criteri aleatori.

El més interessant seria donar-li una certa interpretació a aquest tipus de colors, per exemple fent que els grisos més foscos fossin els cotxes d'una certa categoria i els grisos més clars els d'una altra.

4.7 PERSONALITZACIÓ DE GRÀFICS (II)

En aquest segon vídeo de personalització de gràfics, veurem com afegir elements lleugerament més sofisticats a les nostres visualitzacions. Ens centrarem en la superposició d'informació estadística bàsica i d'elements textuais.

Anteriorment hem vist com afegir punts mitjançant la instrucció *points*. Podem fer el mateix amb la instrucció *abline()* per a línies rectes o *lines()* per afegir figures més complexes.

Anem a afegir, en el gràfic que tenim aquí, les mitjanes per a l'eix vertical i per a l'eix horitzontal.

Ja he comentat anteriorment que podíem utilitzar aquesta funció d'aquí, que és la que vam utilitzar per afegir el model de regressió lineal, per afegir línies horizontals i verticals. Ara, afegirem una línia horitzontal. Per tant, el que farem és calcular la mitjana de la variable *consum*. Utilitzem la funció *mean* de la variable *consum*, que és *mpg*. El mateix podem fer per afegir la mitjana per a la variable *pes*.

```
abline(h = mean(mtcars$mpg))
```

Ara haurem de modificar el paràmetre horitzontal i dir-li que la volem en vertical.

```
abline(v = mean(mtcars$mpg))
```

Si ho executem ara, veurem que no fa res, ja que no hem modificat aquesta variable d'aquí. No és que aquesta instrucció no hagi funcionat, senzillament ens ha dibuixat una recta que cau fora de l'espai que tenim aquí.

Si ara selecciono la variable correcta, veuré que acabo de generar un eix de coordenades i d'ordenades que permet situar com de disperses estan les dades en base a les mitjanes de les seves categories.

```
abline(v = mean(mtcars$wt))
```

Per fer-ho més atractiu, podem modificar els colors d'aquestes línies, fent-les més apagades.

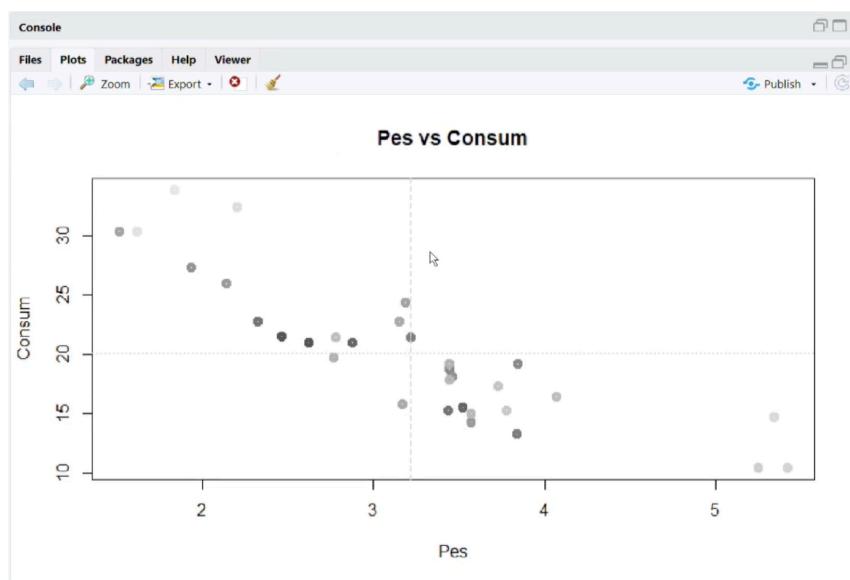
```
col = "lightgray"
```

I podríem canviar-los el tipus. Per exemple, amb aquesta instrucció.

```
lty=3
```

Escollim el 2 i el 3, per veure com funcionen i, si ho executem, veurem que no fa pràcticament res, almenys no perceptiblement.

El que ha passat és que ens ha superposat una línia d'un tipus diferent a sobre la línia que ja teníem. El que hem de fer és tornar a executar el nostre gràfic i ara veurem correctament el que estem fent; és a dir, estem afegint unes rectes discontinues amb una freqüència de talls diferent entre elles.



Per acabar aquest vídeo, veurem com podem afegir text rellevant als nostres gràfics. Ja hem vist com afegir-li un títol, nom als eixos i ara el que veurem és com podem afegir text en tot aquest espai d'aquí.

La funció és la funció `text` i depèn de tres paràmetres. En primer lloc, li hem d'especificar en quina coordenada en l'escala de l'eix X volem que seleccioni, per exemple 4 (seleccionarà aquesta línia vertical), i quina coordenada en l'eix vertical, per exemple podria ser el 25 (i seleccionarà aquesta línia horitzontal).

```
text(x=4,y=25)
```

El tercer paràmetre que li especifiquem és el text que volem afegir, per exemple “El meu cotxe”.

```
text(x=4,y=25, "El meu cotxe")
```

El que ha fet és afegir un text centrat exactament en el punt que li he demanat. A mi m'interessaria, per exemple, remarcar que el punt interessant és aquest d'aquí. Aquest tipus de modificacions es realitzen normalment fent assaig i error. Per exemple, jo aquí el que faria és seleccionar un punt, per exemple el 3, i una altura, per exemple el 32. I veure què tal queda.

```
text(x=3,y=32, "El meu cotxe")
```

Hauria de ser una miqueta més amunt i una miqueta més cap a l'esquerra. 32,5 (no oblidis que els decimals es fan amb un punt) i una miqueta cap a l'esquerra, 2,8, per exemple.

```
text(x=2.8,y=32.5, "El meu cotxe")
```

Com veiem, s'està superposant als meus textos anteriors, ja que R entén afegir nous elements a un gràfic com si fos una etiqueta a sobre del gràfic que ja teníem. Així doncs, si torno a executar-ho tot, obtindré només l'etiqueta definitiva.

Seguim!

4.8 AFEGIR LLEGENDES ALS NOSTRES GRAFICS

Hola de nou!

El procés d'afegir una llegenda a un gràfic en R es realitza manualment, la qual cosa permet un nivell de personalització molt elevat, però se li ha de dedicar temps. Per veure un exemple de codi, crearem un gràfic de punts filtrat amb tres categories i li donarem color.

És important notar el procés amb el qual hem construït aquest gràfic.

```
plot(mtcars$wt,mtcars$mpg,pch = 4,lwd=1.2, col = "orange",main = "Recta de Regressió",ylab = "Consum", xlab = "Pes")
```

En primer lloc, dibuixem tots els punts i afegim títol i nom dels eixos. Després, creem un filtre. Aquest filtre servirà per acolorir aquells cotxes amb un cilindre de 4 unitats. Aquests punts els pintaré de color vermell. Uns certs paràmetres.

```
filtre <- mtcars$cyl == 4
points(mtcars$wt[filtre],mtcars$mpg[filtre],col="darkred",lwd=2,pch=4)
```

Utilitzant el mateix nom, és a dir, sobreescrivint aquest filtre, tornaré a utilitzar la mateixa estructura per cridar els punts pintats de color verd.

```
filtre <- mtcars$cyl == 8
points(mtcars$wt[filtre],mtcars$mpg[filtre],col="darkgreen",lwd=2,pch=4)
```

La funció per crear llegendes es diu *legend*.

```
legend("topright",legend = c("4 cyl","6 cyl","8
cyl"),col=c("darkred","orange","darkgreen"),pch = 4,cex = .8,title = "Llegenda", horiz = TRUE)
```

Aquesta funció depèn de moltes paràmetres. El primer de tots és la posició de la llegenda. La posició de la llegenda la podem especificar numèricament, és a dir, amb els valors de les coordenades; per exemple, 4 i 25. Ens posaria la llegenda aquí. Però jo recomano, almenys per començar, utilitzar el format actual, per exemple “*topright*”. El que estem fent, especificant-li “*topright*”, és que volem que ens posi la llegenda a la part superior de la dreta. Si executem aquesta instrucció, es queixarà; ja que no li estem dient què volem que ens dibuixi. Ens falta el paràmetre *legend*. Aquí, en format vectorial, li hem de posar tots els elements que volem que ens mostri. Així doncs, volem que ens mostri 4 cilindres, 6 cilindres i 8 cilindres (*legend* = c("4 cyl","6 cyl","8 cyl")). Si ara executem, veiem que ens ha afegit un requadre amb els tres noms que li hem especificat. Això, òbviament, ho podem millorar.

Una primera millora a considerar és afegir-li color. Ara veurem com funciona. Li hem d'especificar el paràmetre *color* i, en un vector, li assignarem un color a aquest element, un color a aquest i un color a aquest (*col=c("darkred","orange","darkgreen")*). Ja veurem què passa.

Acabem d'executar i no ha canviat absolutament res. Bàsicament, no ha canviat perquè no li hem dit quin tipus d'element volem que associi a aquests noms. El que hem de fer, per exemple, és dir-li que siguin punts del mateix tipus que els que estàvem dibuixant, és a dir, tipus 4 (*pch = 4*).

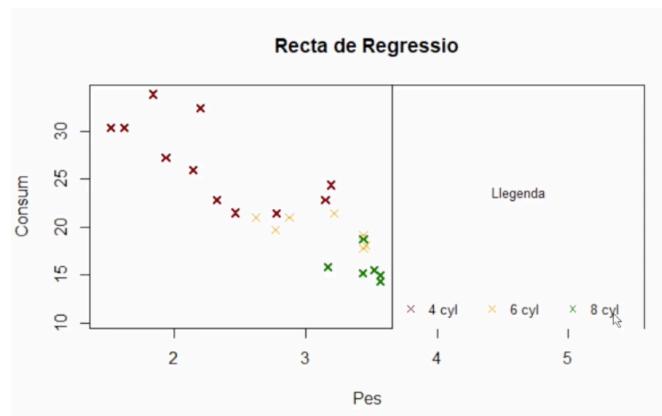
Ara ja apareixen aquests elements. Si poguéssim mostrar diferents tipus de punt (no és el cas en aquest gràfic), però jo ho mostro, podríem fer-ho així (*pch = c(2,4,6)*). I ara tindríem tres punts diferents: 2, 4 i 6, en funció del primer element, el segon element i el tercer element. Deixem-ho amb 4. Modifiquem la mida del text, utilitzant aquesta instrucció d'aquí (*cex = .8*). Veiem que s'ha superposat a sobre la llegenda anterior. Això és el que estava passant fins ara, però ara com que hem variat la mida, apareix superposada i es veu aquesta superposició.

Podem afegir-li també un títol. Ara, en comptes de *main*, és *title*. Li'n direm "Llegenda" (*title* = "Llegenda"). I, per últim, atès que és una llegenda que potser ocupa una mica massa d'espai, la posarem en horitzontal, utilitzant aquest paràmetre, que serà "veritat" o "fals" en funció de com la vulguem (*horiz* = TRUE). Executem i tenim aquesta llegenda d'aquí.



Per recuperar el gràfic i que s'esborrin aquestes llegendes que no necessitem, hem de tornar a executar tot el codi.

És important notar que, quan tenim un gràfic i estem afegint-li elements, importa la mida de la pestanya que tenim. Així doncs, si jo tinc un gràfic així, i li afegeixo la llegenda; tornant a executar, un cop ho faci gran, la llegenda em quedarà en aquest format.



Òbviament, això no és una bona visualització, així que s'ha de considerar que la mida de la pestanya que tinguem a l'hora d'executar és molt rellevant, ja que defineix també la mida de la llegenda que li estem afegint al gràfic.

Continuem!

4.9 IDEES CLAU: VISUALITZACIÓ DE DADES

Ja som al final del mòdul. A continuació, farem un repàs de les idees més importants que han aparegut.

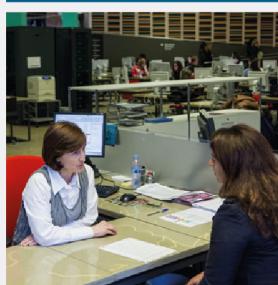
En aquest mòdul final hem vist una breu introducció a la generació i **edició de gràfics en R**, la seva exportació, la creació de gràfics múltiples, així com maneres senzilles d'enriquir les nostres visualitzacions per fer-les més atractives i útils per als lectors o oients. Els diferents tipus de gràfics que hem explorat són:

- Diagrama de sectors
- Diagrama de barres
- Histograma
- Diagrama de caixa

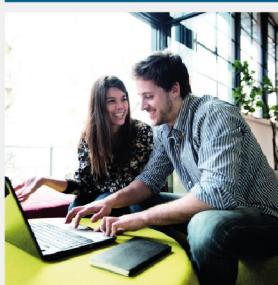
- Núvol de punts i recta de regressió

L'elecció del tipus de gràfic i les modificacions que hi fem han de venir sempre donats com una resposta als resultats que vulguem comunicar, per la qual cosa és necessari explorar prèviament i en profunditat les dades amb les quals estiguem treballant.

Descobreix tot el que Barcelona Activa pot fer per a tu



Acompanyament durant tot el procés de cerca de feina
barcelonactiva.cat/treball



Suport per posar en marxa la teva idea de negoci
barcelonactiva.cat/emprendoria



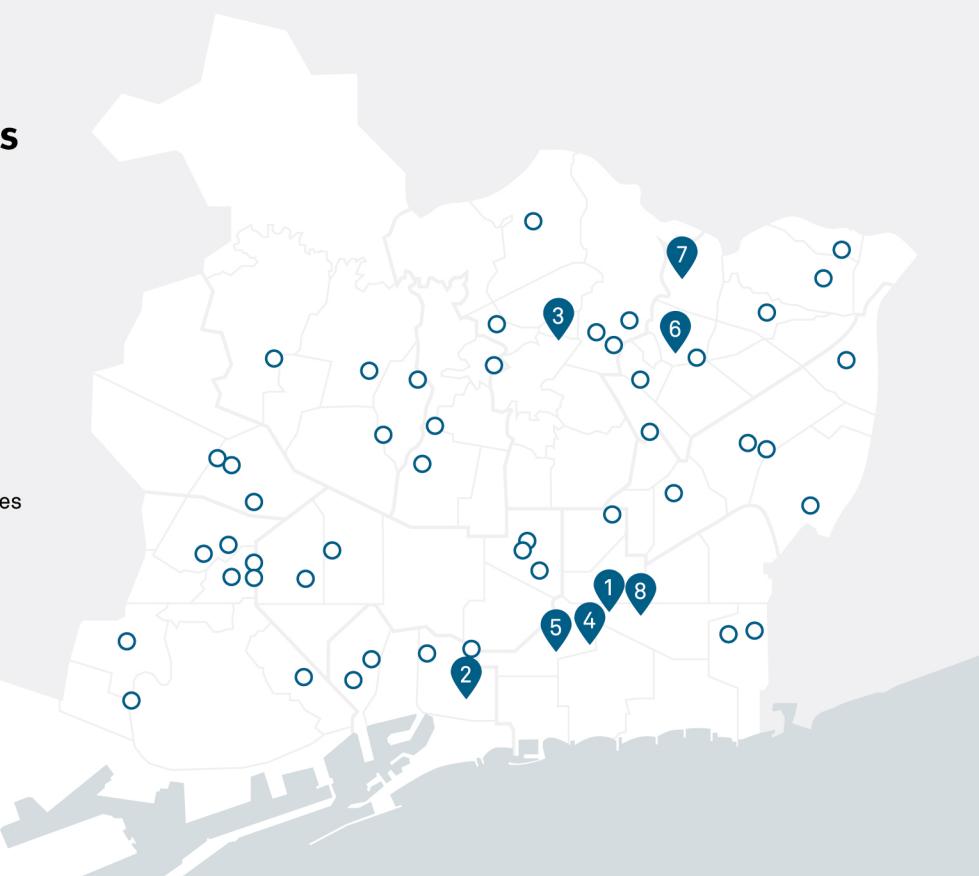
Serveis a les empreses i iniciatives socioempresariales
barcelonactiva.cat/empreses



Formació tecnològica i gratuïta per a la ciutadania
barcelonactiva.cat/cibernarium

Xarxa d'equipaments de Barcelona Activa

- 1 Seu Central Barcelona Activa
Porta 22
Centre per a la Iniciativa Emprenedora Glòries
Incubadora Glòries
- 2 Convent de Sant Agustí
- 3 Ca n'Andalet
- 4 Oficina d'Atenció a les Empreses Cibernàrium
Incubadora MediaTIC
- 5 Incubadora Almogàvers
- 6 Parc Tecnològic
- 7 Nou Barris Activa
- 8 innoBA
- Punts d'atenció a la ciutat



© Barcelona Activa
Darrera actualització 2018

Cofinançat per:



Segueix-nos a les xarxes socials:

-  barcelonactiva.cat/empreses
-  [barcelonactiva](https://www.facebook.com/barcelonactiva)
-  [barcelonactiva](https://twitter.com/barcelonactiva)
-  [company/barcelona-activa](https://www.linkedin.com/company/barcelona-activa)