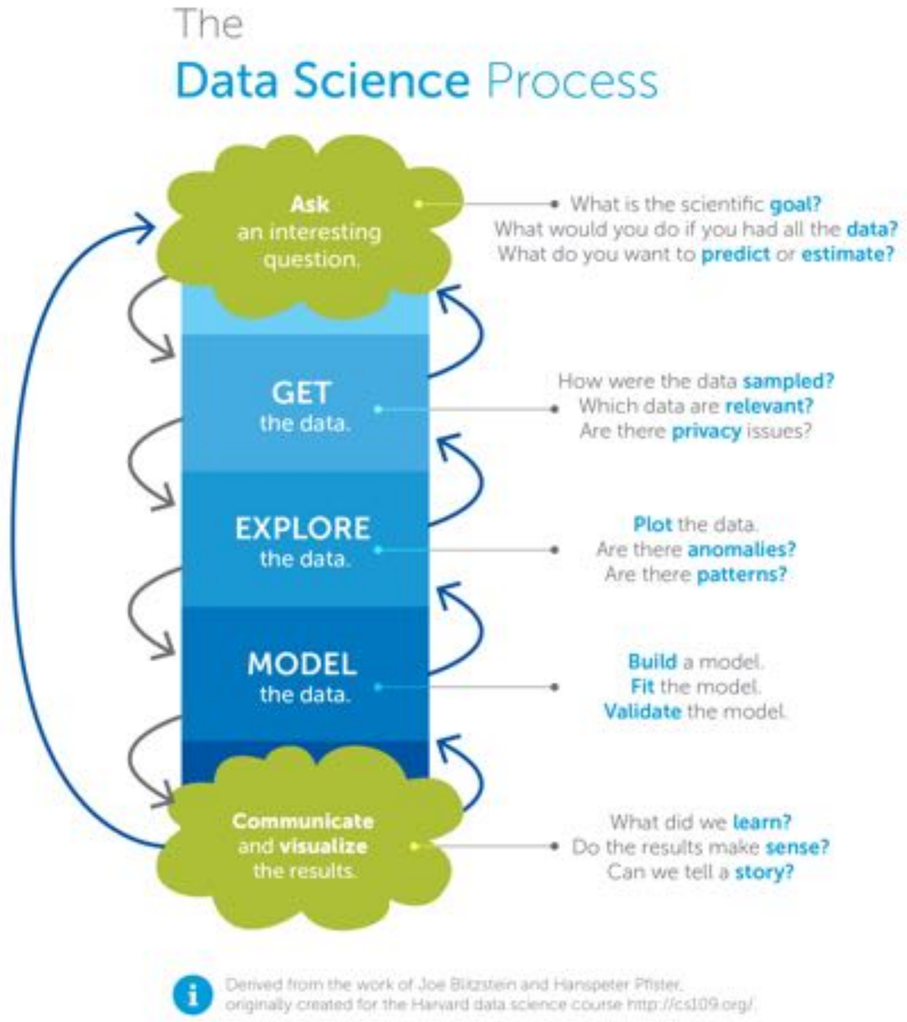
Session 2

(Customer Analytics)

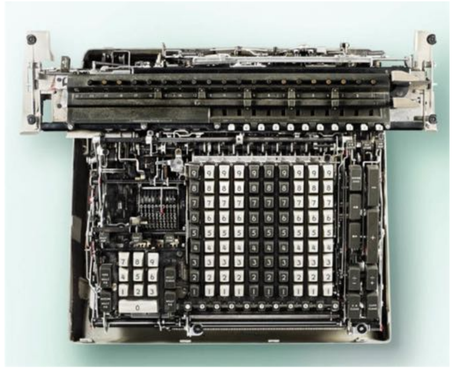
Josep Curto
Professor, IE Business School | CEO | DS



R can be used:

- Data Import (access any type of data)
- Data Wrangling (tidy, prepare and sample)
- Exploratory Data Analysis (plot, anomalies & patterns)
- Analysis (build, fit, validate)
- Communicate

What is R?



R-intro-session1.R*

Source on Save

```
1  2+3 #simple math
2
3  x<- 34.56 + 23.15 + log(456)
4  print(x)
5
6  x<- c(1,2,4)
7  y<- c(5,6,7)
8
9  print(x+y)
10
11
12
```

What is R?

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponential
%%	Modulus
%/%	Integer division

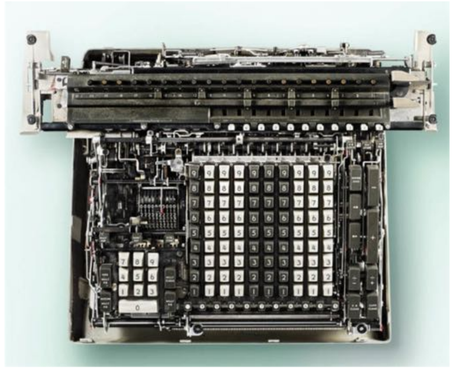
What is R?

f(argument1, argument2, ...)

```r
# log to the base e of 2
log(2)

# antilog of 2
exp(2)

# log to base 2 of 3
log(3,2)

# log to base 10 of 2
log10(2)

# square root of 2
sqrt(2)

# !5
factorial(4)
```

```r
# largest interger smaller than 2
floor(2)

# smallest integer greater than 6
ceiling(6)

# round 3.14159 to three digits
round(3.14159, digits=3)

# cosine of 3
cos(3)

# sine of 3
sin(3)

# tangent of 3
tan(3)
```

What is R?



- Calculate the value of 1 euro after 1,3,5,8,10 years at a 5% compound rate

- FV = value ^(years*rate)

```
# Our first R code

years <- c(1,3,5,8,10)
result <- exp(0.05*years)
print(result)

# We can improve the code

rate <- 0.05
result <- exp(rate*years)
print(result)
```

What is R?

- Structured, standardized unit of:
  - R code
  - documentation
  - data
  - external code

dim(available.packages())

A Swiss Army Knife with its thousands of packages

- What packages do I have installed?
  - (.packages())

Handling packages

- Install with install.packages(name)
  - install.packages("ggplot2")
- Uninstall with remove.packages(name)
  - remove.packages("ggplot2")
- Load with library(*name*)
  - library("ggplot2")
  - You can also use require() but it is a bad programming practice
- Package-level help:
  - library(help=*name*)
- Unload with detach(package:*name*)
  - detach("package:ggplot2", unload=TRUE)

- See the packages tab in Rstudio
- Checkmark to load
- Some are already loaded!!
- Click on the name for help

Exploring a dataset

- data(name)
  - data(“diamonds”)
    - Loads the dataset in memory
    - data() shows datasets available

Filter

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

  - View(diamonds)
  - head(diamonds)
  - tail(diamonds)

R-style Recommendations

Indentation

Indent lines with two spaces, not tabs. If code is inside parentheses, indent to the innermost parentheses

Spacing

Use only single spaces. Add spaces between binary operators and operands. Do not add spaces between a function name and the argument list. Add a single space between items in a list, after each comma

Blocks

Don't place an opening brace ("{") on its own line. Do place a closing brace ("}") on its own line. Indent inner blocks (by two spaces)

Semicolons

Omit semicolons at the end of lines when they are optional

Naming

Name objects with lowercase words, separated by periods. For function names, capitalize the name of each word that is joined together, with no periods. Try to make function names verbs.

Assignment

Use <-, not = for assignment statements

Values – value assignment
Varibles - dynamically type variables

```r
# Values assigment
a <- 5
b <- a
a <- 7
print(b)
```

Unlike other languages, R variables do not need to be declared and typed

- Assigning a sequence of numbers to x forces x to be a numeric vector
- Given x, executing class(x) reports the class. This indicates which functions can be used on x
- Everything is a vector

```r
# Dynamically type variables
x <- 5
class(x)
x <- "hello"
class(x)
x <- 5L
class(x)
x <- TRUE
class(x)
x[1]
class(x[1])
```

Variables – Mode vs. Class

In R, every object has a *mode*, which indicates how it is stored in memory: as a number, as a character string, as a list of pointers to other objects, as a function, and so forth:

Object	Example	Mode
Number	`3.1415`	numeric
Vector of numbers	`c(2.7.182, 3.1415)`	numeric
Character string	`"Moe"`	character
Vector of character strings	`c("Moe", "Larry", "Curly")`	character
Factor	`factor(c("NY", "CA", "IL"))`	numeric
List	`list("Moe", "Larry", "Curly")`	list
Data frame	`data.frame(x=1:3, y=c("NY", "CA", "IL"))`	list
Function	`print`	function

Variables – Mode vs. Class

Additionally every object also has a *class*, which defines its abstract type.

- A single number could represent many different things: a distance, a point in time, a weight
- All those objects have a mode of "numeric" because they are stored as a number; but they could have different classes to indicate their interpretation.

```
# Modes vs Class
crashCourseDate <- as.Date("2016-10-19")
typeof(crashCourseDate)
class(crashCourseDate)
```

Variables – Testing and Changing types

Type	Testing	Coercing
Array	is.array	as.array
Character	is.character	as.character
Complex	is.complex	as.complex
Dataframe	is.data.frame	as.data.frame
Double	is.double	as.double
Factor	is.factor	as.factor
List	is.list	as.list
Logical	is.logical	as.logical
Matrix	is.matrix	as.matrix
Numeric	is.numeric	as.numeric
Raw	is.raw	as.raw
Time series (ts)	is.ts	as.ts
Vector	is.vector	as.vector

Data Objects in R

- **vector** - a sequence of numbers or characters, or higher- dimensional arrays like matrices
- **list** - a collection of objects that may themselves be complicated
- **factor** - a sequence assigning a category to each index
- **data.frame** - a table-like structure (experimental results often collected in this form)
- **environment** - hash table. A collection of key-value pairs

Data Objects in R - Vectors

- Vectors can only contain entries of the same type: numeric or character; you can't mix them.
  - Note that characters should be surrounded by " ".

```
#  Vectors
x <- c("a","b","c")
length(x)
```

- The most basic way to create a vector is with c(x1, . . . , xn), and it works for characters and numbers alike.

Data Objects in R - Vectors

Vector variables can and should be named to allow for future indexing

```
x <- c(s1=0.5,s2=0.8,s3=0.1)
x
sort(x)
names(x)

y <- c(0.5,0.8,0.1)
names(y) <- c("s1","s2","s3")
y
```

Data Objects in R - Vectors

var, mean, summary, min and max are useful functions for working with vectors ,

```
> x
[1] -1.6267904 -1.2337142  0.2257716 -0.6470700

> var(x)
[1] 0.6485375

> max(x)
[1] 0.2257716

> min(x)
[1] -1.62679

> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.6270 -1.3320 -0.9404 -0.8205 -0.4289  0.2258
```

Data Objects in R – Scalars are vectors

Scalars are vectors of length 1

```
> x<-34
> x[1]
[1] 34
> x[2]
[1] NA
> length(x)
[1] 1
```

Vectors – Generating sequences

- c – concatenate
- seq, :, and rep
- vector, numeric, character, etc

```r
# Vector creation
a <- c(1,2,3)
a
b <- seq(0,10, by=2)
b
c <- rep(b, times=5)
c
```

Vectors - Logical

Operators	Meaning
<	Less than
<=	Less than or equal to
>	More than
>=	More than or equal to
==	Equal to
!=	Not equal to
!a	Not a
a|b	a or b
a&b	a and b
isTRUE(a)	Test if a is true

Special values - NA

In R, the NA values are used to represent missing values. (NA stands for “not available.”) You may encounter NA values in text loaded into R (to represent missing values) or in data loaded from databases (to replace NULL values).

```
> x <- c(1:3,NA)
> is.na(x)
[1] FALSE FALSE FALSE  TRUE
```

Special values – Inf & -Inf

If a computation results in a number that is too big, R will return Inf for a positive number and -Inf for a negative number (meaning positive and negative infinity, respectively):

```
> 1/0
[1] Inf
> 2^35670
[1] Inf
> -3^4567789
[1] -Inf
```

Special values – Nana

If the result of a statement doesn't make sense Not a Number will be the result

```
> log(-34)
[1] NaN
Warning message:
In log(-34) : Se han producido NaNs
> 0/0
[1] NaN
```

Special values – Null

NULL represents the lack of existence (vs. Na or NaN)

```
> x<-c(8,2,3,NULL)
> x
[1] 8 2 3
> x[2]<- NULL
Error in x[2] <- NULL : replacement has length zero
> x
[1] 8 2 3
> x[2]<-NaN
> x
[1]   8 NaN   3
```

Vectors – Subsetting

- An element of a vector v is assigned an index by its position in the sequence, starting with 1
  - The basic function for subsetting is [ ]
  - x[1] is the first element, x[length(x)] is the last.

```
> x <- c("a","b","c","d","e","f")
> x[2]
[1] "b"
> x[length(x)]
[1] "f"
```

Vectors – Adding an element

```
> x <- c(1,2,3)
> x
[1] 1 2 3
> x[3] <- 45
> x
[1]  1  2 45
> x[7] <- 48
> x
[1]  1  2 45 NA NA NA 48
> append(x, c(1,23))
[1]  1  2 45 NA NA NA 48  1 23
> y <- c(45,23,32)
> x <- c(x,y)
> x
 [1]  1  2 45 NA NA NA 48 45 23 32
> x <- append(x, 34, after=4)
> x
 [1]  1  2 45 NA 34 NA NA 48 45 23 32
```

Exercice – Vector Manipulation

- Create a variable and store the number 25
- Calculate the square root of 25
- Create a vector with the first 100 numbers
- Create a vector to hold the data of a student (grade, course and class, all of them numeric)
    - access the element with the name “grade”
    - Change the data to character data
- Create a vector with the first 50 numbers
    - Create another vector withe the numbers 51-100
    - Add them together
    - Multiply them
    - Join both of them in a larger vector

Useful functions

Keeping track of your work

```
# Save the commands used during the session

savehistory(file="mylog.Rhistory")

# Load the commands used in a previous session

loadhistory(file="mylog.Rhistory")

# Display the last 25 commands

history()

# You can read mylog.Rhistory with any word processor. Notice that the file has to have the extension
   *.Rhistory
```

Useful functions

Exploring the workspace

```
objects()        # Lists the objects in the workspace
ls()             # Same as objects()
remove()         # Remove objects from the workspace
rm(list=ls())    #clearing memory space
detach(package:ABC)    # Detached packages when no longer need them
search()         # Shows the loaded packages
library()        # Shows the installed packages
dir()            # show files in the working directory
```

Lists

- An ordered collection of heterogenous variables

```
> student <-list(name="Josep",surname="Curto Díaz",
+                male=TRUE, age=39,grades=list(5,6,7,8))
> student[1]
$name
[1] "Josep"

> student$age
[1] 39
> student
$name
[1] "Josep"

$surname
[1] "Curto Díaz"

$male
[1] TRUE

$age
[1] 39

$grades
$grades[[1]]
[1] 5

$grades[[2]]
[1] 6

$grades[[3]]
[1] 7

$grades[[4]]
[1] 8
```

Matrices

A multi-dimension object that keeps the underlying class

```
> m <- matrix(data=1:12, nrow=4, ncol=3, dimnames = list(c("r1","r2","r3","r4"),c("c1","c2"
,"c3")), byrow=TRUE)
> m
   c1 c2 c3
r1  1  2  3
r2  4  5  6
r3  7  8  9
r4 10 11 12
```

Matrices - indexing

```
> m[1,3]
[1] 3
> m[1,]
c1 c2 c3
 1  2  3
> m[,3]
r1 r2 r3 r4
 3  6  9 12
> m["r1","c2"]
[1] 2
> m[c(1:2),c(1:3)]
   c1 c2 c3
r1  1  2  3
r2  4  5  6
```

Matrices – adding elements

```
> m <- rbind(m, r5=c(13,14,15))
> m
   c1 c2 c3
r1  1  2  3
r2  4  5  6
r3  7  8  9
r4 10 11 12
r5 13 14 15
```

```
> m <- cbind(m, c4=c(34,23,12,11,67))
> m
   c1 c2 c3 c4
r1  1  2  3 34
r2  4  5  6 23
r3  7  8  9 12
r4 10 11 12 11
r5 13 14 15 67
```

Matrices – editing values

```
> rownames(m)<-c("row1","row2","row3","row4","row5")
> colnames(m)<-c("col1","col2","col3","col4")
> m
     col1 col2 col3 col4
row1    1    2    3   34
row2    4    5    6   23
row3    7    8    9   12
row4   10   11   12   11
row5   13   14   15   67
```

Matrices – combining

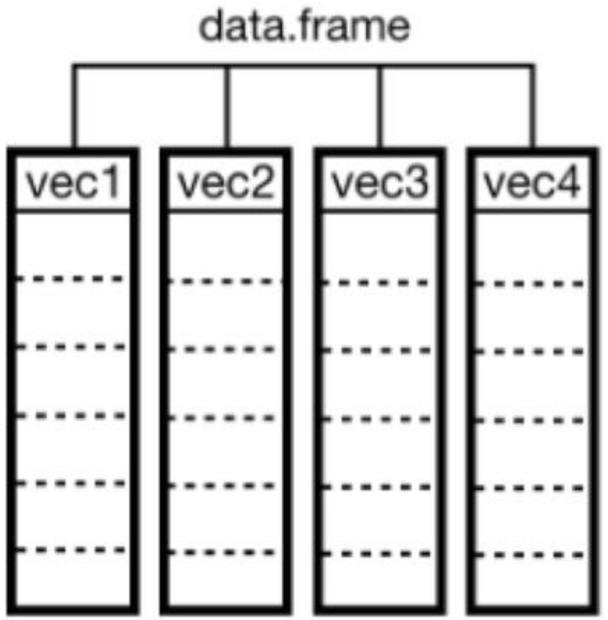
```
> m2 <- matrix(data=13:32, nrow=5, ncol=4,
+            dimnames=list(c("r1", "r2", "r3", "r4","r5"), c("c1", "c2", "c3","c4")),byrow=TRUE)
> m2
   c1 c2 c3 c4
r1 13 14 15 16
r2 17 18 19 20
r3 21 22 23 24
r4 25 26 27 28
r5 29 30 31 32
> cbind(m,m2)
     col1 col2 col3 col4 c1 c2 c3 c4
row1    1    2    3   34 13 14 15 16
row2    4    5    6   23 17 18 19 20
row3    7    8    9   12 21 22 23 24
row4   10   11   12   11 25 26 27 28
row5   13   14   15   67 29 30 31 32
> rbind(m,m2)
     col1 col2 col3 col4
row1    1    2    3   34
row2    4    5    6   23
row3    7    8    9   12
row4   10   11   12   11
row5   13   14   15   67
r1     13   14   15   16
r2     17   18   19   20
r3     21   22   23   24
r4     25   26   27   28
r5     29   30   31   32
```

Matrices – operations

```
> x <- matrix(data=1:12, nrow=4, ncol=3, byrow=TR
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> y[4] <- 4
> dim(y) <- c(4,1)
> y
     [,1]
[1,]   45
[2,]   23
[3,]   32
[4,]    4
> t(x)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> diag(4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

Dataframe – collection of vectors

Tabular (rectangular) data structure, which means that it has rows and columns. It is not implemented by a matrix, however. Rather, a data frame is a list:



Exercise – Dataframe manipulation

- Load the test dataset
  - load("dataTB.rdatq")
- What is the median of the final price?
- What is the relationship between final price and user price?
- What is the event with the highest final price?

Session Wrap-up

Q&A


Thank you!
Josep Curto | @josepcurto | jcurto@faculty.ie.edu