Introducció a l'anàlisi de dades amb R

Índex

1 INTRODUCCIÓ AL R I A 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, a 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.

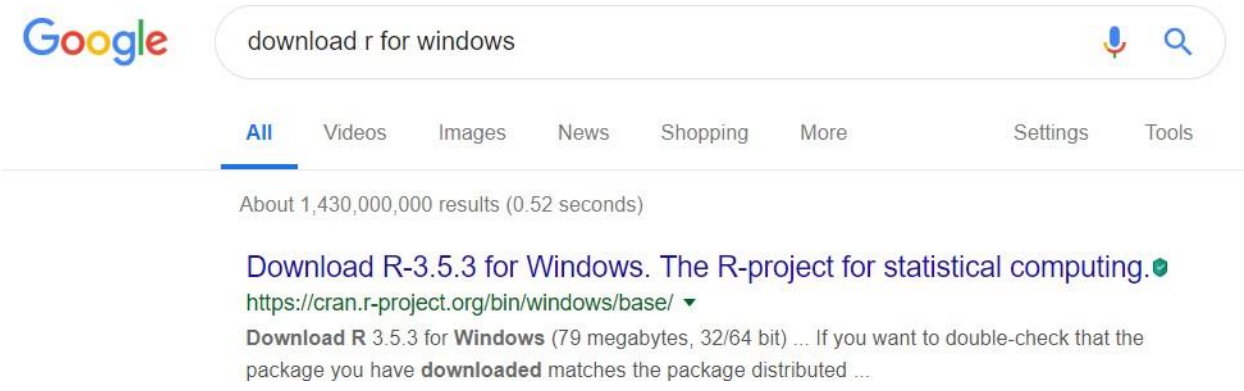
Seguim!

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, independentment del sistema operatiu amb el qual estiguem treballant.

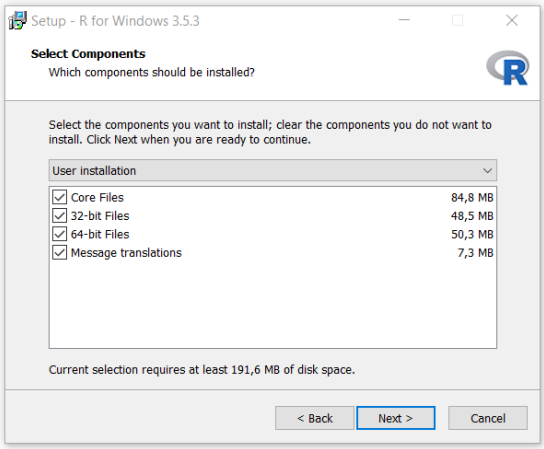
El primer pas que realitzarem és buscar "Install R" a Google i entrarem a la primera pàgina que ens aparegui.

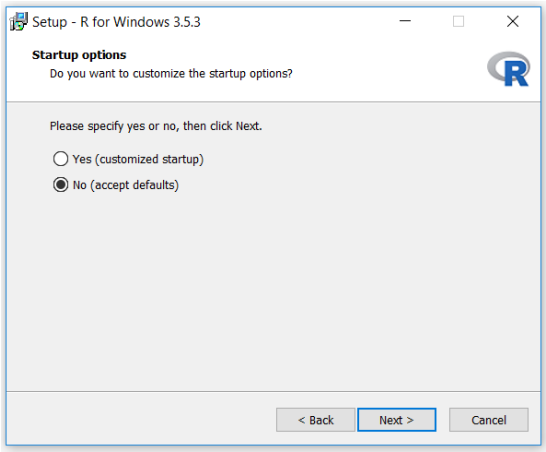


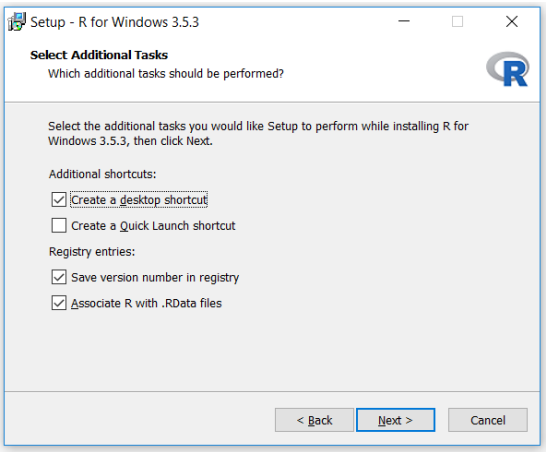
Potser la web no és prou amable, ja que es tracta d'un projecte lliure. Haurem de clicar a "**Download R**", ressaltat en negreta, i ens redirigirà a una pàgina de servidors.

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'ns obri l'assistent d'instal·lació.







Fins aviat!

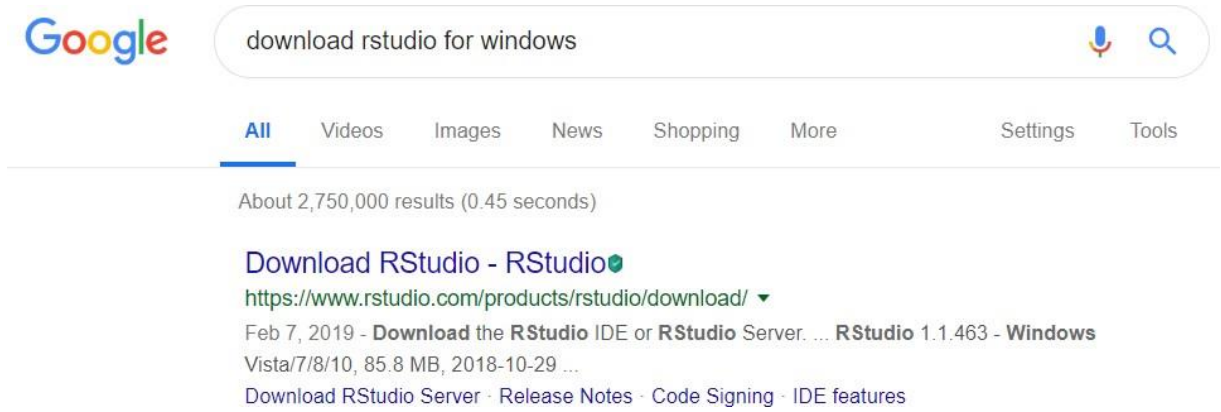
1.3 INSTAL·LACIÓ D'RSTUDIO

Hola de nou!

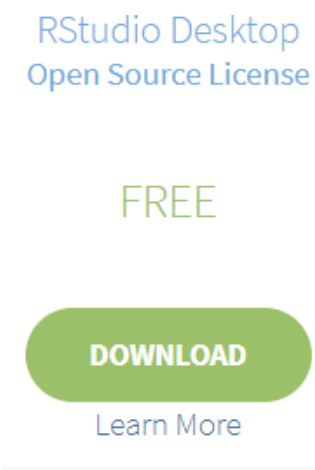
En aquest article avançarem en el procés d'instal·lació de la interfície gràfica RStudio, una interfície gràfica que treballa sobre R i que ens facilitarà enormement la feina, a la vegada que ens permetrà una visualització més agradable i estructurada de l'entorn de programació.

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

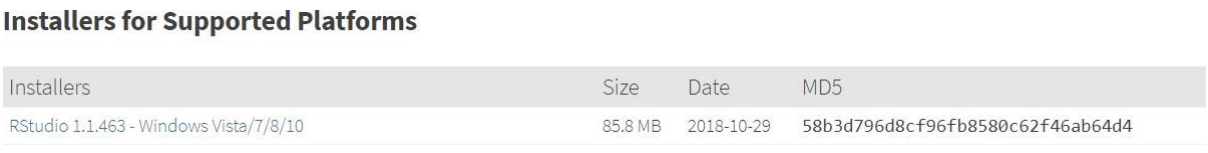
El primer pas que realitzarem és buscar "Install RStudio" a Google i entrarem a la primera pàgina que ens aparegui.



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



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.



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

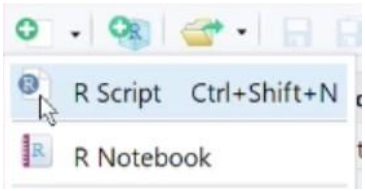
Continuem amb el curs!

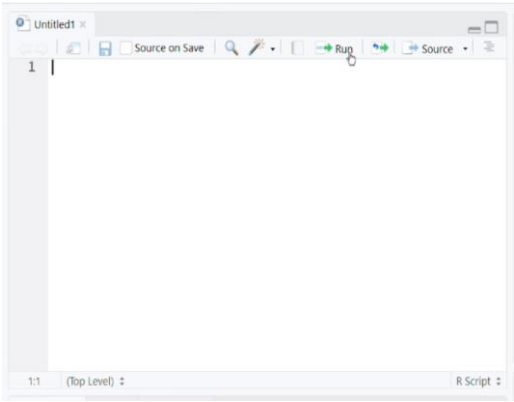
1.4 INTRODUCCIÓ A L'ENTORN DE PROGRAMACIÓ RSTUDIO

Benvinguts i benvingudes!

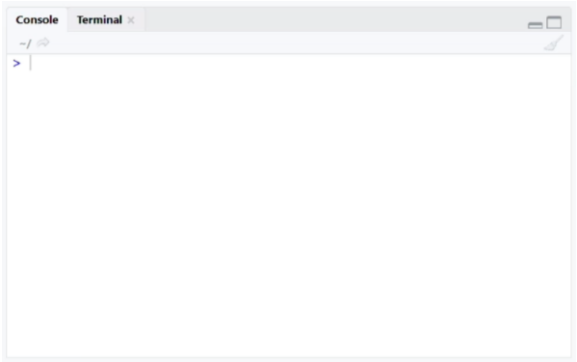
En aquest vídeo presentarem la interfície gràfica RStudio, així com les funcionalitats més interessants que ens pot oferir. Abans de comentar una mica més en detall el que ens trobarem, hem de tenir en compte un consell. Si tenim un cert domini de l'anglès, el més pràctic és veure la interfície en aquest 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:

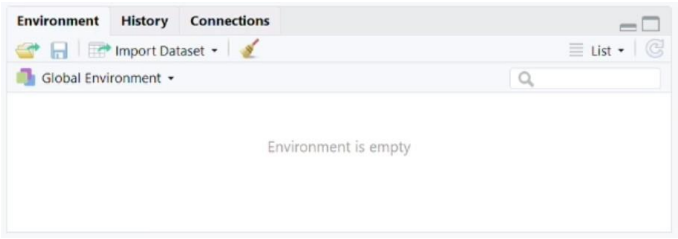


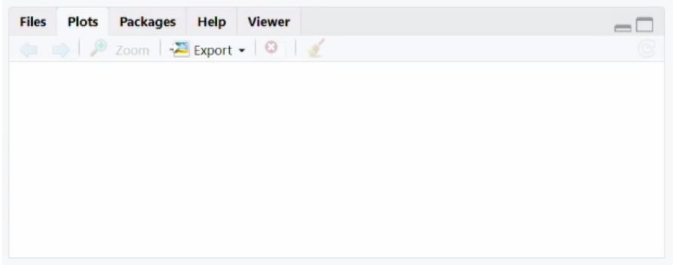


I obrint el document on escriurem el nostre codi. Tot el que escrivim aquí i executem utilitzant aquest botó d'aquí, ho veurem per la consola.



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 veuríem 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.

```
edat <- 20
```

I si l'individu és home o dona.

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

Utilitzant l'estructura *"if"*, podem preguntar si l'edat 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 (>=) 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 està avaluant 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)* i, 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)*" classifiquessis 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 especifiquem, 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 te 366 dies")
```

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

```
else{
return("L'any te 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 te 366 dies")
}else{
return("L'any te 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 r*eturn*(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.

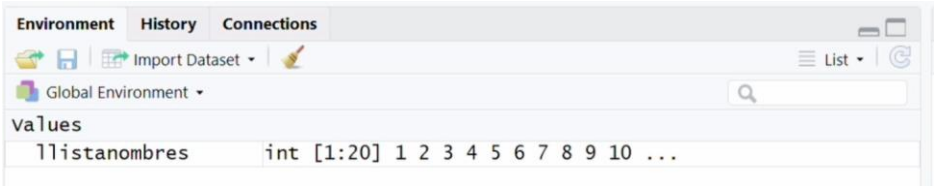
```
llistanombres <- 1:20
```

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( llistanombres[i] ^2 )
```

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

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

for(i in 1:length(llistanombres)){
print( llistanombres[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 (*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 càlcul 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* i *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.