Data Analysis with R

Session 1

Joan-Josep VallbÃ©

Course basics

- Feel free to call me Pep
- No experience fundamentally needed
- 3:45 hour sessions
  - Two 10 min breaks, at 10:30 and 12:00
- Material in English, speech mainly in Catalan; Spanish if needed
- All course material open
- Several practical exercises every day (and some homework, too!)
- Ask all the questions you have, anywhere, anytime
- Stop me when something doesn't work out
- All the material is in ANALISIS DE DATOS EN R (dropbox)

Objectives

- This is all applied stuff, no theory
- This is **not** an *R User Manual* with lists of commands
- Get familiar with R, but not only that...
  - Get to know RStudio as an integrated development environment (IDE)
  - Adopt a useful Data Analysis Workflow (improve your efficiency as analysts)
  - Complete a data analysis project
- ... all this in just 4 days!

Course outline

- **Session 1: Introduction to R and RStudio**
  - The language and the environment
  - Basic R objects and actions
- **Session 2: Data Analysis Workflow**
  - Workflow
  - Data import and data munging
  - Data transformation
- **Session 3: Data management and graphics**
  - EDA and graphics
- **Session 4: Statistical analysis and reproducible reports**
  - Statistical analysis
  - Report generation and reproducible research

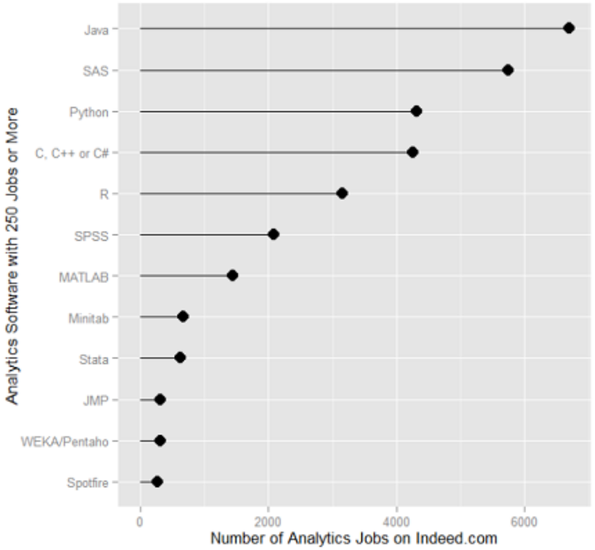
R Basic questions

- What is R?
  - It's a "programming language and software environment for statistical computing and graphics" (Wikipedia). It's also an open-source implementation of the **S** language for statistical programming.
- Open source?
  - Yes, you can download the software ready to install and even the source code, **for free**
  - It's a GNU Project (free software, mass collaboration)
- Who are the R authors?
  - **S** was created by John Chambers at Bell Labs, while **R** was initially written by Ross Ihaka and Robert Gentleman at the U. of Auckland, New Zealand.
  - **R** is now developed by the *R Development Core Team*

Why R?

- It's really versatile
  - It can be installed in Mac, Windows, Linux
  - It can be used from various interfaces
- It's free
  - The R project website: `http://cran.r-project.org/`
  - Versions are updated and improved regularly
- A collection of over 7,000+ libraries (called *packages*)
- An immense and active community in industry and academia
- Thousands of free online tutorials, templates and examples on **everything** R
- A strong environment to carry out all stages of **data analysis**

Why R?



Java
SAS
Python
C, C++ or C#
R
SPSS
MATLAB
Minitab
Stata
JMP
WEKA/Pentaho
Spotfire
0
2000
4000
6000
Number of Analytics Jobs on Indeed.com
Analytics Software with 250 Jobs or More

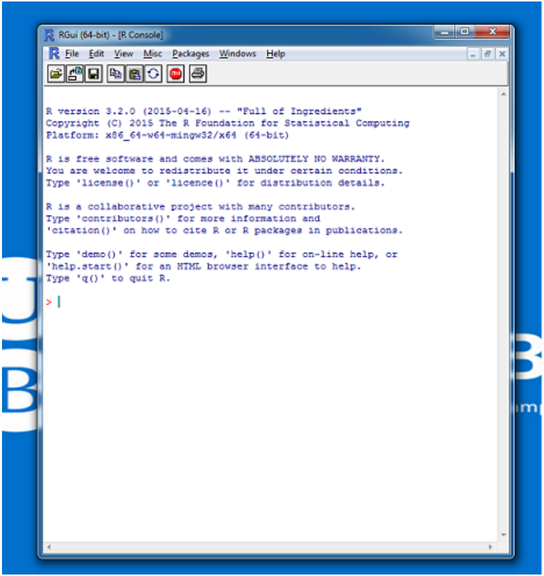
SPSS or Stata view of data

- data can be only in specified, **proprietary** formats (.sav,.dta)
- computations are done on one single dataset at a time
- row-by-column structure
- rows are cases and columns are variables
- variables have attributes: labels, types, levels, etc.

R view of data

- data are objects
- there are many types of objects
- you give objects a name you can call at any moment
- you can use ready-made *functions* to perform actions on those objects and get results
- you can create your own *functions*
- objects are not permanent unless you save them
- R works with objects in disk memory
- results from statistical analysis are also objects so that stuff can be done with them after the main analysis
- social scientist $\rightarrow$ data analyst and a programmer

The essence of R



R message

R version 3.2.0 (2015-04-16) – "Full of Ingredients" Copyright (C) 2015 The R Foundation for Statistical Computing Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.

Getting help

```
> help.start()
```

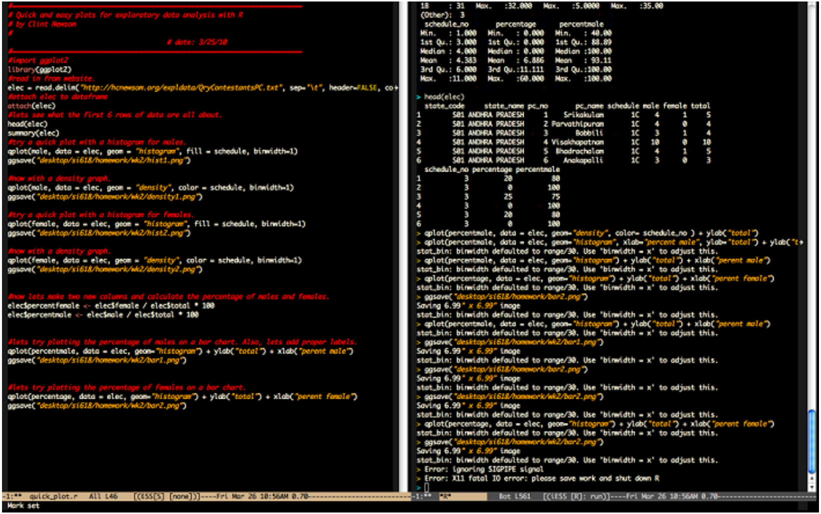
```
> help("lm")
```

```
> help.search("regression")
```

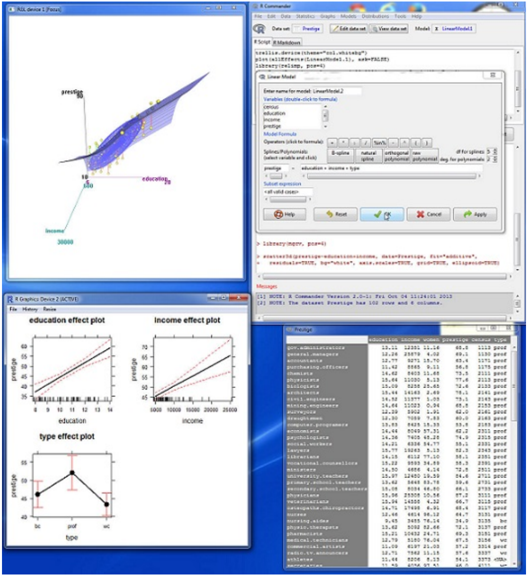
But you'll get most help online

- Quick-R
- Stack overflow
- Stackexchange
- IDRE UCLA
- Just google your problem

Some R interfaces (emacs)

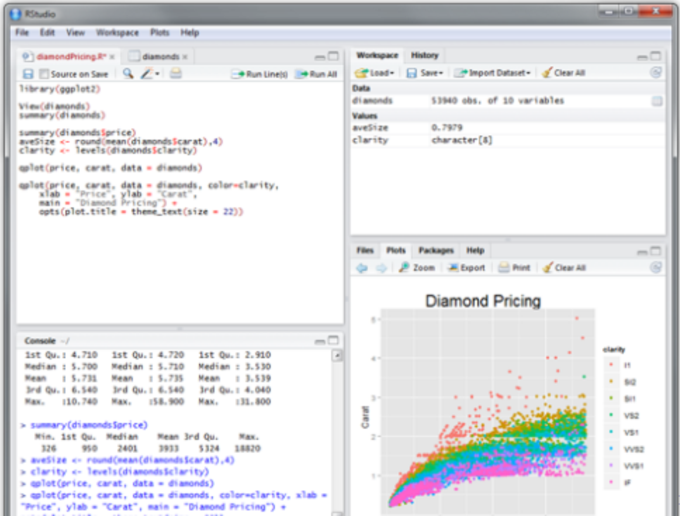


R Commander


education effect plot
income effect plot
type effect plot
prestige
education
income
type
Linear Model
Messages

RStudio

- RStudio is an environment for data analysis **and** report generation
- Just open up a new world!



Your first R session

- anything you create in R is an object (it's similar to other programming langaugues such as Python)
- objects can be numbers, formulas, collections of objects, names, etc.

```
> # A comment line is started by `#'.
> # Everything after `#' is ignored by R.
> a <- 5
```

- this command creates an object named **a** to which you've given the value **5**
- the <- is the assignment operator (= is equivalent)

```
> a
```

```
## [1] 5
```

Not only numbers

```
# To assign non-numeric values (characters, character stri
#R needs quotation marks to know what we're doing
d <- "a"
d

#but if we make a mistake and don't use "quotes", then:
d <- a
a

# we can assign any name to objects
e <- "pepitu"
e

# even longer names
f <- "the_perfect_song"
g <- "Justin Beaver is my God"
```

Listing and deleting objects

```
#This function call lists all the objects currenly in memor

ls()

#This function removes a particular object from the working
#environment

rm(a)
ls()

# what happens when you call the object "a" now?
a
```

Object manipulation

```
#Get a back and create another numeric object, called b,
#and check its value
a <- 5
b <- 4
b

#We can now do stuff with those values
a + b

# We can also assign a name to this operation
c <- a + b
c

#But what happens when we try to add a number and a charac
a + d
```

Functions

- The `ls()` command we used before is a **function**: it lists all objects in our workspace
- R is a largely functional programming: we use functions all the time, and most of them have been already created by others (we'll see plenty)
- We can create our own functions

```
suma <- function(x,y){result <- x + y; return(result)}
```

- function named `suma` which accepts two parameters, `x` and `y`
- within the body of the function ($\{\dots\}$) we add the two parameters together and call that operation `result`
- finally, we include an already existing R function called `return()` to give us the `result`

Operations with functions

```
#What's there?
ls()
suma

#Simple operations
suma(6,7)
suma(a,b)
h <- suma(a,b)

#Try this
suma(a,6)

#Whappens here?
suma(d,6)

#Finally, try this. What happens? Why?
suma(2,3,4)
```

Functions

- Functions are always created the same way
  - name
  - instructions
  - **function()** command
  - parametes inside **function()**
  - further actions

```
#Another function
divit <- function(x,y){result <- x / y; return(result)}
divit(9,3)
divit(3,9)
divit(3,x=9) #What do you expect to happen?
```

Exercises and break!

```r
#First type this
rm(list=ls())
```

1. Create an object `a` with value 5
2. Create an object `b` with value 7
3. Add `a` to `b`
4. Divide `b` by `a`
5. Divide 7 by 5 without first assigning 7 and 5 to objects
6. Create a function called `this()` which has one parameter `x`, and simply returns the value of `x` sent to it.
7. Run `this()` sending a numeric value to it.
8. Create a function `mulp()` which multiplicates two numbers.
9. Send 4 and 8 to `mulp()`
10. Send `a` and `b` to `mulp()`
11. Nest the output of `this()` which has an input of 6 with one of the inputs of `mulp()`. Let the other input of `mulp()` be 7.

Exercise solutions

```
#First type this
rm(list=ls())

# 1. Create an object a with value 5
a <- 5
# 2. Create an object b with value 7
b <- 7
# 3. Add  a to b
a + b
```

```
## [1] 12
```

```
# 4. Divide b by a
b/a
```

```
## [1] 1.4
```

Exercise solutions

```
# 5. Divide 7 by 5 without first assigning 7 and 5 to obje
7/5
```

```
## [1] 1.4
```

```
# 6. Create a function called this() which has one paramete
# and simply returns the value of x sent to it.

# 7. Run this() sending a numeric value to it.
```

Exercise solutions

```
# 5. Divide 7 by 5 without first assigning 7 and 5 to obje
7/5
```

```
## [1] 1.4
```

```
# 6. Create a function called this() which has one paramet
# and simply returns the value of x sent to it.
this <- function(x){result <- x; return(result)}

# 7. Run this() sending a numeric value to it.
this(8)
```

```
## [1] 8
```

Exercise solutions

```
# 8. Create a function mulp() which multiplicates two numbe
# 9. Send 4 and 8 to mulp()
# 10. Send a and b to mulp()
# 11. Nest the output of this() which has an input of 6 wi
# of the inputs of mulp(). Let the other input of mulp() be
```

Exercise solutions

```r
# 8. Create a function mulp() which multiplicates two numbe
mulp <- function(x,y){result <- x*y; return(result)}

# 9. Send 4 and 8 to mulp()
mulp(8,3)
```

```
## [1] 24
```

```r
# 10. Send a and b to mulp()
mulp(a,b)
```

```
## [1] 35
```

```r
# 11. Nest the output of this() which has an input of 6 wi
# of the inputs of mulp(). Let the other input of mulp() be
```

Exercise solutions

```
# 8. Create a function mulp() which multiplicates two numbe
mulp <- function(x,y){result <- x*y; return(result)}

# 9. Send 4 and 8 to mulp()
# 10. Send a and b to mulp()
mulp(a,b)
```

```
## [1] 35
```

```
# 11. Nest the output of this() which has an input of 6 wi
# of the inputs of mulp(). Let the other input of mulp() be
mulp(this(5),7)
```

```
## [1] 35
```

The workspace

- **workspace** is all the objects created and used by us in a session
- when we exit R we are asked whether we want to save our workspace
- if **yes**, that workspace will be loaded automatically the next session
- it will be located in our current **working directory**

```
getwd()
```

```
## [1] "F:/R_course/reports"
```

```
setwd()
```

Basic operations and objects

```
5+2 #sum
5-2 #difference
5*2 #multiplication
5/2 #division
5^2 #exponentiation
(5+2)*(8/4) #concatenation of operations
```

- These operations, though, cannot only be carried out on single integers
- There are numerous other types of data objects in R
  - vectors
  - matrices
  - data frames
  - lists

Vectors

- A vector is just a **concatenation** of data elements
- These elements may be numbers, characters, or whatever
- To build vectors in R we use the **concatenate** function `c()`

```
a <- c(1,2,3,4)
a <- 1:4
a + 4
b <- a/4
a + b
c <- c(1,2,3)
a*c
d <- c(1,2,3,"k") #what happens here?
```

- The number of elements of a vector is its **length**

```
length(a)
```

Dereferencing and lenghts

- Sometimes we want to access only parts of vectors or arrays of elements

```
a <- 1:8
a[2]
```

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

Dereferencing and lenghts

- Sometimes we want to access only parts of vectors or arrays of elements

```
a <- 1:8
a[2]
```

```
a[c(2,3)]
```

```
b <- c(3,4)
a[b]
```

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

Dereferencing and lenghts

- Sometimes we want to access only parts of vectors or arrays of elements

```
a <- 1:8
a[2]
a[c(2,3)]
b <- c(3,4)
a[b]
```

- The square brackets [] for dereferencing works in other types of R objects that contain arrays of data such as matrices and data frames

Matrices

- A matrix is just a collection of vectors

```
a <- 1:8
d <- matrix(a, ncol=2, byrow=TRUE)
d
```

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

- `matrix()` just converts vector a into a matrix with 2 columns (`ncol=2`) and fills it by rows (`byrow=TRUE`)
- What happens if we set `byrow=FALSE` ?
- Try also `ncol=3`

Indexing and derefereing matrices

- What should we do to select a specific element of a matrix?

```
d[2,1]
```

- It refers to the 2nd row of the 1st column of the matrix
- In R (and almost everywhere), we first say rows and then columns
  - A 2x3 matrix is something like

```
matrix(2,nrow=2,ncol=3)
```

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

Indexing and derefereing matrices

- We can use vectors for dereference matrices

```
d[c(1,3), 1] #what does this mean?
```

- We can use variables and use them to dereference matrices

```
e <- 2
d[1,e]
```

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

- It's useful to be able to extract whole columns or rows from matrices

```
d[,2]
d[3,]
```

Negative indices

- Negative indices act as an exclusion list for a data object

```
a[-5]
a[-b] #which is equivalent to
a[-c(3,4)]
```

- Negative indices can also be used with matrices

```
d[-3,]
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    7    8
```

```
d[,-1]
```

Data frames

- They are a variant of matrices for data storage, with variables as columns
- Variables can be either numeric, factors, or characters
- We can construct a `data.frame` from `d`

```
f <- data.frame(d)
f
```

```
##   X1 X2
## 1  1  2
## 2  3  4
## 3  5  6
## 4  7  8
```

Dereferencing data frames

- We can use square brackets aswell, but not only that

```
f[3,2] #Regular use of square brackets
f[, 2] #2nd column
f[1, ] #1st row
```

- We now can call columns (variables) by their name!

```
f$X1
```

```
## [1] 1 3 5 7
```

```
f$X2[3]
```

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

Exploring a data frame (Exercise)

- Data frames have different characteristics we want to know about
- What information do these commands give you about the `f` dataframe?

```
length(f)
nrow(f)
ncol(f)
dim(f)
```

Exploring a data frame

```r
length(f) #how many variables
```

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

```r
nrow(f)    #how many rows
```

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

```r
ncol(f)    #how many columns (=length(f))
```

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

```r
dim(f)     #dimensions of the data frame
```

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

Statistical functions

```
mean(f$X1) #mean
sd(f$X2)  #standard deviation
var(f$X2)  #variance
median(f$X1) #median
min(x)
max(x)
```

Exercises

- Get the pdf file called `Exercise_1` from the course folder
- Complete all the tasks
- Start now!