Sessió 1

Preliminars de *R*

1.1 Iniciant R i RStudio

R és un programari per a la manipulació, anàlisi i representació gràfica de dades. És també un llenguatge de programació, orientat a l'anàlisi de dades i a la representació gràfica. És *open source* i gratuït, i una eina molt potent utilitzada i desenvolupada en col·laboració per professionals de l'estadística de tot el món.

R es pot trobar a

`http://www.r-project.org`

on hi ha, a més del codi font, els executables per a una varietat de sistemes operatius. El més pràctic és baixar els executables (*binaries*) per al sistema operatiu adequat.

Es tracta d'un sistema extensible, a partir de paquets (*packages*) que es poden instal·lar addicionalment al sistema bàsic en el moment que es necessitin, i als quals tothom pot contribuir. Actualment **R** té més de 10 000 paquets disponibles. **R** té una versió comercial amb menús, S-plus, amb el qual comparteix moltes ordres.

R es pot fer córrer des de la terminal del sistema o bé des de diferents entorns, com ara **RStudio** o **RGui**. Nosaltres utilitzarem **RStudio**, que és un entorn més amable per a l'usuari i que proporciona menús per gestionar paquets, variables, obrir scripts, generar informes, etc. És també lliure i disponible per a Windows, Linux i Mac Os. Podeu trobar-lo a

`http://www.rstudio.com/`

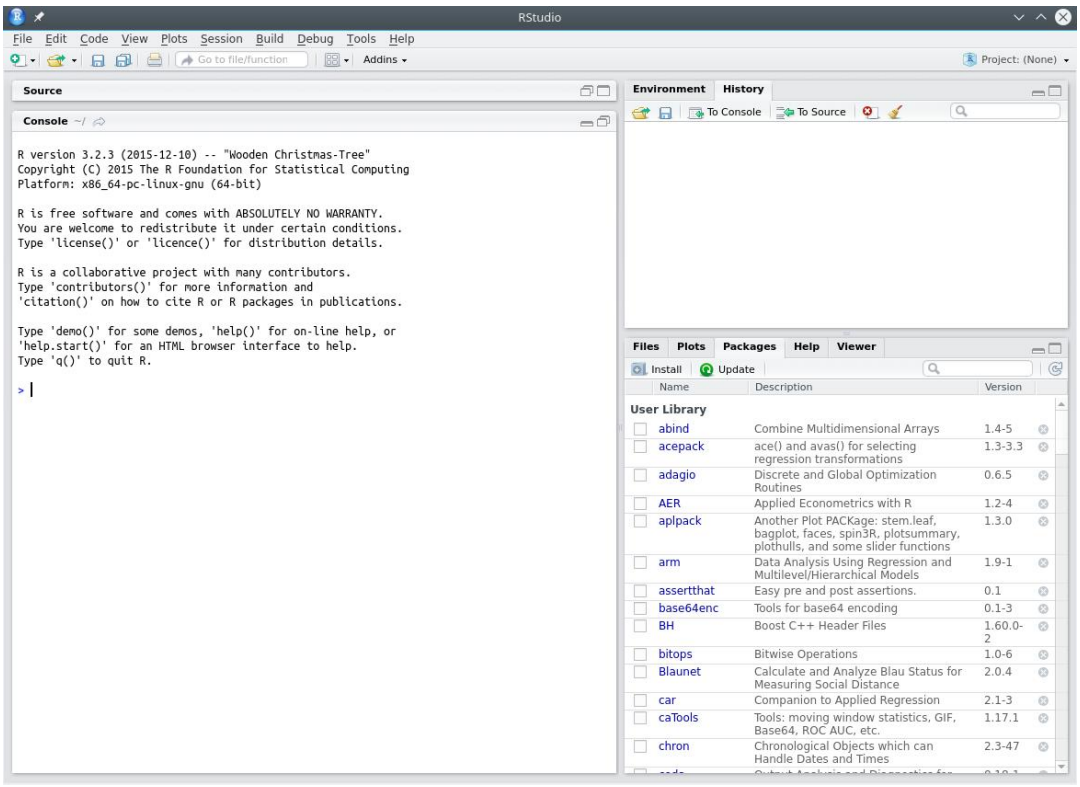
Per instal·lar **RStudio** convé instal·lar abans **R** i a continuació **RStudio**.

Tot i que començarem a treballar amb **RStudio**, en aquesta primera sessió ens limitarem només a aprendre a utilitzar **la consola** per executar ordres i a **crear scripts** per guardar la nostra llista d'instruccions i poder-les recuperar. Podeu obrir la consola i l'editor de scripts amb **RGui** o bé directament des de **RStudio**.

1.2 Executar instruccions

A la consola de **R** hi introduïm instruccions que després executem. Aquesta és la manera més bàsica de treballar, i tots els entorns de **R** tenen aquesta finestra.

Habitualment, quan obrim **RStudio** per primer cop té l'aspecte següent:



Ara ens limitarem de moment a utilitzar només la consola.

1.2.1 Executar instruccions a la consola

Les ordres es poden executar de manera interactiva escrivint després del *prompt* > a la consola i prement seguidament **Enter**. S'exectuen una després de l'altra.

```
> 3+4
[1] 7
> exp(2)
[1] 7.389056
> 1:54
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
```

• L'índex `[1], [19], etc.` que apareix en la llista de resultats a l'inici de cada fila indica la posició dins del vector de resultats.

• El símbol prompt > no s'ha d'escriure, ja apareix a la consola.

• Per recuperar una instrucció ja executada podeu utilitzar les fletxes ↑ i ↓ del teclat.

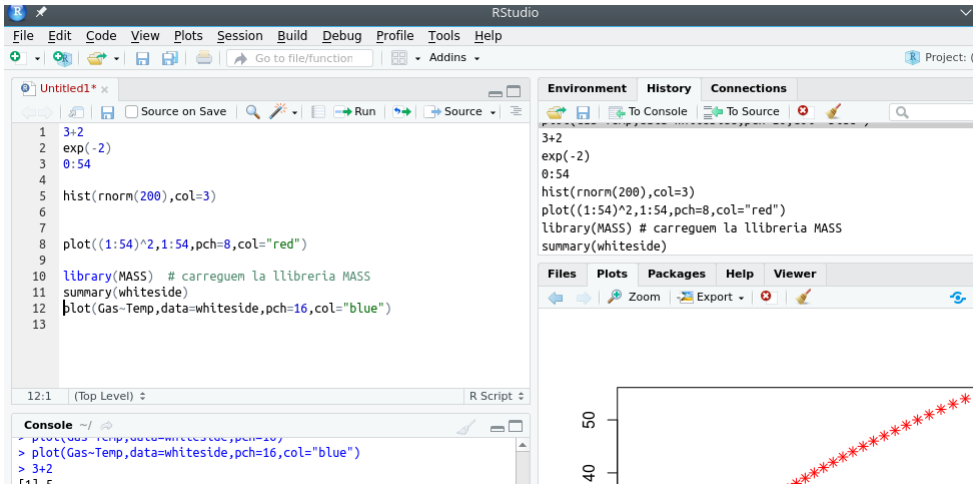
Com a mostra de càlculs i gràfics que es poden fer amb **R** proveu d'executar les instruccions següents, que més endavant explicarem:

```
> plot((1:54)^2,1:54,pch=8,col="red")
> plot(rnorm(50),rnorm(50))
> plot(1:20,pch=1:20)
> hist(rnorm(200),col=3)

> summary(CO2)
> summary(airquality)

> plot(Ozone~Solar.R,data=airquality,pch=16,col="blue")

> pie(table(apply(cbind(sample(1:6,100,replace=TRUE),
        sample(1:6,100,replace=TRUE)),1,sum)),col=c(2:6,7,6:2))
```

1.2.2 *Crear* scripts *i executar-los amb* ***Ctrl + ENTER***

Un *Script* de **R** és un arxiu de text sense format que conté una llista d'instruccions de **R** i que es pot guardar a l'ordinador per obrir-lo en una altra sessió o bé executar-lo sense necessitat d'obrir-lo.

Per crear un nou script, aneu al menú **File → New File → R Script** de **RStudio**. S'obre una altra finestra a **RStudio**, que s'anomena **Source**.



- S'escriuen les instruccions a l'editor i s'executen des del mateix *script* (fixeu-vos que **no s'ha de posar el *prompt* >).**
- Un cop escrites, se seleccionen totes les instruccions i després la combinació de tecles **Ctrl + ENTER**.
- També es poden executar línia a línia, situant el cursor damunt de la línia i fent **Ctrl + ENTER**.
- Amb el **coixinet** # podem escriure comentaris, que van molt bé per recordar després el procediment seguit o bé fer observacions sobre els resultats.

Guardar un script

Pràctica: Anant al menú **File → Save As**, guardeu el fitxer de la finestra **Source** amb el nom **`codi0.R`** en una carpeta que anomenarem `prac1`. Tanqueu el fitxer. Al menú **File → Open File** podeu obrir novament el fitxer i tornar-hi a treballar.

1.3 Expressions i assignacions a variables

Al nostre script hi escrivim instruccions, que poden anar separades per punt i coma `;` o bé en línies diferents.

- En general, les expressions donen un resultat a la consola i no es guarden.
- Però hi ha expressions que són **assignacions** i serveixen per guardar un resultat. Es fan amb `<-` Guarden el resultat d'una expressió en el que anomenem una **variable**.

  Seguidament, veiem com s'assigna el resultat de `3+2*5` a la variable `a`. Fixem-nos que **`R`** distigeix `A` de `a`.

```
3+2*5          # expressió
a <-7          # assignació, no escriu el resultat
a              # veiem el valor
a<-3+2*5       # assignació el valor resultat del càlcul
a
A<-(3+2)*5     # R distingeix majúscules i minúscules
A; a           # veiem els dos valors
a-A       # utilitzem a i A per fer un càlcul
B <- a-A    # assignem a B l'expressió anterior
```

- **Noms de les variables:**
  - **`R`** distingeix majúscules i minúscules com a noms diferents de variables.
  - Les variables poden tenir noms llargs, començant sempre per lletra majúscula o minúscula o bé un punt (.). Les variables no poden començar amb un número.
  - Després de la lletra inicial en el nom de la variable podem posar més lletres, números i NOMÉS els símbols `.` i _ (exemples: `llista.1`, `Llista_1`, `a`, `basedades`).
  - Una variable no pot començar amb un punt seguit d'un número.
  - Hi ha noms "reservats" per a funcions i constants de R que lògicament no es poden utilitzar, com ara `if else repeat while function` o `TRUE FALSE NULL Inf NaN NA` Consulteu quines són amb l'ordre `help(reserved)`.
  - No podem utilitzar lletres accentuades ni la ç ni la ñ.
- També es poden recuperar les expressions ja executades amb la "fletxa" ↑ del teclat
- Per ajuntar grups de instruccions, les posem dins de claus `{}`.

```
{h<-5; p<-3; h+p}      # només mostra l'últim resultat !!
```

- Fixem-nos que quan fem una assignació, com `x<-3+2^5`, no surt cap valor per pantalla. Si volem veure el valor, hem d'executar en una altra línia `x`. Una manera alternativa de fer una assignació i alhora veure el valor que pren la variable és **escrivint l'assignació entre parèntesi:**

```
> (x<-3+2^5)    # assignació entre parèntesi, apareix el valor de x
35
```

Quines assignacions hem fet? La funció ls()

La funció **ls()** ens mostra la llista de variables que hem creat a la sessió. També apareixen a la pestanya **Environment** (entorn de treball).

Pràctica:
- Crea una variable `x` que contingui el resultat de l'operació 3^5 i una variable y que contingui $1/31$.
- Troba la seva suma i assigna el resultat a la variable z.
- Comprova amb la funció `ls()` que has creat les variables `x, y, z`.

1.4 Tipus de valors

- Tenim tres tipus de valors principals:
  - *numeric* (número amb decimals)
  - *character* (cadenes de caràcters)
  - *logical* (valors lògics)
- Els possibles valors lògics són `TRUE` i `FALSE` -majúscules- (també `T` i `F`, però millor no utilitzar-ho) i quan es tracten numèricament es converteixen en `1` i `0`, respectivament:

```
TRUE
FALSE
TRUE+FALSE; TRUE*FALSE; TRUE+TRUE
3-TRUE; 1-FALSE     # en operar 3 amb TRUE,  el TRUE es converteix en 1.
```

- Vegem exemples de codificacions de valors (*numèric*, *lògic*, *caràcter*). La funció `mode` ens diu quina codificació té l'objecte.

```
x<-TRUE          # lògic
y<-3+4^2       # numèric
z<-"vermell"   # caràcter
mode(x); mode(y); mode(z);
```

- Conversió automàtica de tipus per operar amb el "mode més general"

```
x+y    # lògic i numèric ---> numèric
x+z    # !! error: un caràcter no és operable
y+y; y*y; x*y; (1-x)*y
```

- Conversió forçada de modes (si és factible): `as.`

```
as.numeric(x)
as.character(x)
as.logical(as.numeric(x))
```

- Fixem-nos què passa si executem `{x<-as.character(45); x^2}`

Pràctica:
• Crea les variables $a1$, $a2$ i $a3$ que continguin respectivament els valors 3, `TRUE` i "hola".
Comprova què passa amb les operacions

$$a1 + a2, \quad 3 \cdot a1, \quad a2 + a3, \quad a1 + a3$$

• Amb la funció `mode` treu per pantalla el tipus de cada una de les tres variables.

1.5 Creació de vectors.

Els vectors són un objecte bàsic molt utilitzat amb **R**. Veurem com crear vectors numèrics, cadenes de caràcters (*strings*), etc.

Els vectors són objectes formats per diversos elements o components que han de ser **del mateix tipus** (mateix *mode*).

- La funció de *concatenació* `c( )` és la manera més bàsica i universal de definir vectors

```
D<-c(0,3,3,3,3,1,-2);D     # introdueix un vector
D1<-c(D,-3); D1            # afegim una component al vector D
c(D,D1)                    # concatenem dos vectors
```

- Les funcions **seq**, **rep** i **els dos punts** defineixen seqüències i creen vectors
  - Els **dos punts** serveixen per generar un vector format per números consecutius, per exemple `3:6` equival a escriure `c(3,4,5,6)`.
  - La funció `seq` crea també un vector.
    - Podem donar només inici i final: `seq(from=3,to=6)` és el mateix que `3:6` i que `c(3,4,5,6)`. També podem escriure `seq(3,6)` o `seq(to=6,from=3)`.
    - Podem donar principi, final i l'increment: `seq(from=2,to=10,by=2)` equival a escriure `c(2,4,6,8,10)`. També podem escriure `seq(2,10,by=2)`
    - Podem fixar principi, final i la longitud del vector: `seq(from=0,to=10,length=5)` dóna el vector `c(0.0,2.5,5.0,7.5,10.0)`
  - La funció `rep` repeteix un valor els cops que li indiquem.
    `rep(5,times=3)` equival a `c(5,5,5)`. També podem escriure `rep(5,3)`
    Però també permet repetir vectors sencers `rep(c(1,2,3),times=2)` dona el vector $(1,2,3,1,2,3)$.
    - Si el primer argument és un vector, podem especificar el nombre de repeticions de cada component `rep(c(1,2,3),times=c(1,2,3)` dona $(1,2,2,3,3,3)$.
    - Finalment, podem repetir cada component del vector dos cops, per exemple:
    `rep(c(1,2,3),each=2)` dóna $(1,1,2,2,3,3)$.

A continuació, alguns exemples:

```
m0<-3; m1<-150
V<-m0:m1; V                # seqüència des de 3 a 150
V<-seq(from=m0,to=m1); V      # igual que l'anterior

W<-m0:m1-1; m0:(m1-1)      # atenció amb els signes + i - !!
-1:7; -(1:7)

V<-seq(from=m0,to=m1,by=5); V   # fixat l'increment
seq(-pi,pi,pi/8)
V<-seq(from=1,to=11,length=4);V # fixada la longitud
V<-rep(0,times=10); V           # per repetir un valor
V<-rep(V,times=2);V             # repetim dos cops el vector
rep(1:10,times=3)               # per repetir un vector
rep(1:10,times=rep(3,10))       # vegeu la diferència amb l'anterior
rep(1:10,each=3)                # el mateix que l'anterior
rep(c(1,2,3),times=c(1,2,3))    # una altra seqüència diferent
```

- Els vectors de caràcters (cadenes, *strings*) s'usen per guardar noms, etiquetes o valors de variables categòriques. Els valors es guarden entre cometes dobles o simples:

```
 letters               # vector amb totes les lletres minúscules
 LETTERS               # vector amb totes les lletres majúscules
 month.name            # vector amb noms dels mesos
 month.abb             # vector amb noms dels mesos abreujats
 c("Pepa","Ton","Jana")   # veiem apostrof per pantalla
 c('Pepa','Ton','Jana')   # també podem utilitzar apòstrof
 x<-c(1,2,'altre');x      # ho converteix tot a format caràcter
```

- Podem treballar amb vectors de tipus lògic

```
cert<-c(TRUE,FALSE,TRUE,TRUE, FALSE)
cert
cert<-c(TRUE,FALSE,TRUE,TRUE,FALSE)   # és  equivalent
```

- Iniciar vectors a *zero*, de longitud fixada, segons el tipus

```
vector("integer",3)   # o bé
integer(3)
vector("numeric",3); numeric(3)
vector("logical",3);  logical(3)
vector("character",3); character(3)
```

Pràctica:
• Considera les dades de 10 individus, per als quals s'han avaluat les variables:
`sex` (0=masculí, 1=femení): `0,0,0,1,0,0,1,1,0,1`
`edat` : `18,19,18,22,30,21,18,19,20,21`
`iest` (interés per l'estadística, de 0 a 10): `8,7,9,7,9,9,8,8,10,6`
`ipol` (interès per la política, de 0 a 10): `8,6,5,1,7,9,9,10,0,9`
Introdueix cada variable com un vector numèric.
• Utilitza les funcions rep, seq i els dos punts per crear els vectors següents:
$(3,4,5,6,7,8,9)$, $(1,1.5,2,2.5,3,3.5,4)$, $(1,1,2,2,3,3)$, $(3,2,1,3,2,1)$, $(5,5,6,6,6,7,7,7,7)$
• Quina és la diferència entre `rep(1:3,2)` i `rep(1:3,rep(2,3))`?
• Quin és el resultat d'avaluar `rep(rep(c(``X'',``Y''),rep(3,2)),1:6)`? Per què?

1.6 Valors especials

En estadística apareixen sovint els anomenats "valors perduts" (*missing*): dades que falten, ns/nc, etc.

Entenem que valors especials són:

Inf	Infinit
NaA	(*Not A Number*)
NA	valor perdut (*Not Available*, *missing*)
"	espai buit, caràcter perdut
NULL	no existeix

Vegem alguns exemples:

```
4/0; 0/0
## Resultat:  [1] Inf    [1] NaN
x<-c(3,3,4,,2)     # falta un valor i dóna error
x<-c(3,3,4,NA,2)   # ho entén
x<-c("x","y",)     # falta un valor i dóna error
x<-c("x","y","")   # ho entén
x<-c("x","y",NA)   # també ho entén
names(x)
 ## Resultat:  NULL   # no li hem assignat noms !
```