Scaling with R

Frédéric Logé
frederic.logemunerel@gmail.com

2025-10-17

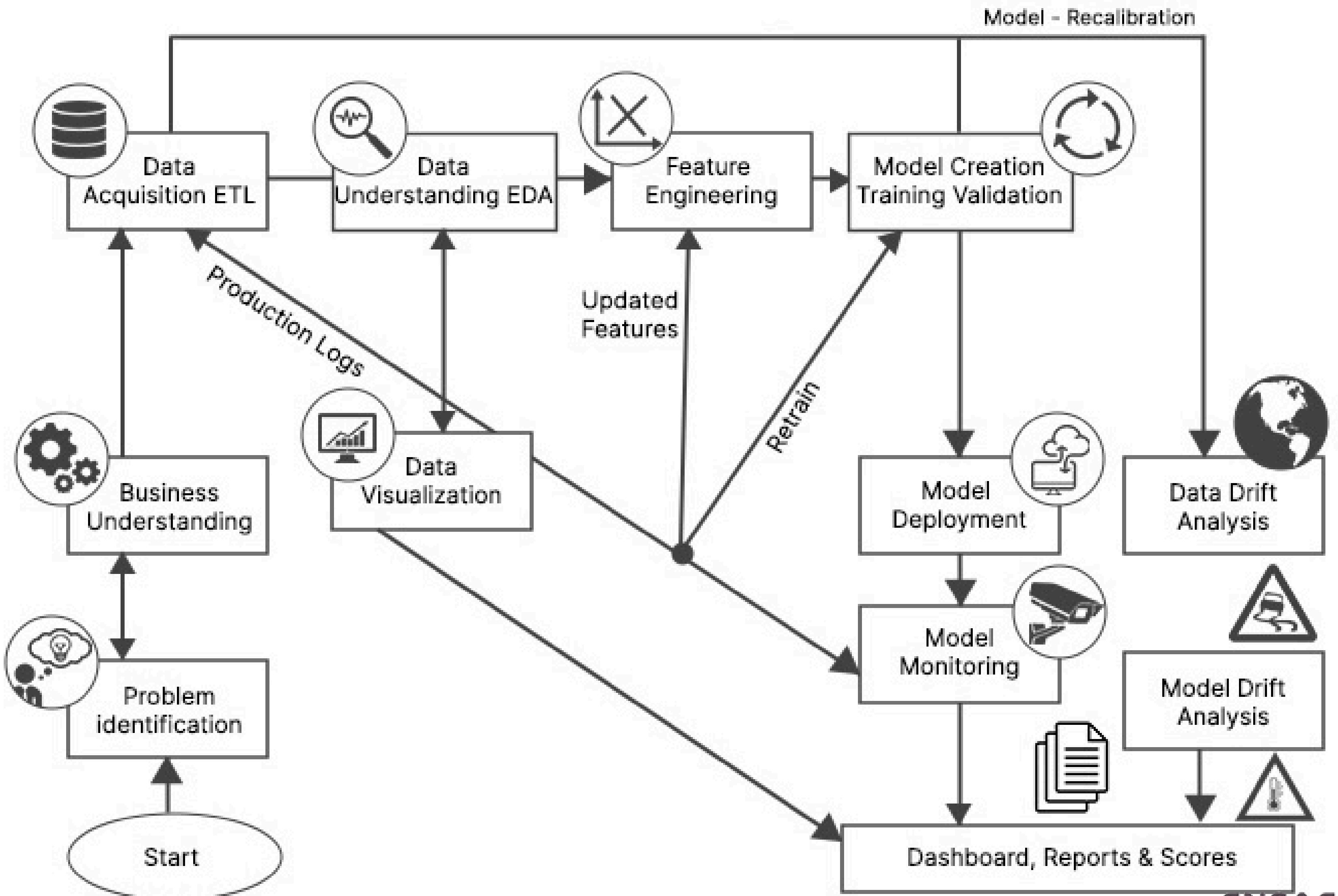
ENSAE-ENSAI
Formation continue
(Cepe)

Table of contents

ENSAE-ENSAI
Formation continue
(Cepe)

Introduction

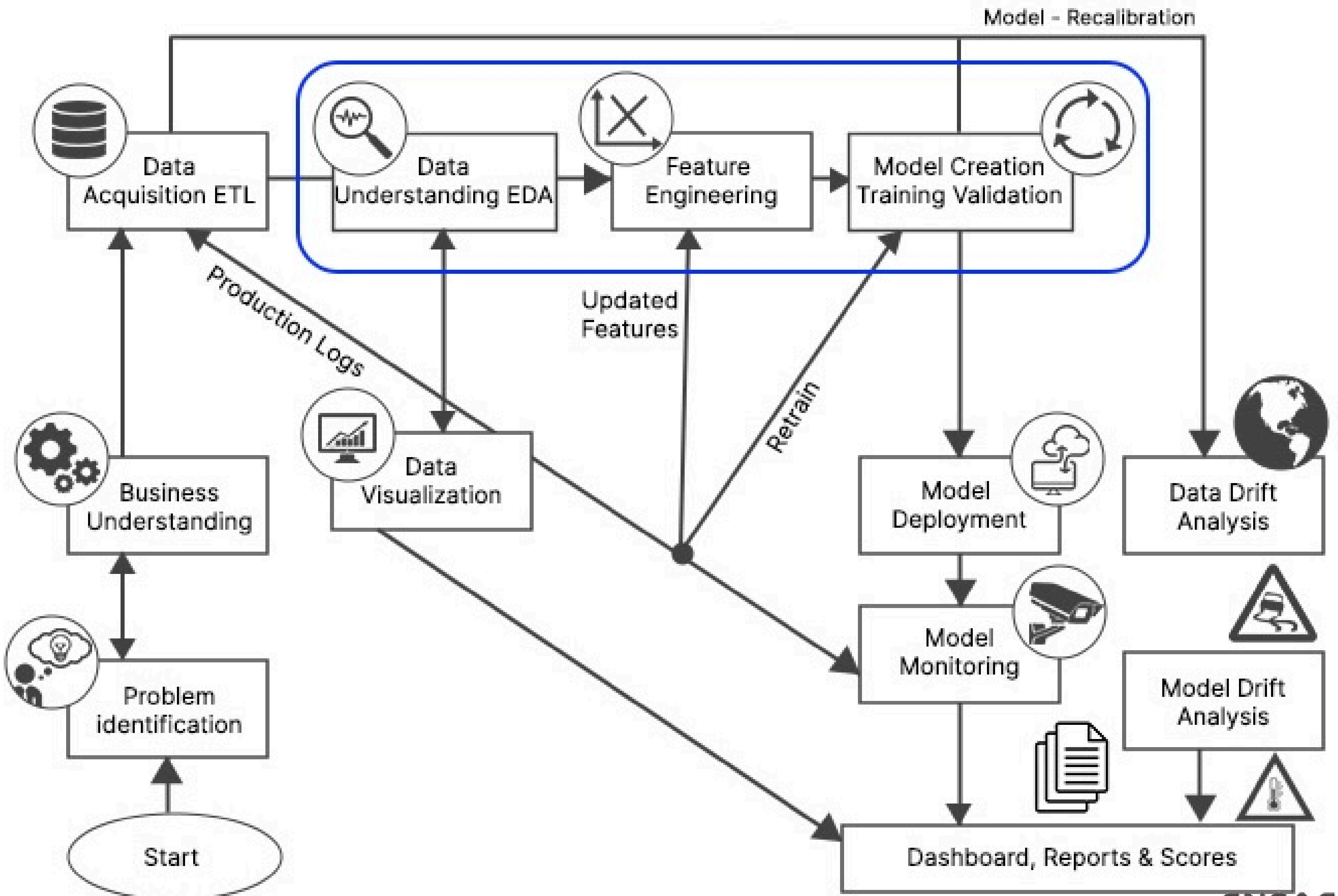
Data Science Lifecycle



Data Science Life cycle, source

ENSAE-ENSAI Formation continue (Cepe)

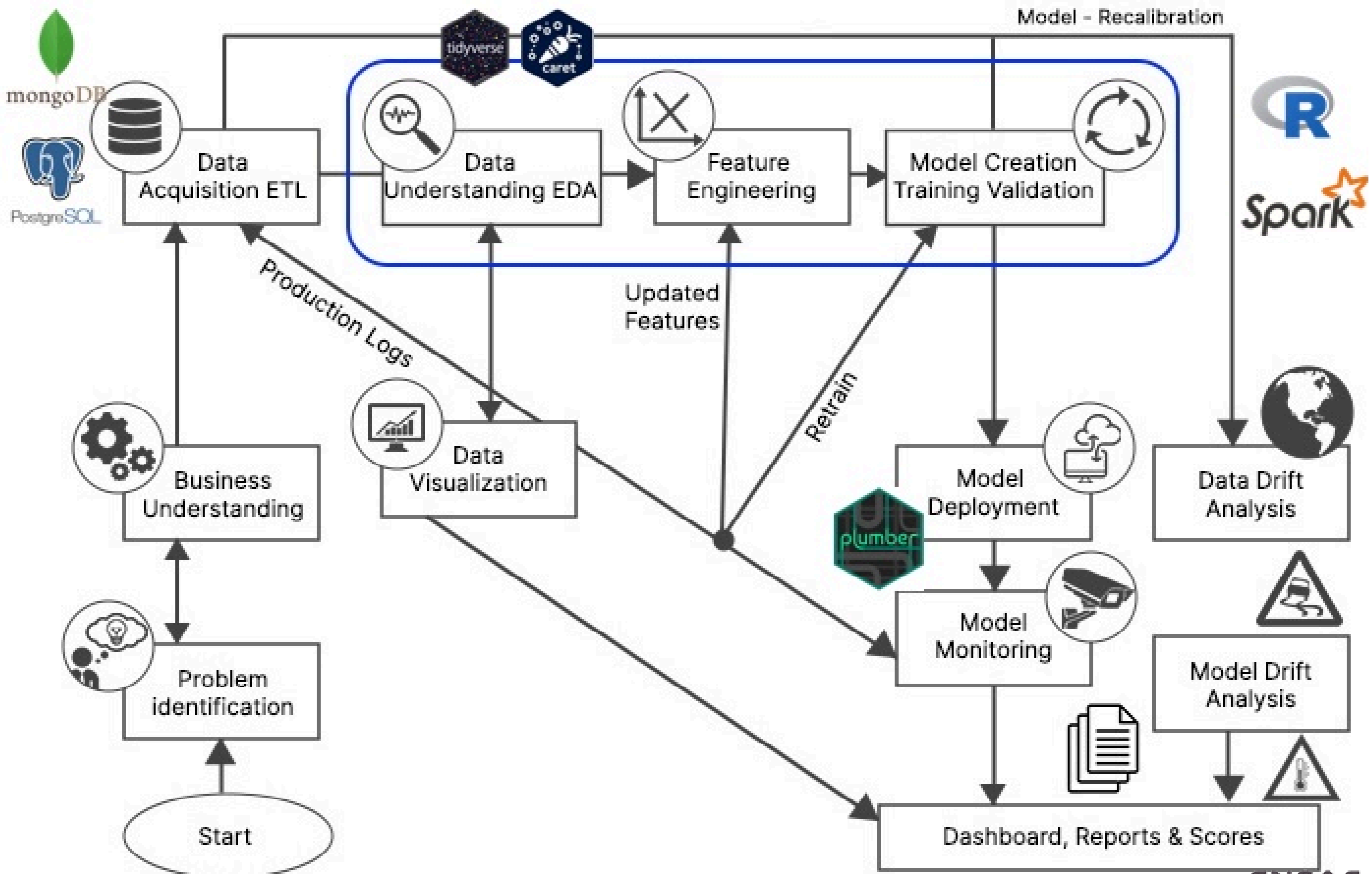
What are we going to work on today ?



Data Science Life cycle, source

ENSAE-ENSAI Formation continue (Cepe)

What are the tools involved ?

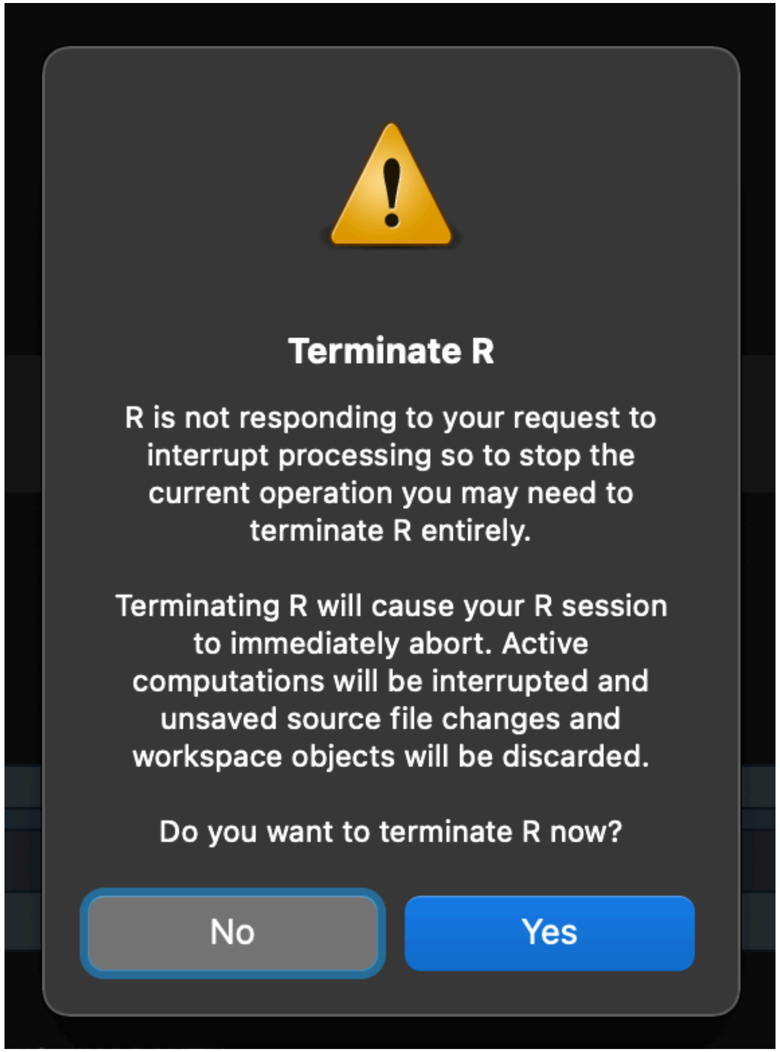


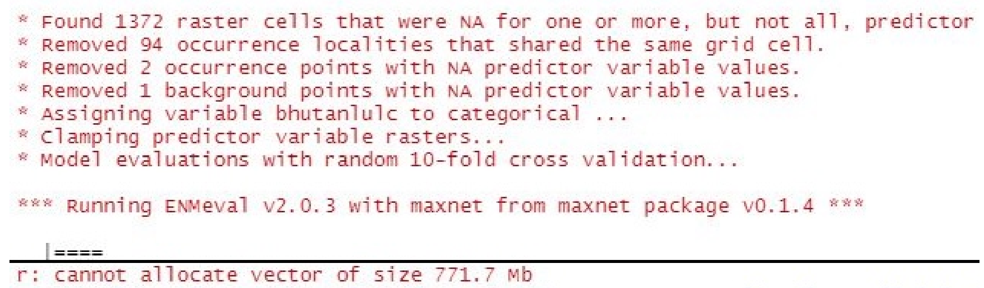
Data Science Life cycle, source

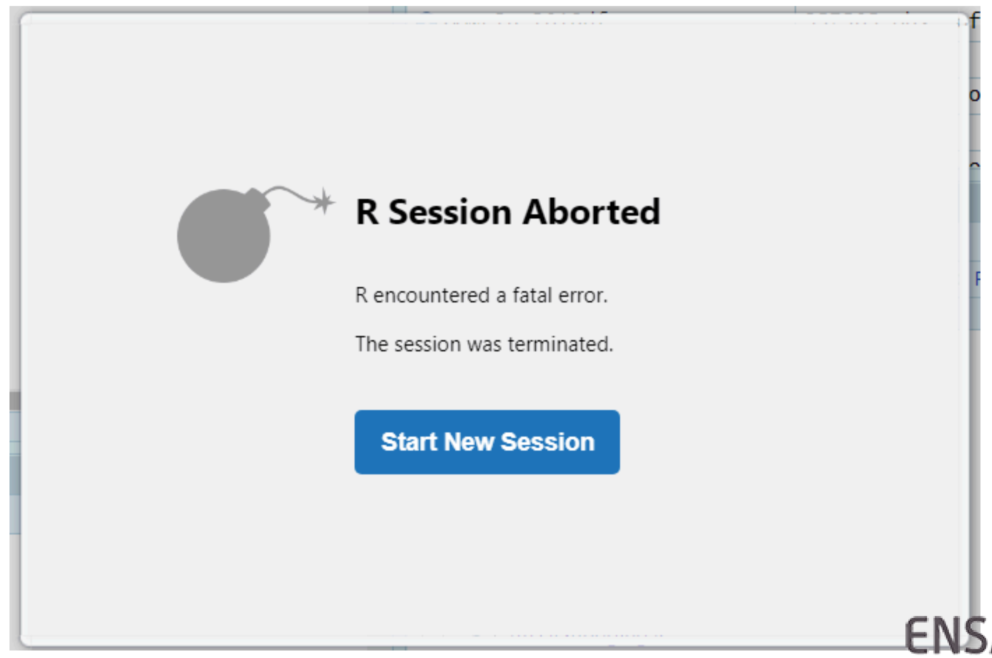
ENSAE-ENSAI
Formation continue
(Cepe)

Common issues with large datasets

Time, Memory, Crash







But what can we do ?

A not exhaustive list of what we may do:

- **Optimize your R code** e.g. remove unnecessary loops, avoid data copy, loading unnecessary packages, etc, `Rcpp`
- **Rely on (most efficient) R packages** e.g. `dplyr`, `data.table`, `arrow`
- **Run code in parallel** e.g. `future` exploit hardware and distribute the work
- **Upgrade session's available memory** e.g. change RStudio config, get hardware update, setup virtual machine with higher memory config
- **Delegate treatment** e.g. `DBI`, `sparklyr`, `h2o` to perform operations with an engine more efficient than R
- **Work on data samples** e.g. active learning
- Breakdown tasks in your data science life cycle: **you cannot do everything in R**.

ENSAE-ENSAI Formation continue (Cepe)

Writing efficient code

ENSAE-ENSAI
Formation continue
(Cepe)

Best R practice recommendations

- Cleanliness & tidyness: avoid data copy, avoid garbage names throughout code, treat your RAM with kindness
- Comments are mandatory, even if variable names are explicit
- Work under RProject
- If you wish to build something that should last (used by others, robust etc), developing as an R Package is mandatory. Not suitable when doing tutorials and trying a million different things though :/

ENSAE-ENSAI Formation continue (Cepe)

Timing

When you are writing an R function, measuring it’s execution time is good practice.

```
# define a function
foo <- function(){ return(sum(1:1e6)) }

# measure execution of the function, once
# user: actual CPU time for the process
# system: any indirect operation due to the process: I/O of files, GC, memory allocation, ...
# elapsed: total elapsed time
system.time({ foo() })
```

```
   user  system elapsed 
  0.000   0.000   0.001 
```

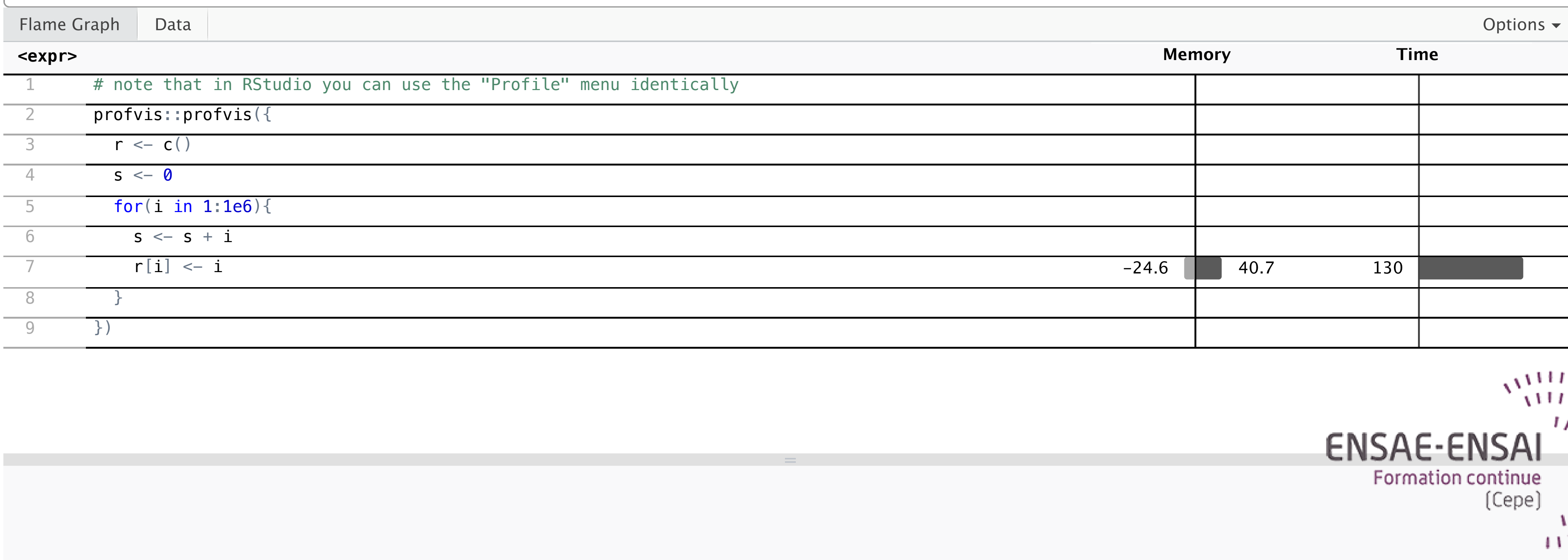
```
# repeat 50 times the function to get some statistics
microbenchmark::microbenchmark({ foo() }, times = 50)
```

```
Unit: nanoseconds
          expr min  lq     mean median  uq    max neval
 {     foo() } 164 164 13479.98    205 205 663831    50
```

Profiling code

When your function (or general code) contains several instructions, profiling it allows you to see where exactly things go wrong. Here's a vanilla example:

```r
# note that in RStudio you can use the "Profile" menu identically
profvis::profvis({
  r <- c()
  s <- 0
  for(i in 1:1e6){
    s <- s + i
    r[i] <- i
  }
})
```




<GC>
r[i] <- i
doTryCatch
tryCatchOne
tryCatchList
tryCatchList
Sample Interval: 10ms
130ms

Exercise: Moving Average

Moving Average

Input: x, numerical vector of length N

Output: y, numerical vector of same length, where $y_i := (x_i - 1) + x_i + (x_i + 1)/3$ if $i > 1$ and $i < N$, otherwise y_i is NA.

Propose some implementations of this function and use the timing functions seen previously to time yourselves.

Solution(?)

```r
x <- rnorm(n=1e6)
n <- length(x)

ma_0 <- function(x, n){
  y <- rep(NA, n)
  for(i in (2:(n-1))){
    y[i] <- (x[i-1]+x[i]+x[i+1])/3
  }
  return(y)
}

ma_1 <- function(x, n){
  return((c(NA, x[1:(n-1)]) + x + c(x[2:n], NA))/3)
}

microbenchmark::microbenchmark(ma_0(x=x, n=n), ma_1(x=x, n=n))
```

```
Unit: milliseconds
               expr      min       lq     mean   median       uq      max neval
 ma_0(x = x, n = n) 70.60356 74.18450 77.02036 75.49027 77.47805 119.4600   100
 ma_1(x = x, n = n) 10.97320 13.21899 16.20814 14.16101 15.40983  45.8453   100
```

Vector operations are a must-use resource of R.

ENSAE-ENSAI
Formation continue
(Cepe)

Why still create our own R functions ?

If it exists in R and ruins decently well, then why rebuild it ?

There are many cases were we might not find our pick:

- Data simulation purposes
- Statistical metrics
- Optimization functions

…

- Because we don't know any other language 😭

Exercise: Simulation Example

Simulation

Consider the following dynamic system:

- x_0 is in $[0; 1]$
- λ is in $[0; 4]$

Then, for all $t \geq 1$:

$x_t = \lambda * x_{t-1} * (1 - x_{t-1})$.

Step 1. Code an R function which takes as input (x_0, λ, n) and returns output x_n.

Step 2. Run the function for a uniform grid of (x_0, λ), the size of your choosing.

Solution to step 1

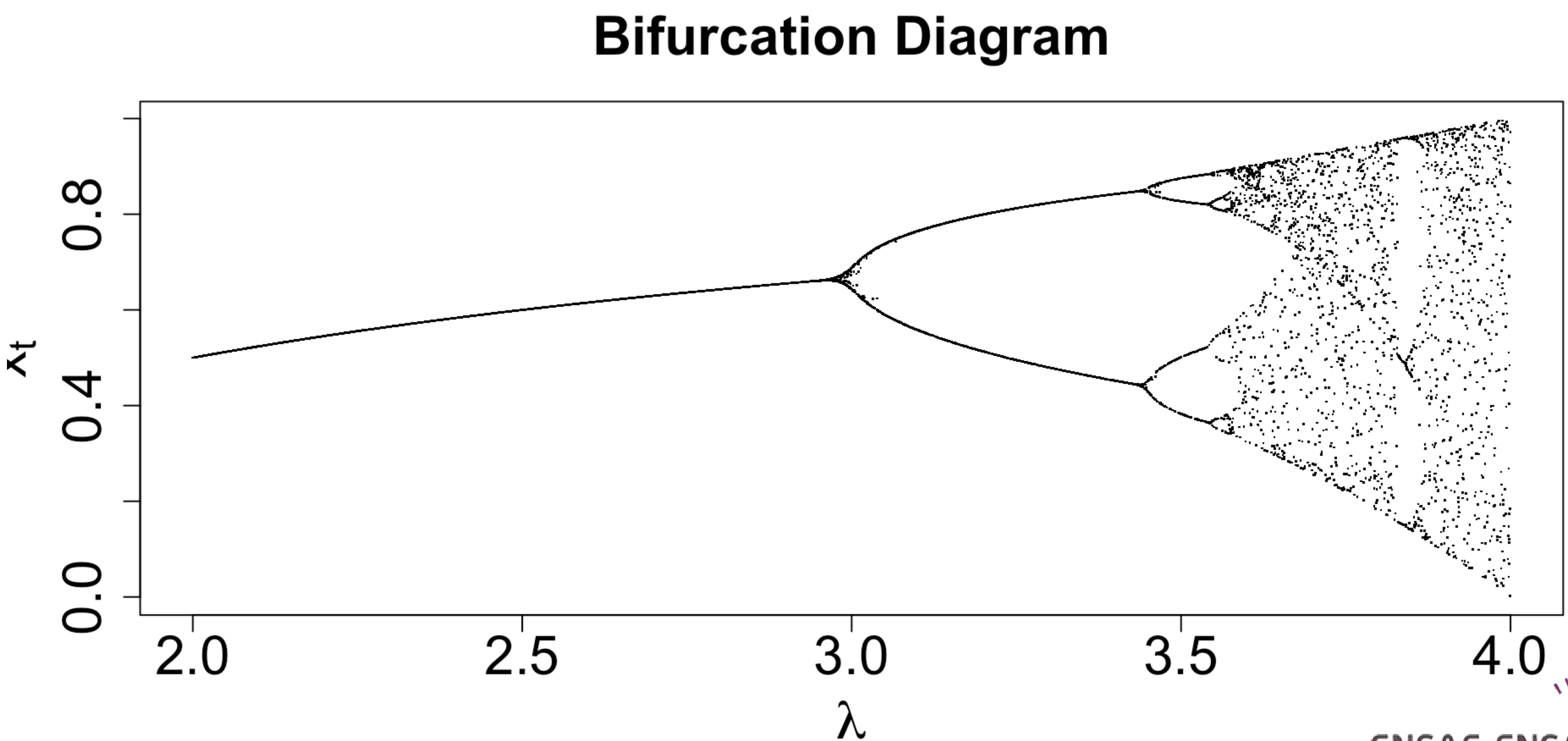
```r
library(Rcpp)

run_iteration <- function(n_iter, x0, lambda){
  for(i in 1:n_iter){
    x0 <- lambda * x0 * (1-x0)
  }
  return(x0)
}

cppFunction("
    double run_iteration_cpp(int n_iter, double x0, double lambda) {
      for (int i = 0; i < n_iter; i++) {
        x0 = lambda * x0 * (1.0 - x0);
      }
      return x0;
    }
  "         )

microbenchmark::microbenchmark(run_iteration(n_iter=1e6, x=0.5, lambda=3.8), run_iteration_cpp(n_ite
```

```
Unit: milliseconds
                                                   expr       min        lq
     run_iteration(n_iter = 1e+06, x = 0.5, lambda = 3.8) 17.765874 19.007231
 run_iteration_cpp(n_iter = 1e+06, x = 0.5, lambda = 3.8)  2.634496  2.856409
      mean    median        uq       max neval
 19.441426 19.655154 19.833156 21.343288   100
  2.994166  3.042877  3.066574  3.728909   100
```

ENSAE-ENSAI Formation continue (Cepe)

Solution to step 2 - plot

See more info on this over here.



Memory

Aside from time benchmarks, scanning our environment for large objects can always be useful.

- `ls()` provides you the list of elements in your env
- `object.size(<the element>)` gives you its size (also `lobstr::obj_size(<the element)`)
- `rm()` removes an object from the env
- `gc()` runs garbage collection (which runs periodically anyway so no need usually)

Why do we need garbage collection ?

- RHS data is created and bounded to one or more names
- When a modification on RHS is requested, a copy is made.
- When an element is removed from the environment, it removes the name and its bind to the value, but the value in memory is still taken, until the garbage collector does its job.

```r
a <- c(1, 2, 3)
b <- a # make copy

print(lobstr::obj_addr(a))
```

```
[1] "0x115b9cbe8"
```

```r
print(lobstr::obj_addr(b))
```

```
[1] "0x115b9cbe8"
```

```r
b[1] <- 0

print(lobstr::obj_addr(a))
```

```
[1] "0x115b9cbe8"
```

```r
print(lobstr::obj_addr(b))
```

```
[1] "0x114694ce8"
```

ENSAE-ENSAI
Formation continue
(Cepe)

Exercise: Order of Magnitude

Order of Magnitude

Generate random datasets:

- vary the data types: numerical, boolean, categorical
- vary the number of observations and columns

For each dataset generated, compute the object size and make a nice visualization from it

Solution (?)

```r
n <- 1e6

vec_1_million <- rnorm(n=n)
print(lobstr::obj_size(vec_1_million))
```

```
8.00 MB
```

```r
binary_vec_1_million <- round(rnorm(n=n))
print(lobstr::obj_size(binary_vec_1_million))
```

```
8.00 MB
```

```r
char_vec_1_million <- as.character(rnorm(n=n))
print(lobstr::obj_size(char_vec_1_million))
```

```
8.00 MB
```

Conclusion ?

To confuse you:

```r
char_vec_1_million <- as.character(rnorm(n=n))
print(lobstr::obj_size(char_vec_1_million))
```

```
8.00 MB
```

```r
toto <- list(char_vec_1_million, char_vec_1_million, char_vec_1_million)
print(lobstr::obj_size(toto))
```

```
8.00 MB
```

```r
toto[[1]][1] <- 9
print(lobstr::obj_size(toto))
```

```
95.65 MB
```

```r
banana <- "bananas bananas bananas"
print(lobstr::obj_size(banana))
```

```
136 B
```

```r
print(lobstr::obj_size(rep(banana, 100)))
```

```
928 B
```

ENSAE-ENSAI Formation continue (Cepe)

Take-Home Exercises

- Write an efficient ifelse block statement taking as input a numerical vector, with the conditions of your choice
- Investigate confusing memory examples in slide above
- Rcpp implementation of optimization function e.g. Decision Tree

Parallel treatment

The `future` package

If you look for resources around parallel programming, you'll inevitably found the following names: `snow`, `parallel`, `foreach` and `future`. Those are not the only ones but clearly the most popular.

In all instances, the code looks somewhat (if not a little more complex) like the one we shall use from the `future` package:

```r
library(future)
library(doFuture)

plan(multisession, workers=1) # of workers - to be changed
system.time({
  x <- foreach(i = 1:4) %dofuture% {
    Sys.sleep(2)
  }
})
```

```
   user  system elapsed 
  0.059   0.003   8.078 
```

All for one ?

```
plan(multisession, workers=4)

# choose a slow function
slow_fct <- function(x){ Sys.sleep(1e-5) ; log(x) }

# main vector
x <- c(1:1e5)

# iterate through the list
system.time({
  iter <- 1:length(x)
  foreach(i=iter, .combine='c') %dofuture% {
    slow_fct(x[i])
  }
})
```

```
  user  system elapsed 
11.830   0.136  12.530 
```

```
# iterate faster (especially in very large setting)
system.time({
  iter <- itertools::isplitIndices(n=length(x), chunks=4)
  foreach(i=iter, .combine='c') %dofuture% {
    slow_fct(x[i])
  }
})
```

```
  user  system elapsed 
 0.052   0.006   0.087 
```

```
# baseline
system.time({
  sapply(X=1:length(x), FUN=function(i){ slow_fct(x[i]) })
})
```

ENSAE-ENSAI Formation continue [Cepe]

```
   user  system elapsed 
  0.135   0.107   1.627 
```

Analytics & analysis (part 1)

ENSAE-ENSAI
Formation continue
(Cepe)

Data Wrangling

Recommended max data size	Package
~100K	base R
~1M	`readr`, `dplyr`
~10M	`data.table`
Inf	`arrow`, `duckdb`

ENSAE-ENSAI
Formation continue
(Cepe)

Analytics & analysis on cluster

db connection h2o spark

ENSAE-ENSAI
Formation continue
(Cepe)

Resources

ENSAE-ENSAI
Formation continue
(Cepe)

Resources

Textbooks

Advanced R, 2nd Edition, utilitR book, Advanced R training

Benchmarks

Data Wrangling Benchmark 1, Data Wrangling Benchmark 2

Cheatsheets & documentation links

data.table, dtplyr, sparklyr

Miscellaneous

SAS to R