(Half) Big Data handling with R

Severine Bord, Tristan Mary-Huard

2018-06-29

Dataset 1: Arabidopsis

1,307 Arabidopsis lines
Sequenced at 214,051 biallelic markers

```
Arabidopsis[1:5,1:10]
```

```
##    L1 L2 L3 L4 L5 L6 L7 L8 L9 L10
## M1  1  0  1  1  0  1  0  1  1   1
## M2  1  0  1  1  0  1  1  1  1   1
## M3  1  0  1  1  0  1  1  1  1   1
## M4  0  0  0  0  1  0  0  0  0   0
## M5  0  0  0  0  0  0  0  0  0   0
```

Goal: perform basic descriptive analysis
- Allelic frequency per marker,
- Filtering markers with low polymorphism,
- Compute a kinship matrix (ie a genetic similarity matrix).

Dataset 2: Orange

Hour per hour activity of 22,772 relay antennas:
Id variables: *Date_Time, Id_RelAnt*
Measurement variables: *NbSim_Fr, NbSim_Other, NbCom_Fr, NbCom_Other*

```
Orange[1:5,]
```

```
##        Date_Time  Id_RelAnt NbSim_Fr NbSim_Other NbCom_Fr NbCom_Other
## 1 2017-04-15 00 00000001A1       37           0      141           0
## 2 2017-04-15 00 00000001B1      134           0      503           0
## 3 2017-04-15 00 00000001B2      170           2      752           2
## 4 2017-04-15 00 00000001B3       93           0      550           0
## 5 2017-04-15 00 00000001C1      122           2     1149          13
```

Goal: perform basic descriptive analysis
- compute mean activity per hour.

By default analysis

Arabidopsis

Load data

```
read.table('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';') %>%
  as.matrix() -> Arabidopsis
```

57s

Compute frequencies

```
Freq <- apply(Arabidopsis,2,mean)
```

4s

Filter frequencies

```
MinMaf <- 0.05
MafFilter <- Freq > MinMaf | Freq < 1-MinMaf
Filtered <- Arabidopsis[MafFilter,]
```

6s

Arabidopsis

Compute kinship

```
Kinship <- (crossprod(Filtered) + crossprod(1-Filtered))/nrow(Filtered)
```

9.5s

Invert matrix

```
solve(Kinship)
```

0.2s

Total (default) computational time: **76.8s**

Arabidopsis

Load data

```
fread('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';') %>%
  as.matrix() -> Arabidopsis
```

9.4s ($\times 6$)

Compute frequencies

```
Freq <- rowMeans(Arabidopsis)
```

0.7s ($\times 6$)

Invert matrix

```
InvKinship <- chol2inv(chol(Kinship))
```

0.2s ($\times 6$)

Arabidopsis

Summary

Greatly improves performance:
$\text{Total (default) computational time:}$ **76.8s**
$\text{Total (improved) computational time:}$ **25.7s**

But…

Arabidopsis from read.table: **1067.4 Mb**
Arabidopsis from fread: **1067.4 Mb**
More on this latter…

Does not solve the **memory** problem…

Orange

```
MeanPerHour <- Orange %>%
  mutate(Time = substr(Date_Time,start=12,stop=13)) %>%
  group_by(Time) %>%
  summarise(M_NbSim_Fr=mean(NbSim_Fr),M_NbSim_Other=mean(NbSim_Other),
            M_NbCom_Fr=mean(NbCom_Fr),M_NbCom_Other=mean(NbCom_Other))
head(MeanPerHour)
```

```
## # A tibble: 6 x 5
##   Time  M_NbSim_Fr M_NbSim_Other M_NbCom_Fr M_NbCom_Other
##   <chr>      <dbl>         <dbl>      <dbl>         <dbl>
## 1 00          77.3         1.71       428.           5.64
## 2 01          40.5         1.16       208.           3.75
## 3 02          23.7         1.00       107.           3.17
## 4 03          16.1         0.861       63.6          2.69
## 5 04          13.9         0.794       47.6          2.43
## 6 05          15.4         0.782       46.6          2.44
```

Again, **limitation comes from loading the data**…

Today's topic

Handle "half big" data, i.e. datasets whose size
- is too large to be loaded in R memory,
- is small enough to be loaded in RAM.
($\approx$ 5-15 Go)

$\color{blue}{\text{Different strategies and packages}}$

1. **Chunk the data yourself**
   * no package required…
   * … but requires to adapt your code
   * can be efficient **IF** calculations can be chunked.
2. **Use RAM rather that R memory**
   * to perform matrix algebra: package `bigmemory`
   * to perform data curation: package `sparklyr`

Chunking

Arabidopsis revisited

Strategy Split the data into K chunks, compute on each chunk the required quantities, then collect all results.

Are the tasks chunk friendly ?
- Allelic frequency per marker,
- Filtering markers with low polymorphism,
- Compute a kinship matrix (ie a genetic similarity matrix).

Any idea ?

```
Freq <- (mc)lapply(1:K, function(k){
  fread(Chunk_k)
  rowMeans(Chunk_k)
})
```

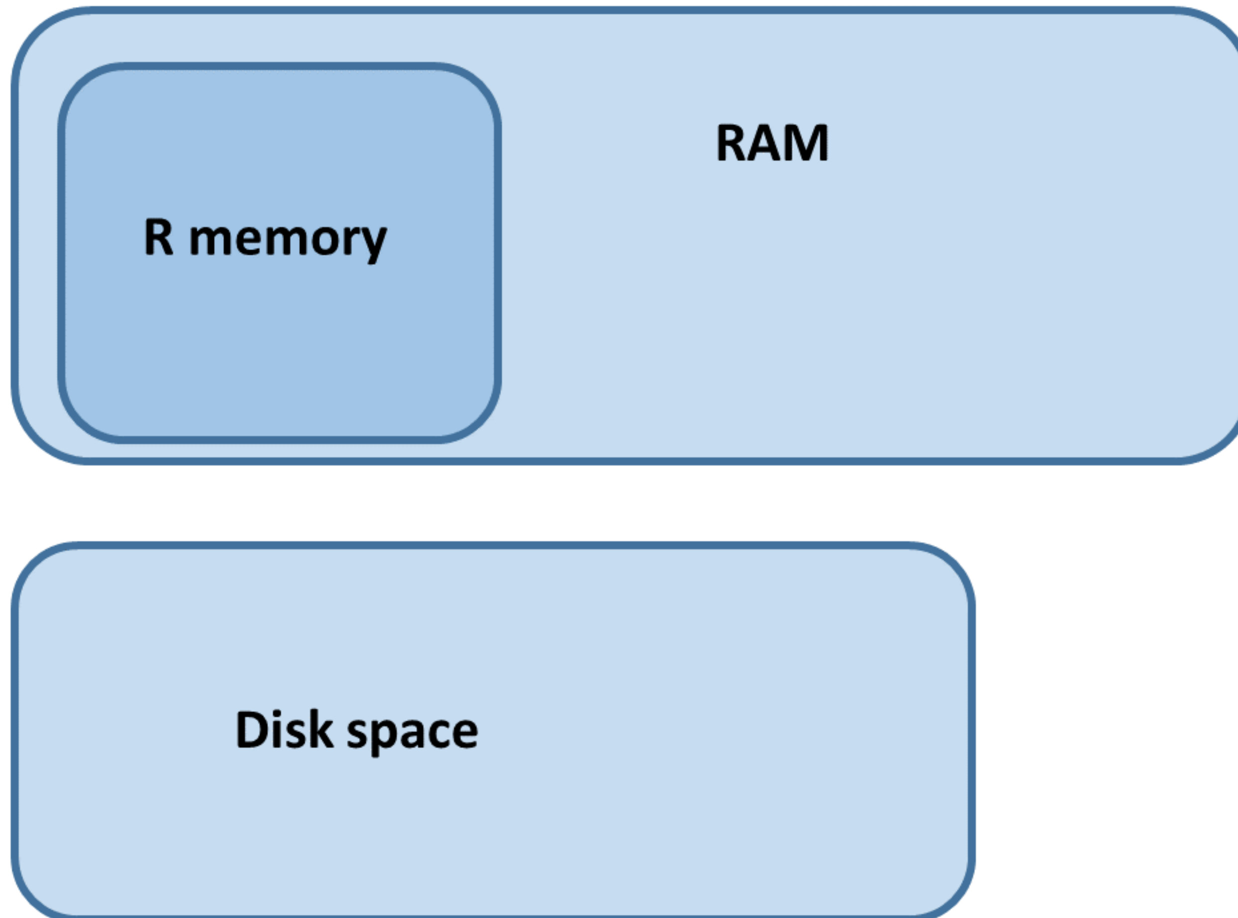
Arabidopsis revisited

```
trial <- file('./Data/Arabidopsis/Arabidopsis.txt')
Kinship <- matrix(0,NbInd,NbInd)
open(trial)
Freq <- (mc)lapply(NbSnpPerRound, function(nblines){
  Don <- matrix(scan(trial,skip=0,nlines=nblines,quiet=T,sep=';'),
                nrow=nblines,byrow = TRUE)
  FreqLoc <- rowMeans(Don)
  Kinship <<- Kinship + crossprod(Don) + crossprod(1-Don)
  return(FreqLoc)
})
close(trial)
```

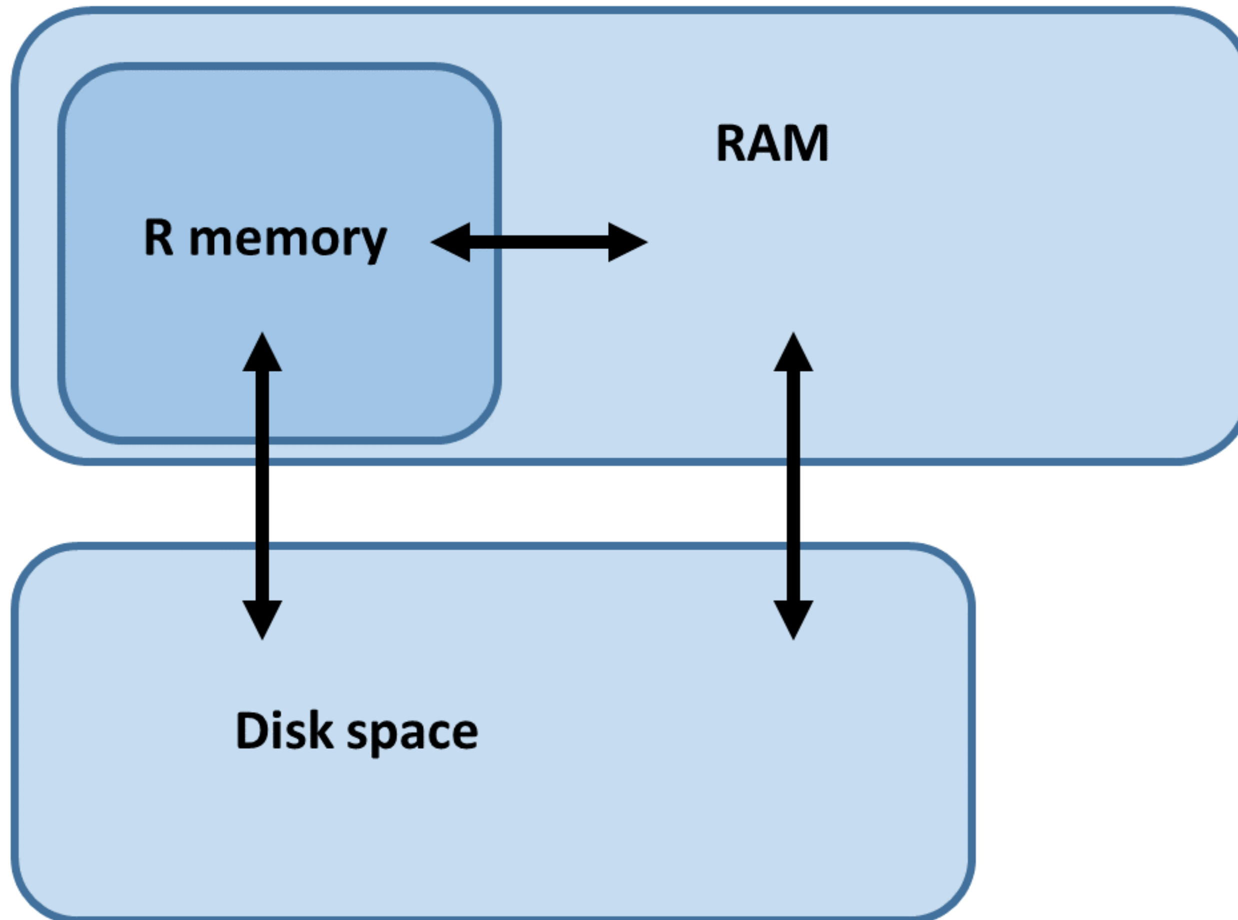
Total computational time: **94s**

The "big" packages

Main idea



Main idea



The 'big' package family

The big family consists of several packages for performing tasks on large datasets:

1. `bigmemory` for loading of large matrices in RAM.
2. `bigalgebra` provides BLAS and LAPACK linear algebra routines for native R matrices and big.matrix.
3. `biganalytics` provides analysis routines on big.matrix such as GLM and bigkmeans.

Limitations
Matrices can contain only one type of data.
Since the matrix will be stored as a C++ object in RAM, data types for elements are dictated by C++: `double`, `integer`, `short`, `char`.

17/44

Getting prices right!

```r
memory.size()
Arabidopsis <- fread('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';',colClasses = 'integer')
format(object.size(Arabidopsis),units='Mb')
memory.size()
rm(list=ls())
gc()
memory.size()
Arabidopsis <- read.table('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';',colClasses = 'integer')
format(object.size(Arabidopsis),units='Mb')
memory.size()
```

```
> memory.size()
[1] 107.18
> Arabidopsis <- fread('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';',colClasses = 'integer')
Read 214051 rows and 1307 (of 1307) columns from 0.521 GB file in 00:00:09
> format(object.size(Arabidopsis),units='Mb')
[1] "1067.4 Mb"
> memory.size()
[1] 1205.98
> rm(list=ls())
> gc()
          used (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells 1152461 61.6    1770749   94.6   1770749   94.6
Vcells 2296900 17.6  442908268 3379.2 841649793 6421.3
> memory.size()
[1] 106.17
> Arabidopsis <- read.table('./Data/Arabidopsis/Arabidopsis.txt',header=F,sep=';',colClasses = 'integer')
> format(object.size(Arabidopsis),units='Mb')
[1] "1067.4 Mb"
> memory.size()
[1] 1542.82
>
```

Arabidopsis, the 'big' way

```
206
207        #### Load data from a file
208
209
210 ## Load the data
211 ptm <- proc.time()
212 Arabidopsis <- read.big.matrix("./Data/Arabidopsis/Arabidopsis.txt",
213                               type ="integer", header = F,sep=';')
214 Times.big[cpt,] <- (proc.time()-ptm)[1:3]
215 Names.big[cpt] <- 'Load with read.big.matrix'
216 cpt <- cpt+1
217 NbSnp <- nrow(Arabidopsis)
218 NbLines <- ncol(Arabidopsis)
219
220
221        #### Compute Freq, filter on MAF and compute kinship
222
223
224 ##Compute frequencies using colmean
225
```

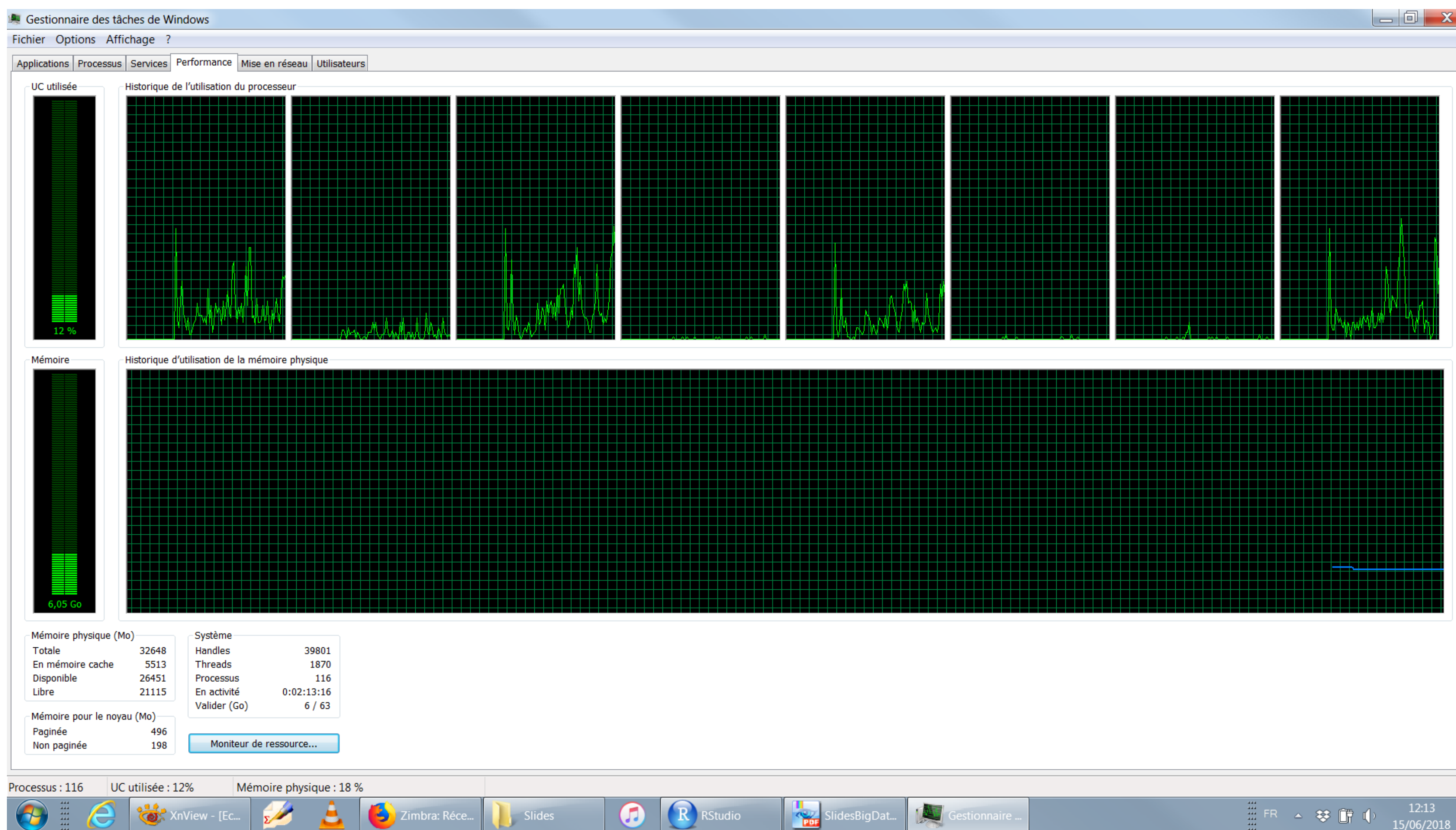
```
> memory.size()
[1] 117.99
>
```

Environment:

Data
Times.big num [1:100, 1:3] 0 0 0 ...
Values
cpt 1
Names.big logi [1:100] NA NA NA NA ...

19/44

Arabidopsis, the 'big' way



Physical memory: 6.05 Go

20/44

Arabidopsis, the 'big' way

```
Arabidopsis <- read.big.matrix("./Data/Arabidopsis/Arabidopsis.txt",
                               type ="double", header = F,sep=';')
```

```
206
207         #### Load data from a file
208
209
210 ## Load the data
211 ptm <- proc.time()
212 Arabidopsis <- read.big.matrix("./Data/Arabidopsis/Arabidopsis.txt",
213                                type ="integer", header = F,sep=';')
214 Times.big[cpt,] <- (proc.time()-ptm)[1:3]
215 Names.big[cpt] <- 'Load with read.big.matrix'
216 cpt <- cpt+1
217 NbSnp  <- nrow(Arabidopsis)
218 NbLines <- ncol(Arabidopsis)
219
220
221         #### Compute Freq, filter on MAF and compute kinship
222
223
```

Data	
Arabidops…	Formal class big.matrix
Times.big	num [1:100, 1:3] 0 0 0 …
Values	
cpt	1
Names.big	logi [1:100] NA NA NA NA …

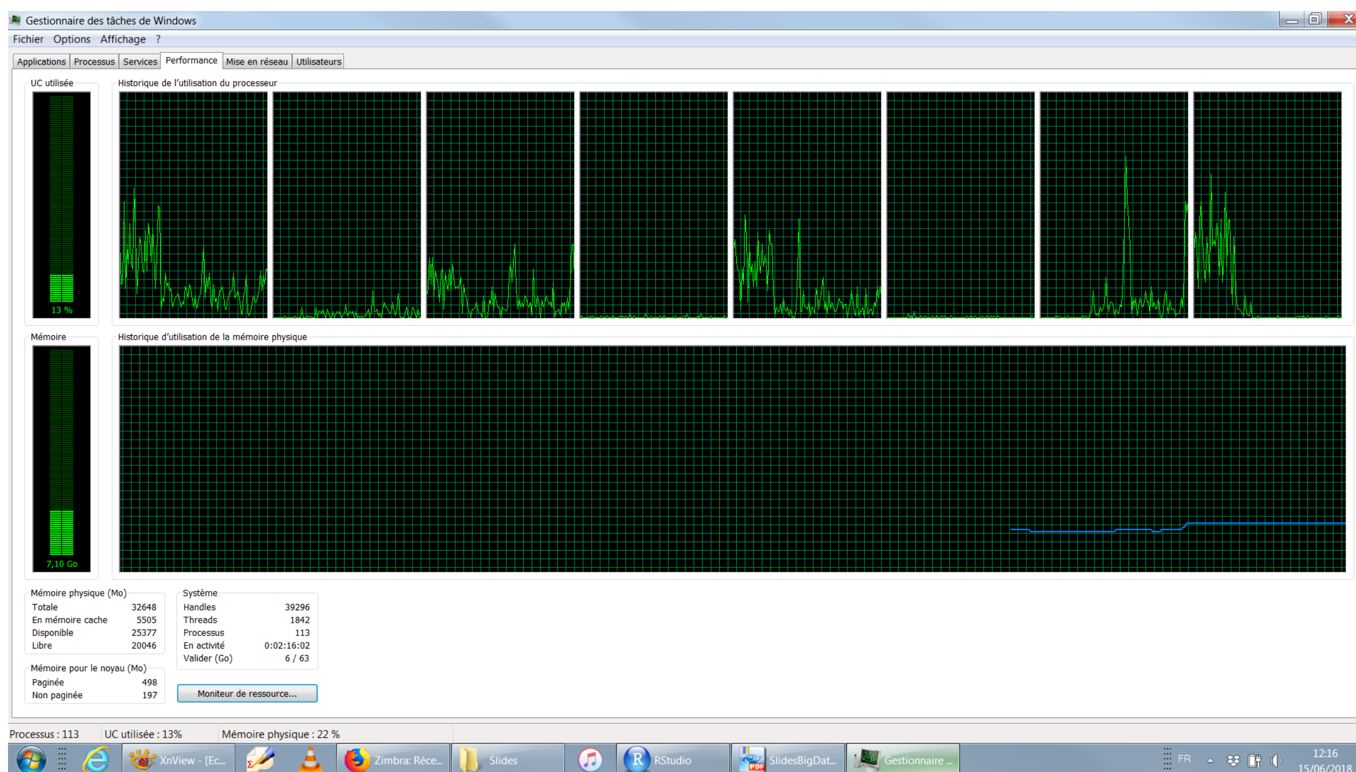
```
> memory.size()
[1] 117.99
> Arabidopsis <- read.big.matrix("./Data/Arabidopsis/Arabidopsis.txt",
+                                type ="integer", header = F,sep=';')
> memory.size()
[1] 118.03
> Arabidopsis
An object of class "big.matrix"
Slot "address":
<pointer: 0x000000001cf13a70>

>
```

21/44

Arabidopsis, the 'big' way

```
Arabidopsis <- read.big.matrix("./Data/Arabidopsis/Arabidopsis.txt",
                               type ="double", header = F,sep=';')
```



Physical memory: 7.1 Go

Matrix algebra

Available from `bigmemory`:

`dim`,`ncol`,`nrow`

Available from `bigalgebra`:

`crossprod`,`tcrossprod`,`t`,`%*%`, `eigen`,`chol`

Available from `biganalytics`:

`col{mean,min,max,var,sd,sum,prod}`

Not available:

`svd`,`solve`,

Frequencies

Several ways to compute frequencies:

```
##Compute frequencies using apply
Freq <- biganalytics::apply(Arabidopsis,1,mean)

##Compute frequencies using rowMeans
Freq <- rowMeans(Arabidopsis[,])

##Compute frequencies using algebra
Freq <- Arabidopsis[,]%*%rep(1/NbLines,NbLines)

##Compute frequencies using big algebra
Freq <- Arabidopsis%*%as.big.matrix(rep(1/NbLines,NbLines))

##Compute frequencies using big algebra
Freq <- Arabidopsis %>% t %>% colmean %>% as.big.matrix
```

What about memory ?

Kinship and inverse

```
##Local version
Kinship <- ( crossprod(Arabidopsis[,]) + crossprod(1-Arabidopsis[,])
           )/nrow(Arabidopsis)
InvKin <- chol2inv(chol(Kinship))

##RAM version (wherever feasible)
Kinship <- ( crossprod(Arabidopsis) + crossprod(1-Arabidopsis)
           )/nrow(Arabidopsis)
InvKin <- chol2inv(chol(Kinship)[,])
```

Loading time: **64s**
Filtering time: **15.5s**

Processing time (local): **13s**
Processing time (RAM): **18.2s**
Compared to 10.3 with the default analysis.

Summary

Lesson 1

The `big` packages make your analyses **feasible, not faster**.
No need (or gain) to perform all computations in RAM.
Store your big objects there, keep the rest as usual…

Lesson 2

Check memory storage in every ways…
… including tmp files !

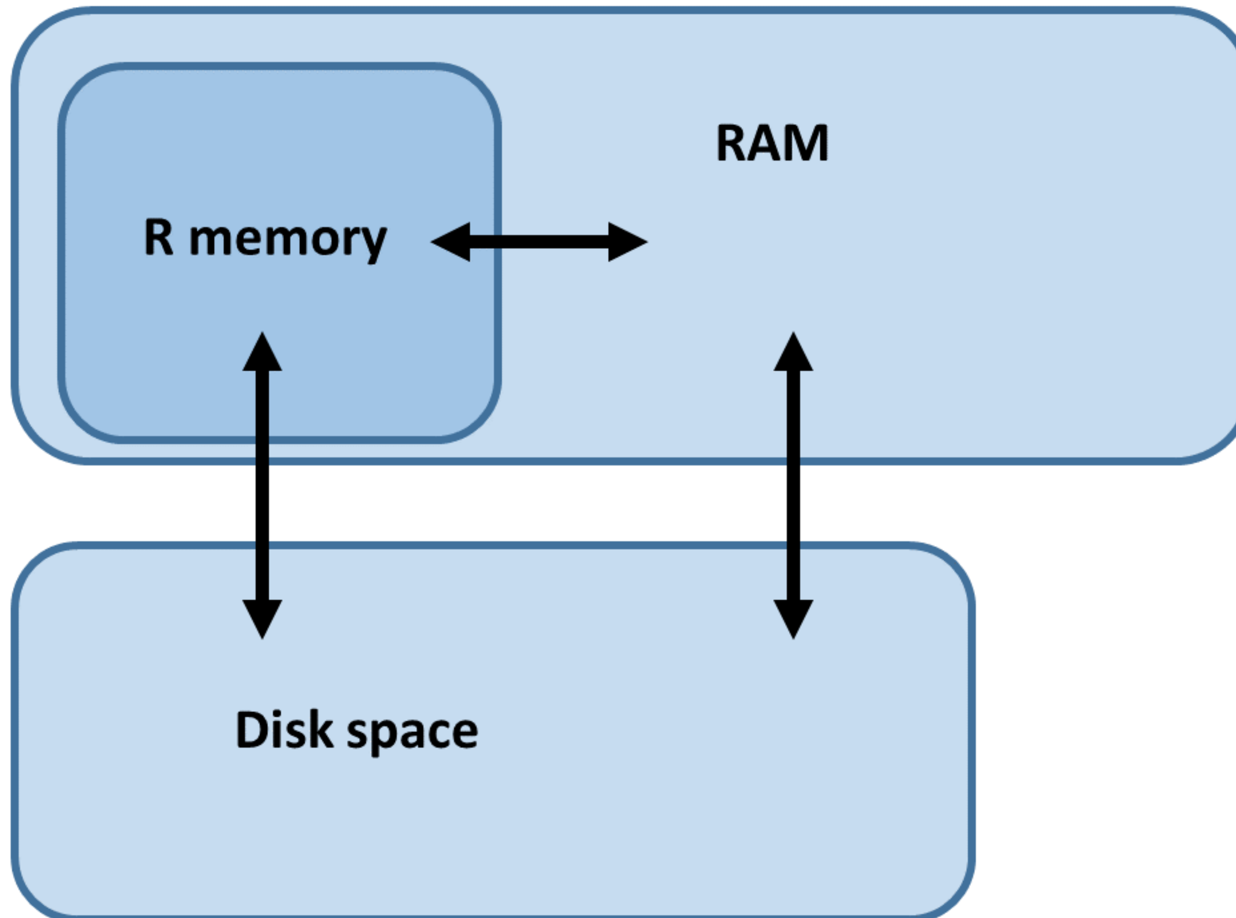
```
##Compare some basic functions to compute frequencies
res <- microbenchmark(rowMeans(Arabidopsis[,]),
                      Arabidopsis %>% t %>% colmean %>% as.big.matrix,
                      Arabidopsis%*%as.big.matrix(rep(1/NbLines,NbLines)),
                      times = 50)
```

Uncovered topics

Some really efficient functions, eg `mwich`.
Possible to load the matrix in shared memory.

Spark for R

Local use of Spark



Same philosophy about using RAM outside R memory.
BUT spark may be used on clusters for **distributed memory**.

What is Hadoop?

Hadoop is an **open-source software framework** for **storing data and running applications on clusters**.

- good for simple information requests and problems that can be divided into independent units,
- not efficient for iterative and interactive analytic tasks.

Based on two concepts:

- the 'HDFS' file system: chunks and distributes the data on the different nodes of the cluster,
- the 'MapReduce' algorithm: translates the task into small distributed operations, summarizes results obtained from the nodes.

Nodes do not communicate except through sorts/shuffles.

What is (Apache) Spark?

Apache Spark:

- is **more accessible** and **more powerful** than Hadoop.
- began life in 2009 as a project within the AMPLab at the University of California, Berkeley.
- became an incubated project of the Apache Software Foundation in 2013.
- Simplicity, Speed, Support

Tasks most frequently associated with Spark include

- **interactive queries** across large data sets,
- **processing of streaming data** from sensors or financial systems, and
- **machine learning** tasks.

What Does Spark Do?

Handles **several petabytes** of data at a time, **distributed** across a cluster of thousands of cooperating physical or virtual servers.

Supports different **languages** such as Java, Python, R, and Scala, benefits from many classical libraries.

Performance:

- From the beginning, Spark was optimized to run in memory.
- process data far more quickly than Hadoop MapReduce (write data to and from computer hard drives between each stage of processing, 100 times faster than Hadoop MapReduce).

In this presentation: 0.01% of Spark capacities are explored !

Requirements

Install Java 8
* To create a virtual machine on your computer (for local use),
* To interact/communicate with Spark

Install sparklyr
From CRAN or gitHub. Once install, you can
* install spark (and hadoop) on your machine using command `spark_install`,
* open a spark session using command 'spark_connect'

Start playing with your new toy !

Orange, the Spark way

```
setwd('D:/R/StateOfTheR/BigData')
Orange_tbl <-
  spark_read_csv(sc = sc, name = 'Orange',
                 path = './Data/Orange/NIDT_D4C_2G3G4G_2017105.CSV.gz',
                 header = FALSE, delimiter = ';')
```

```
## # Source:   lazy query [?? x 6]
## # Database: spark_connection
##   Date_Time     Id_RelAnt  NbSim_Fr NbSim_Other NbCom_Fr NbCom_Other
##   <chr>         <chr>         <int>       <int>    <int>       <int>
## 1 2017-04-15 00 00000001A1       37           0      141           0
## 2 2017-04-15 00 00000001B1      134           0      503           0
## 3 2017-04-15 00 00000001B2      170           2      752           2
## 4 2017-04-15 00 00000001B3       93           0      550           0
## 5 2017-04-15 00 00000001C1      122           2     1149          13
## 6 2017-04-15 00 00000001D1       19           0       79           0
```

Lazy evaluation: what does it mean ?

```
MeanPerHour <- Orange_tbl %>%
  mutate(Time = substr(Date_Time,start=12,stop=13)) %>%
  group_by(Time) %>%
  summarise(M_NbSim_Fr=mean(NbSim_Fr),...)
```

$0.01s$ (Local: 0.07)

```
GlobalMean_NbSimFr <- MeanPerHour %>% summarise(Mean=mean(M_NbSim_Fr))
```

$0.0s$ (Local: 0.0)

```
GlobalMean_NbSimFr
```

$0.34s$ (Local: 0.01)

Pay attention to the way computational times are evaluated !

34/44

Transfering data

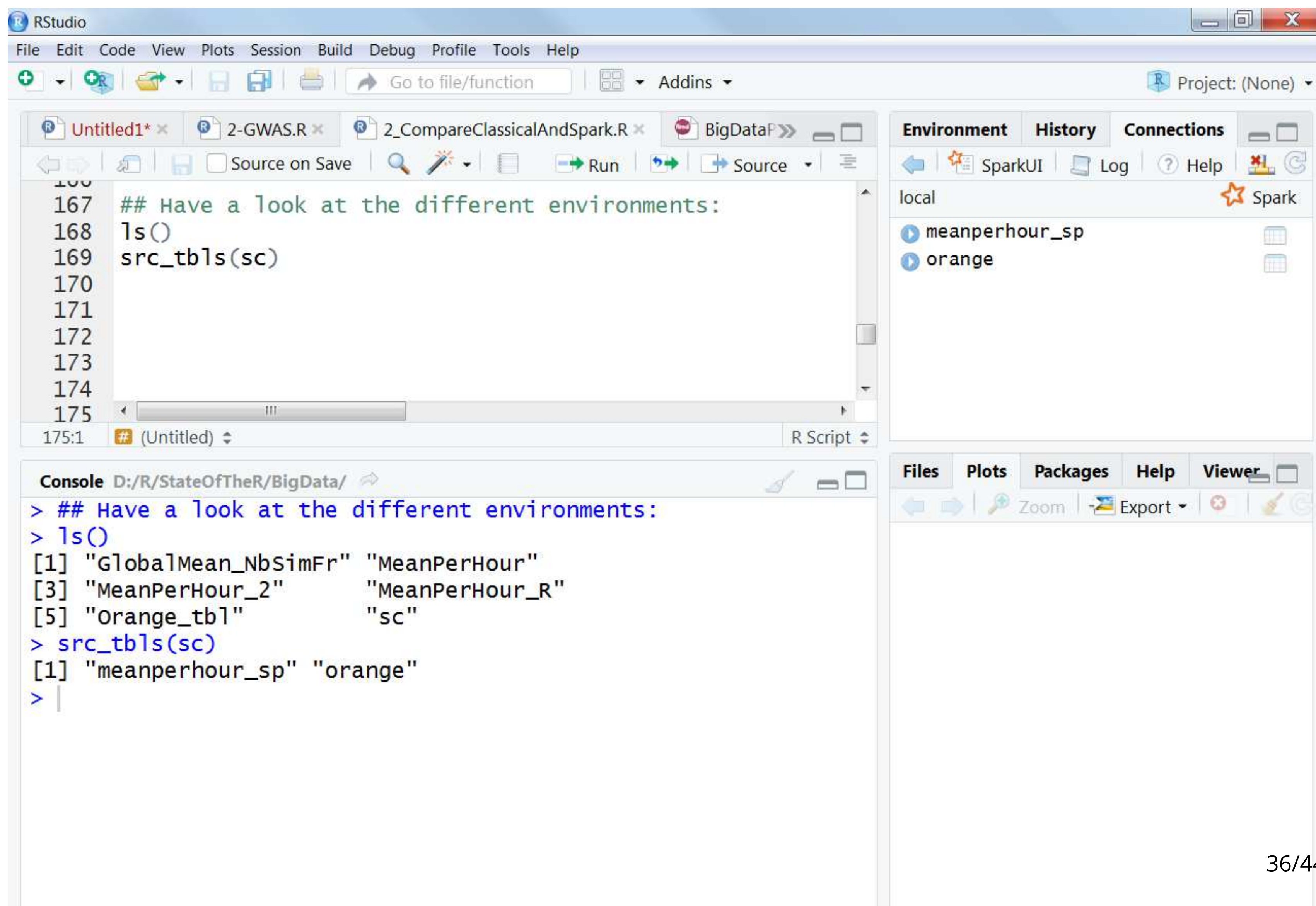
```
## From Spark to R memory
MeanPerHour_R <- collect(MeanPerHour)

## From Spark to Disk
spark_write_csv(x=MeanPerHour, path = './Results/MeanPerHour.csv',
                header = T, delimiter = ';')

## From R to Spark
MeanPerHour_2 <- copy_to(sc, MeanPerHour_R,
                         name= "MeanPerHour_Sp",overwrite = T)

## Have a look at the different environments:
ls()
src_tbls(sc)
```

Check where the objects are



36/44

Pay attention to object classes

```
ListRelAnt <- Orange_tbl %>% distinct(Id_RelAnt)
ListRelAnt$Id_RelAnt[1:10]
```

```
## NULL
```

```
collect(ListRelAnt)$Id_RelAnt[1:10]
```

```
##  [1] "00000001B2" "00000001D1" "00000001F5" "00000001J4" "00000001Q1"
##  [6] "00000001T2" "00000001V1" "00000001W4" "00000002C1" "00000002M2"
```

Use of R functions to create tables in the spark environment

What are the difference between the 3 following expressions ?

```
AddDensity_trial1 <- MeanPerHour %>%
  mutate(Density = dnorm(M_NbSim_Fr,mean = 100,sd=50))

AddDensity_trial2 <- MeanPerHour %>%
  collect() %>%
  mutate(Density = dnorm(M_NbSim_Fr,mean = 100,sd=50))

AddDensity_trial3 <- MeanPerHour %>%
  spark_apply(function(d){dnorm(d$M_NbSim_Fr,mean = 100,sd=50)})
```

38/44

An example of "fancy" application

```
MeanPerHour %>%
  mutate(TimeNum = Time + 0) %>%
  mutate(TimeSlice = ifelse(TimeNum %in% c(0:5,18:23), 'Evening',
                            ifelse(TimeNum %in% 6:11, 'Morning',
                                   'Afternoon'))) %>%
  mutate(TimeRecoded = ifelse((TimeSlice == 'Evening')&(TimeNum %in% 18:23),
                              TimeNum-24, TimeNum)) %>%
  group_by(TimeSlice) %>%
  spark_apply(
    function(d) broom::tidy(lm(M_NbCom_Fr ~ TimeRecoded, d)),
    names = c("term", "estimate", "std.error", "statistic", "p.value","sigma"),
    group_by = "TimeSlice"
  )
```

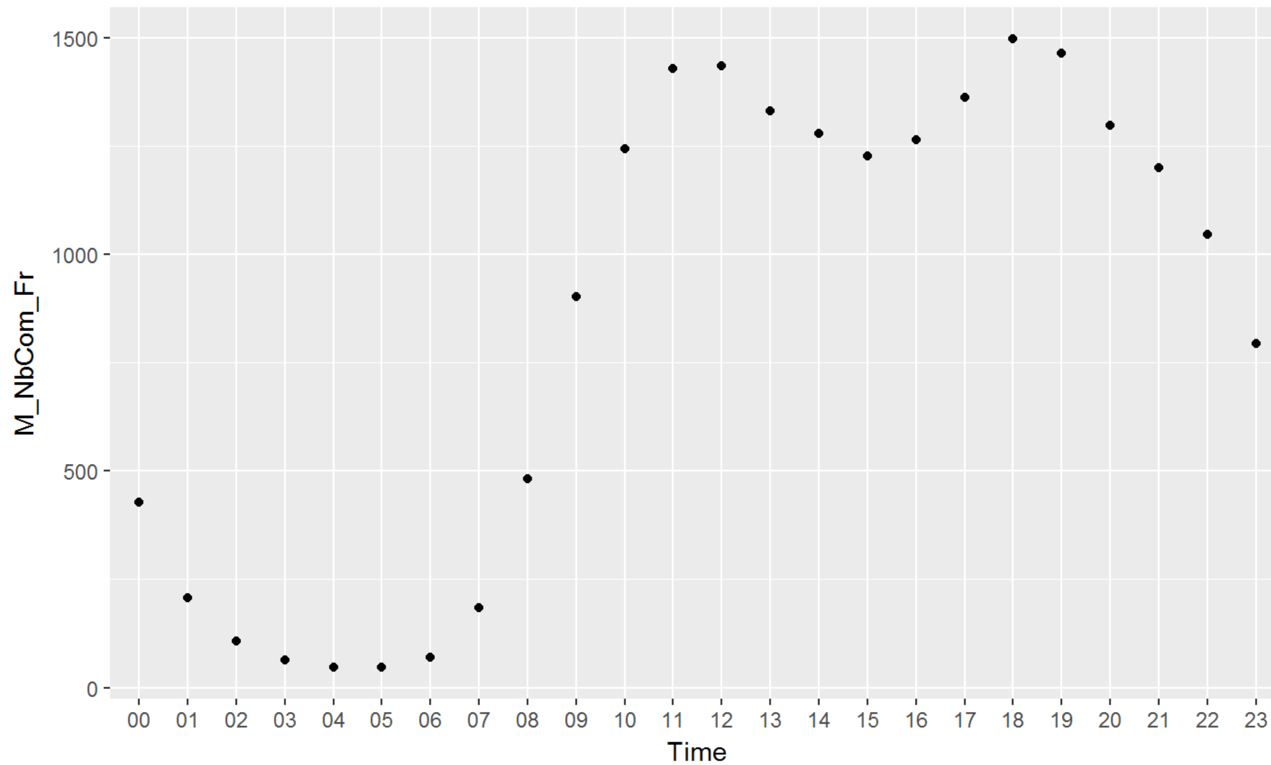
About the tidy function

Broom_vignette

An example of "fancy" application

```
## # Source:   table<sparklyr_tmp_211826b15bcf> [?? x 6]
## # Database: spark_connection
##   TimeSlice term        estimate std.error statistic      p.value
##   <chr>     <chr>          <dbl>     <dbl>     <dbl>        <dbl>
## 1 Evening   (Intercept)    604.       44.0    13.7   0.0000000828
## 2 Evening   TimeRecoded   -160.       12.6   -12.7   0.000000176
## 3 Morning   (Intercept)  -1805.      189.     -9.57  0.000667
## 4 Morning   TimeRecoded    297.       21.8    13.6   0.000167
## 5 Afternoon (Intercept)   1573.      263.      5.97  0.00394
## 6 Afternoon TimeRecoded    -17.7      18.0    -0.982 0.382
```

An example of "fancy" application



Machine learning with spark

Many classical/ML algorithms available:

Supervised methods

`ml_linear_regression`, `ml_logistic_regression`, `ml_survival_regression`, `ml_generalized_linear_regression`, `ml_decision_tree`, `ml_random_forest`, `ml_gradient_boosted_trees`,...

Unsupervised methods

`ml_kmeans`

Exploratory methods

`ml_pca`

Beyond local use of Spark

Working on a cluster, one can
- distribute the data: each node of the cluster will get a (random) chunk of the data,
- distribute computation: each node can process only the data at hand.

Consequences:
- real parallel computation,
- exact results for many reshaping treatments,
- non-exact results for many ML tasks ?

Summary

Both `bigmemory` and `sparklyr` allows one to circumvent the memory limitation of R.

The two packages are **still in development**…
- latest versions not always on CRAN,
- inconsistent/obsolete tutorials,
- many compatibility troubles.