Data Friday : A quick introduction to R

GjT - Ioda Team

Tuesday, July 15, 2014

Contents

Introduction ? R

R est l'outil de Data open source le plus cit? au monde d'apr?s la derni?re enqu?te de Rexer Analytics. Ce week-end encore, un tweet le citait comme le langage math?matique avec un haut degr? d'abstraction le plus complet (kdnugget).C'est un environnement int?gr? de manipulation de donn?es, de calcul et de pr?paration de graphiques. Toutefois, ce n'est pas seulement un ?autre? environnement statistique (comme SPSS ou SAS, par exemple), ni un enviromment purement de Machine Learning (SparK, Mahout, etc), mais aussi un langage de programmation complet et autonome qui propose les fonctionnalit?s **les plus avanc?es au monde autour de la manipulation de la donn?e et de la mod?lisation en Data Science ou en Statistiques computationelles**

Ce tutoriel est une introduction ? R, de ses concepts basiques h?rit?s de sa vocation initiale (donner au math?maticien un degr? d'abstraction pour se concentrer sur les th?ories et les m?thodes) jusqu'au niveau le plus avanc? aujourd'hui, le calcul paral?lle dans un environnement Big Data.

Bien qu'il ait vocation ? donner une introduction litt?rale des bases de la programmation avec R, on ne fera pas l'?conomie d'un vrai apprentissage du langage et d'un vrai manuel (s'il existe !)

Dans cette premi?re it?ration, nous n'allons pas jusqu'? la notion de calcul parall?le, le format ne s'y pr?te pas. Mais n?anmoins, nous abordons les grandes notions et introduisons quelques biblioth?ques qui bien qu'ajout?es au langage par une communaut? active, n'en finissent pas moins de devenir de v?ritables standards de manipulations. C'est un document de travail et il sera modifi? progressivement aussi en fonction de vos retours et de vos commentaires et nous utiliserons gitlab pour communiquer et ainsi, vous pourrez toujours avoir acc?s aux derni?res versions disponibles !

Je me suis appuy? pour r?aliser ce tutoriel exclusivement sur des forums de discussions, des exp?riences personnelles et je ne saurais donc remercier une contribution particuli?re si ce n'est toute la communaut? cran et la R Mailling List ? laquelle je suis abonn? depuis bient?t 8 ans.

Un mot pour RStudio qui a popularis? depuis 5 ans environ la mani?re de travailler dans R au point o? certains se demandent si parfois il s'agit de la m?me chose. RStudio fourni toutes les couches au dessus de R, notamment la possibilit? de travailler dans un IDE int?gr? et intelligent (auto-completion, Visualisation avanc?e, espaces de travail convivial) et surtout l'int?gration des couches html5, markdown et latex qui permettent par exemple de r?aliser ce tutoriel exclusivement dans R, sans jamais en sortir.

Enfin, quelques erreurs peuvent s'?tre gliss?es dans ce document de travail et selon la formule habituelle, ces erreurs sont enti?rement celles des autres !

R

Huit choses essentielles ? retenir ? propos de R :

- Il est open source
- R n'est pas RStudio
- L'anc?tre de R est S-plus (dont personne ne parle plus aujourd'hui)
- R est un langage interpr?t? (contrairement au C ou C++ qui sont compil?s)
- Avec R, les notions de math?matiques, particuli?rement de calcul matriciel sont **importantes**
- On ne fait pas de boucles avec R
- R demande un haut niveau d'abstraction
- R est le langage le plus cit? sur stackoverflow

L'aide en ligne et les tutoriels foisonnent. Quelques uns

http://fr.openclassrooms.com/informatique/cours/effectuez-vos-etudes-statistiques-avec-r

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

https://www.youtube.com/playlist?list=PLOU2XLYxmsIK9qQfztXeybpHvru-TrqAP

Mais de fa?on g?n?rale, il est plus simple d'entrer dans R, comme dans n'importe quel langage de programmation par une question et de d?rouler ensuite selon un mod?le de test &learn

Premiers pas :

```
> 2+2
```

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

```
> exp(10)
```

```
## [1] 22026
```

```
> sum(1:10)/10
```

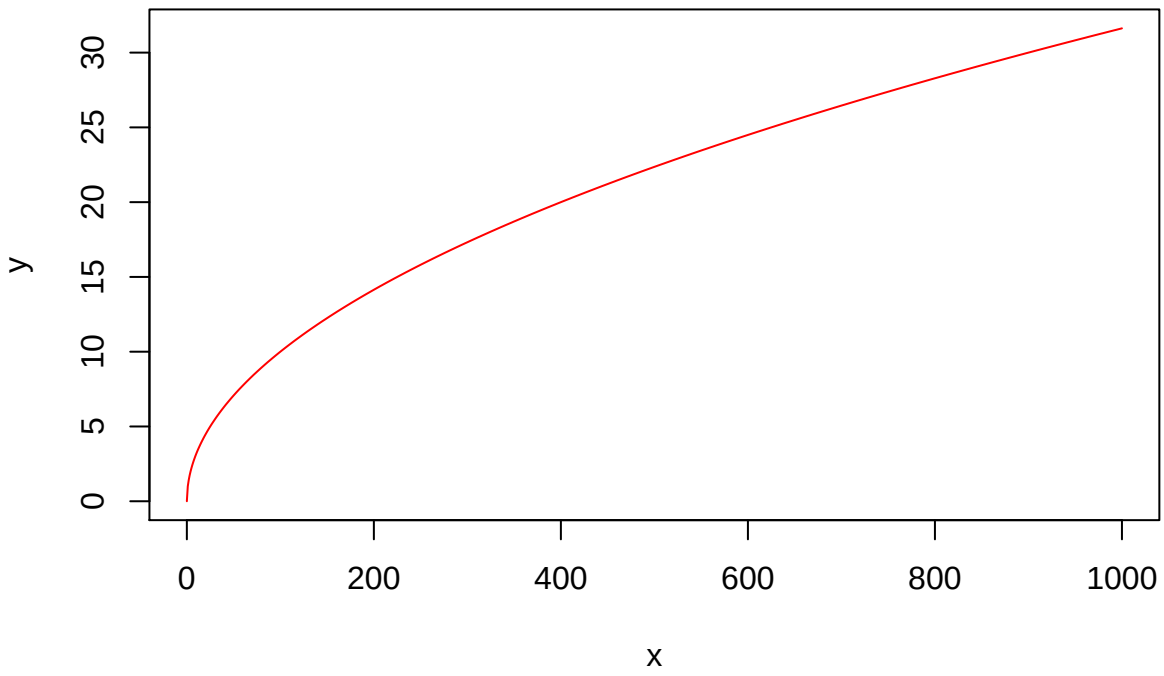
```
## [1] 5.5
```

```
> sqrt(10)/2 + log10(2)
```

```
## [1] 1.882
```

Fr?deric faisait remarquer que R lui faisait penser ? une grosse calculatrice de classe pr?paratoire, notamment parce que la d?claration des variables a une structure conventionnelle tr?s minimale. On verra par la suite que l'op?rateur d'assignation <- pouvait tr?s bien ?tre remplac? par une ?galit? sans que cela ne pertube l'ex?cution des scripts.

```
> x= seq(0,1000,1)
> y = sqrt(x)
> plot(x,y,col="red",type='l')
```



R est un langage de script aussi, bien qu'il soit tr?s peu utilis? de cette fa?on. On peut ?crire des fonctions et les r?utilisez, appeler des programmes dans d'autres programmes, etc...

Son coeur est ?crit en Fortran et il n'est pas rare de voir certains puristes y avoir recours (avec le C), lorsqu'il s'agit d'optimiser la m?moire.

Op?rations Basiques

R travaille sur les vecteurs. C'est l'unit? de base du langage. Un vecteur est une matrice ? une dimension. Le vecteur, symboolis? par c est l'op?rateur basique du langage :

```
> x = c(1,2,3,4,5)
> print(x)
```

```
## [1] 1 2 3 4 5
```

```
> y = sqrt(x)
> print(y)
```

```
## [1] 1.000 1.414 1.732 2.000 2.236
```

```
> x+y
```

```
## [1] 2.000 3.414 4.732 6.000 7.236
```

```
> x/y
```

```
## [1] 1.000 1.414 1.732 2.000 2.236
```

```
> pi
```

```
## [1] 3.142
```

- Op?ration d'assignation et regroupement d'op?rations

Assigner signifie donner une valeur ? une variable. C'est une op?ration classique en programmation, mais un des gros avantages de R est que l'on est pas oblig? de lui donner le type d'une variable pour qu'il l'interpr?te.

```
> x = 10 ; print(x)
```

```
## [1] 10
```

```
> x <- 10 ; x
```

```
## [1] 10
```

```
> {
+   x = 1:20 ; x
+   y = x**2
+   print(y)
+ }
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256 289
## [18] 324 361 400
```

Pour connaitre le type d'nue variable, on utilise l'instruction class

```
> x =10
> class(x)
```

```
## [1] "numeric"
```

```
> x = letters[1:5]
> class(x)
```

```
## [1] "character"
```

```
> xl = list(a=rnorm(10), b= letters, c= as.factor(c(1,3)))
> class(xl)
```

```
## [1] "list"
```

```
> lapply(xl, class)
```

```
## $a
## [1] "numeric"
##
## $b
## [1] "character"
##
## $c
## [1] "factor"
```

On reviendra plus loin sur les fonctions ci-dessous

- Nommage & Conventions

Les caract?res permis pour les noms d'objets sont les lettres minuscules. Selon l'environnement linguistique de l'ordinateur, il peut ?tre permis d'utiliser des lettres accentu?es, mais cette pratique est fortement d?courag?e puisqu'elle risque de nuire ? la portabilit? du code.

- Les noms d'objets ne peuvent commencer par un chi???re. S'ils commencent par un point, le second caract?re ne peut ?tre un chi???re.
- R est sensible ? la casse. delta, Delta, DELTA sont trois objets diff?rents. ma recommandation est de toujours **Utiliser toujours les minuscules**
- Comme dans tous les langages, certains mots sont r?serv?s : mean, Pi,t,sd, etc. . .

- Quelques variables importantes et op?rations de bases

Il est d'usage de laisser certians mots cl?s au langage. R ne fait pas exception :

```
> T
```

```
## [1] TRUE
```

```
> F
```

```
## [1] FALSE
```

```
> # A compl?ter
```

Soit un vecteur vec

```
> vec = rnorm(10,mean = 3,sd = 2)
> vec +1
```

```
##  [1] 1.5579 0.9158 2.5284 5.5011 4.8178 3.9172 5.4922 2.7794 4.9724 4.1288
```

```
> length(vec)
```

```
## [1] 10
```

```
> mode(vec)
```

```
## [1] "numeric"
```

Consid?rons un autre vecteur

```
> vec = c("classe","sig","log","tmp","plus","rennes")
> print(vec)
```

```
## [1] "classe" "sig"    "log"    "tmp"    "plus"   "rennes"
```

```
> vec +1
```

```
## Error: non-numeric argument to binary operator
```

```
> length(vec)
```

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

```
> mode(vec)
```

```
## [1] "character"
```

```
> nchar(vec)
```

```
## [1] 6 3 3 3 4 6
```

Consulter l'aide se fait de plusieurs fa?ons :

```
> help(matrix)
> ?apply
> ?mode
> vignette()
```

- Matrices et tableaux

R ?tant un langage sp?cialis? pour les calculs math?matiques, il supporte tout naturellement et de mani?re intuitive - ? une exception pr?s, comme nous le verrons - les matrices et, plus g?n?ralement, les tableaux ? plusieurs dimensions. Je ne parlerai pas de tableaux ici, mais on peut consulter l'aide sur les arrays.

Une matrice est un vecteur avec un attribut dim de longueur 2. Cela change implicitement la classe de l'objet pour "matrix" et, de ce fait, le mode d'affichage de l'objet ainsi que son interaction avec plusieurs op?rateurs et fonctions.

```
> mat = matrix(1:10, nrow = 5,ncol = 2, byrow = T)
> print(mat)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

```
> rbind(mat,1:2)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
## [6,]    1    2
```

```
> cbind(mat,2:6)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    2
## [2,]    3    4    3
## [3,]    5    6    4
## [4,]    7    8    5
## [5,]    9   10    6
```

```
> # Produit matriciel  !
> mat * mat
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    9   16
## [3,]   25   36
## [4,]   49   64
## [5,]   81  100
```

```
> # Notez la diff?rence
> mat %*% t(mat)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    5   11   17   23   29
## [2,]   11   25   39   53   67
## [3,]   17   39   61   83  105
## [4,]   23   53   83  113  143
## [5,]   29   67  105  143  181
```

- Les listes

La liste est sans doute l'objet le plus int?ressant de R. On peut tout y mettre. Regardez par exemple :

```
> myList <- list(team = "Ioda Team", df = data.frame(logs=c("google","yahoo"),nbclic = c(1,2)),dim = c(
> print(myList)
```

```
## $team
## [1] "Ioda Team"
##
## $df
##     logs nbclic
## 1 google      1
## 2  yahoo      2
##
## $dim
## [1] 1 2 4
##
## $x
## [1] "a" "b" "c" "d" "e"
```

Une liste peut donc contenir des donn?es de type diff?rents et cela ne pose (? priori) aucun probl?me.

L'acc?s ? des ?l?ments d'une liste se fait par le [[]] lorsqu'on veut l'?l?ment correspondant ? une position donn?e. Sinon, cela reste un vecteur comme un autre, et myList[1] donne le r?sultat que l'on imagine : le premier ?l?ment de la liste

```
> # Premier ?l?ment de la liste
> myList[[1]]
```

```
## [1] "Ioda Team"
```

```
> # Deuxi?me ?l?ment
> myList[2]
```

```
## $df
##     logs nbclic
## 1 google      1
## 2  yahoo      2
```

```
> # Appel par ?tiquette
> myList$df
```

```
##     logs nbclic
## 1 google      1
## 2  yahoo      2
```

- Les data.frame

S'il y a un langage qui a popularis? la notion de dataframe, c'est bien R. Repris aujourd'hui par quasiment tous les langages dans plusieurs environnements (dont r?cemment Python Pandas). J'ai m?me entendu les personnes utilisant SAS reprendre ce vocable et pourtant s'il y a bien quelque chose de plus ?loign? d'un data.frame c'est la mani?re dont SAS stocke les data.frame.

- Un data frame est une liste de classe “data.frame” dont tous les ?l?ments sont de la m?me longueur (ou comptent le m?me nombre de lignes si les ?l?ments sont des matrices).
- Il est g?n?ralement repr?sent? sous la forme d’un tableau ? deux dimensions. Chaque ?l?ment de la liste sous-jacente correspond ? une colonne.
- Bien que visuellement similaire ? une matrice un data frame est plus g?n?ral puisque les colonnes peuvent ?tre de modes diff?rents ; pensons ? un tableau avec des noms (mode character) dans une colonne et des notes (mode numeric) dans une autre.
- On cr?e un data frame avec la fonction data.frame ou, pour convertir un autre type d’objet en data frame, avec as.data.frame.
- Le data frame peut ?tre indic? ? la fois comme une liste et comme une matrice.

A retenir : C’est quasiment l’?l?ment central de l’analyse et de l’exploration math?matique/statistique avec R

```
> df<- data.frame(letters = letters[1:10], stat = rnorm(10), indices = sample(letters,10))
> print(df)
```

```
##    letters     stat indices
## 1        a -0.04087       e
## 2        b -0.28216       i
## 3        c  0.33505       l
## 4        d -0.49954       z
## 5        e -0.15850       d
## 6        f  1.54667       w
## 7        g  0.86902       f
## 8        h -0.99616       k
## 9        i  0.63711       j
## 10       j -2.92751       n
```

- Manipulations basiques

Un data.frame est une matrice. Les op?rations d’indicage sont donc comme dans le cas des matrices

```
> # S?lectionner les deux premi?res colonnes
> df[,c(1,2)]
```

```
##    letters     stat
## 1        a -0.04087
## 2        b -0.28216
## 3        c  0.33505
## 4        d -0.49954
## 5        e -0.15850
## 6        f  1.54667
## 7        g  0.86902
## 8        h -0.99616
## 9        i  0.63711
## 10       j -2.92751
```

```
> # s?lectionner les 3 premi?res lignes
> df[1:3,]
```

```
##   letters     stat indices
## 1       a -0.04087       e
## 2       b -0.28216       i
## 3       c  0.33505       l
```

```
> # S?lectionner par les noms des variables
> df[,c("stat","indices")]
```

```
##        stat indices
## 1  -0.04087       e
## 2  -0.28216       i
## 3   0.33505       l
## 4  -0.49954       z
## 5  -0.15850       d
## 6   1.54667       w
## 7   0.86902       f
## 8  -0.99616       k
## 9   0.63711       j
## 10 -2.92751       n
```

```
> # Op?rations logiques
> df[,df$stat>0]
```

```
## Error: undefined columns selected
```

```
> #Op?rations matricielles
> dim(df)
```

```
## [1] 10  3
```

- Quelques fonctions ? connaitre pour la manipulation des data.frame

```
> # Premiers ?l?ments
> head(df)
```

```
##   letters     stat indices
## 1       a -0.04087       e
## 2       b -0.28216       i
## 3       c  0.33505       l
## 4       d -0.49954       z
## 5       e -0.15850       d
## 6       f  1.54667       w
```

```
> # r?sum? statistique d'un df
> sumary(df)
```

```
## Error: could not find function "sumary"
```

```
> # Extraction ?l?gante d'informations
> subset(df,stat >=0.5)
```

```
##   letters   stat indices
## 6       f 1.5467       w
## 7       g 0.8690       f
## 9       i 0.6371       j
```

```
> # Nom des variables d'un df
> names(df)
```

```
## [1] "letters" "stat"    "indices"
```

```
> # Dimension d'un data.frame
> dim(df)
```

```
## [1] 10  3
```

- Quelques fonctions utiles ? la manipulation des donn?es (c'est vrai pour les df comme pour tout le reste)

```
> # G?n?rer une suite de nombre
> seq(from = 1,to = 50,by = 5) ; 1:10
```

```
##  [1]  1  6 11 16 21 26 31 36 41 46
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
> # R?p?tition de valeurs
> rep(5)
```

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

```
> # Trier
> sort(df$stat)
```

```
##  [1] -2.92751 -0.99616 -0.49954 -0.28216 -0.15850 -0.04087  0.33505
##  [8]  0.63711  0.86902  1.54667
```

```
> # Elements unique d'un vecteur
> unique(c(1,1,4,3,2,1,4,6))
```

```
## [1] 1 4 3 2 6
```

```
> # Recherche des ?l?ments selon une condition dans un vecteur
> vec = c(1,2,7,10,4,-1,-6,2)
> vec[vec<0]
```

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

```
> # Max, min, moyenne
> max(vec) ; min(vec); mean(vec)
```

```
## [1] 10
```

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

```
## [1] 2.375
```

```
> # Si on ne veut que la position
> which.max(vec)
```

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

```
> # Position de la premi?re occurence d'un ?l?ment
> match(2,vec)
```

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

```
> # Appartenance ensembliste
> 1 %in% vec
```

```
## [1] TRUE
```

```
> # Arrondi
> round(2.34567,2)
```

```
## [1] 2.35
```

```
> floor(3.98)
```

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

```
> # Op?rations math?matiques basiques
> {vec = rnorm(10,2,4) ;
+  print (vec)
+  }
```

```
##  [1]  5.87766  2.21409  0.31956  1.16424 -0.06763 -0.88665  3.53294
##  [8] 10.00264 -6.09166  7.68585
```

```
> #Somme, Produit
> sum(vec) ;prod(vec)
```

```
## [1] 23.75
```

```
## [1] -480.4
```

```
> #Statistiques simples
> sd(vec) ; var(vec) ; range(vec); median(vec) ; quantile(vec) ; cumsum(vec)
```

```
## [1] 4.643
```

```
## [1] 21.56
```

```
## [1] -6.092 10.003
```

```
## [1] 1.689
```

```
##       0%      25%      50%      75%     100%
## -6.09166  0.02917  1.68917  5.29148 10.00264
```

```
##  [1]  5.878  8.092  8.411  9.576  9.508  8.621 12.154 22.157 16.065 23.751
```

```
> mat <- matrix(rnorm(12),3,4)
> mat
```

```
##         [,1]     [,2]      [,3]    [,4]
## [1,]  1.9248  0.49101 -0.418817 -1.1777
## [2,]  0.5407  0.07464 -0.233375  0.3980
## [3,] -0.0766 -0.79543 -0.004543  0.1691
```

```
> # somme sur les lignes
> rowSums(mat)
```

```
## [1]  0.8193  0.7800 -0.7075
```

```
> # Transpos?e
> t(mat)
```

```
##         [,1]     [,2]      [,3]
## [1,]  1.9248  0.54066 -0.076604
## [2,]  0.4910  0.07464 -0.795433
## [3,] -0.4188 -0.23338 -0.004543
## [4,] -1.1777  0.39803  0.169112
```

Quelques exemples d'utilisations des structures de boucles

```
> # for
> vec = seq(2,10,2)
> for(i in vec) print(1:i)
```

```
## [1] 1 2
## [1] 1 2 3 4
## [1] 1 2 3 4 5 6
## [1] 1 2 3 4 5 6 7 8
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
> for(n in c(2,5,10,20,50)) {
+    x <- stats::rnorm(n)
+    cat(n, ": ", sum(x^2), "\n", sep = "")
+ }
```

```
## 2: 2.422
## 5: 5.95
## 10: 13.77
## 20: 17.95
## 50: 65.16
```

```
> #if
> #to do
```

Bien que je n'utilise (presque) jamais de structures de la sorte, la fonction ifelse est vectorizable et de fait, elle est tr?s agr?able ? utiliser

```
> vec = rnorm(5,1,3)
> ifelse(vec>2,"sup ? 2","inf ? 2")
```

```
## [1] "sup ? 2" "inf ? 2" "inf ? 2" "inf ? 2" "inf ? 2"
```

Si on avait voulu faire cette fonction en utilisant for, on aurait ?crit

```
> vec = rnorm(5,1,3)
> j=c()
> for(i in 1:length(vec))
+   if (vec[i]>2) j[i] <- 'sup ? 2' else j[i] <- 'inf ? 2'
> print(j)
```

```
## [1] "inf ? 2" "inf ? 2" "inf ? 2" "sup ? 2" "inf ? 2"
```

Voici sans doute l'une des forces du langage : Une vraie structure math?matique vectorizable

- Elements de base de la programmation

R est un langage de programmation, mais **Oubliez les structures conditionelles** : if, for, while, etc... Elles existent mais sont en r?alit? tr?s peu utilis?es dans la programmation en R. Pourquoi? Parce que l'?l?ment de base du langage est un vecteur et qu'il est plus commode d'utiliser des op?rateurs vectorizables pour appliquer une fonction sur tous les ?l?ments d'un tableau que de le parcourir un ? un

Les fonctions en R ont la structure suivante.

```
> # Une fonction simple
> f = function(arg1,arg2=2)
+   {
+   return(arg1*arg2)
+   }
> f(2,9)
```

```
## [1] 18
```

Bien entendu, une fonction est en g?n?ral plus complexe et peut m?me comme dans beaucoup de langage vite devenir illisible. Il faut dont toujours garder ? l'esprit qu'une fonction doit rendre service et la construire sur le principe input =>output

Quelques fonctions ? essayer de comprendre pour ?tre sur de comprendre le r?sultat final.

Notez bien : Une fonction R peut retourner n'importe quel type connu de R

```
> # installer automatiquement les packages
> packages<-function(x){
>   x<-as.character(match.call()[[2]])
>   if (!require(x,character.only=TRUE)){
>     install.packages(pkgs=x,repos="http://cran.r-project.org")
>     require(x,character.only=TRUE)
>   }
> }
>
> recherchesAcronymes <- function(dat, ncar=5, sw=tm::stopwords("french"), typeDistance="osa")
> {
>   tmp = dplyr::select(dat)
>   tmp = subset(tmp, (reformulation_plus_frequente!= "NULL"))
>   tmp = subset(tmp, nb_mots>=3)
>   library(tm)
>   firstLetter <-function(str)
>   {
>     tmpx = gsub(pattern="[[:punct:]]",replacement=" ",x=str)
>     tmpx= unlist(strsplit(tmpx,c(" ","'")))
>     tmpx = tmpx[!tmpx %in% sw]
>     first=paste(substr(tmpx,1,1),sep="",collapse="")
>     first= gsub(pattern="?","e",first)
>     return(first)
>   }
>   tmp$firstLetter = sapply(tmp$quiquoi_plus_frequent,firstLetter)
>   library(stringdist)
>   tmp$lev= stringdist(tmp$reformulation_plus_frequente,tmp$firstLetter,typeDistance)
>   library(plyr)
>   tmpx = arrange(tmp,lev)
>   acronymes <- subset(tmpx, lev<=2)
>   return(acronymes)
>   }
```

J'invite ? consulter les manules d'introduction ? R pour avoir un bagage ?tendu sur les concepts de base de R. En r?alit?, le langage R est tellement mouvant qu'il serait etr?s rare aujourd'hui de voir tous ce que je viens ici d'?noncer. Comme on le verra plus loin, plusieurs libaries encapsulent aujourd'hui ces notions de manipulations basiques et font parfois perdre de vue que R est avant tout un langage m?ta-math?matique. C'est une bonne chose pour tous ceux qui arrivent d'un autre langage vers R mais peut se r?vl?er d?sastreux dans la compr?hension des concepts sous-jacents. Attention donc !

Concepts avanc?s de R

Il y a une grande vari?t? de ce qu'on appelle en R des concepts avanc?s. Je n'en retiens que quelques uns et invite ? consulter les manuels de r?f?rence pour aller plus loin.

Concept 1 : Lecture et ?criture des fichiers R lit ? peu pr?s tous les formats de fichiers. La fonction de base de lecture est read.table dont les argument sont :

- file : fichier que l'on souhaite lire
- sep = "" : pr?cise le s?parateur (permet d'inclure des espaces ? l'int?rieur des champs). On peut aussi utiliser read.delim() qui est la m?me chose que read.table, mais avec comme s?parateur par d?faut).
- na.strings : indique les cha?nes de caract?res qui ont valeur de NA. Attention : si un champ vaut la cha?ne NA, il sera interpr?t?comme NA et non comme la cha?ne "NA" ! (mettre na.strings ? autre chose pour l'?viter)
- colClasses : force un type pour chaque colonne (numeric, factor, character, logical, Date, POSIXct).
- nrows : ne lit que les n premi?res lignes.
- as.is = TRUE : indique que les cha?nes doivent ?tre lues comme des cha?nes de caract?res et pas des facteurs.
- check.names = FALSE : ne change pas les noms des colonnes si celles-ci ne sont pas correctes (uniques et commen?ant par une lettre). Par d?faut, change les noms des colonnes pour avoir des noms valides (transforme 3 en X3 par exemple).
- encoding: format d'encodage du fichier
- header = T ou F : indique la pr?sence d'une ent?te ou non dans le fichier

Les formats de lecture, `read.csv, read.csv2`, etc pr?sentent les options similaires avec des options particuli?res en fonction du format du ficihier,

La fonction d'?criture `write.table` fonctionne sur le m?me principe.

Rendez-vous sur le Site aide m?moire de R pour tout apprendre sur la lecture et la manipulation des fichiers !

N?anmoins, quelques exemples :

```
> # Data ? lire
>
> logs_sle <- read.table("D:/Blog/Data_Friday/data/sle_time_series",header=F,
+                        col.names = c("timeStamp","nbAffichages","nbClics"),stringsAsFactors = F,sep="
> # Affichage des premiers ?l?ments
> head(logs_sle)
```

```
##             timeStamp nbAffichages nbClics
## 1 2014-06-02 15:00:00           99       1
## 2 2014-06-02 16:00:00          131       0
## 3 2014-06-02 17:00:00          131       0
## 4 2014-06-02 18:00:00          165       0
## 5 2014-06-02 19:00:00          124       0
## 6 2014-06-02 20:00:00           80       2
```

```
> # Affichage des infos du fichier
> str(logs_sle) ; summary(logs_sle)
```

```
## 'data.frame':    681 obs. of  3 variables:
##  $ timeStamp   : chr  "2014-06-02 15:00:00" "2014-06-02 16:00:00" "2014-06-02 17:00:00" "2014-06-02
##  $ nbAffichages: int  99 131 131 165 124 80 164 105 82 29 ...
##  $ nbClics     : num  1 0 0 0 0 2 0 0 0 0 ...
```

```
##   timeStamp          nbAffichages    nbClics
##  Length:681         Min.   :  1   Min.   : 0.00
##  Class :character   1st Qu.: 53   1st Qu.: 1.00
##  Mode  :character   Median :188   Median : 4.00
##                     Mean   :184   Mean   : 6.73
##                     3rd Qu.:286   3rd Qu.:10.00
##                     Max.   :542   Max.   :66.00
```

Par d?faut, lorsqu'un fichier est lu dans R, il est import? in-memory comme un data.frame. Il existe plusieurs autres fonctions pour lire et ?crire des fichiers : * scan() et read.lines() pour d?couper les valeurs ? extraire et les affecter ? un objet * Pour cr?er un ???chier au format voulu il faut cr?er les lignes voulues et utiliser la fonction cat() ou write.lines() et fermer le connecteur avec close. Citons aussi `readLines`, `read.delim2`, etc...

En tapant dans google : read files in R ou Import Data in R, on a une palanqu?e de fonctions possibles pour l'import des fichiers.

Concept 2 : Les librairies dans R Un package R est un ensemble coh?rent de fonctions, de jeux de donn?es et de documentation permettant de compl?ter les fonctionnalit?s du syst?me ou d'en ajouter de nouvelles. Les packages sont normalement install?s depuis le site Comprehensive R Archive Network. Normalement parce que depuis environ 3 ann?es, tout le monde peut d?velopper des librairies et les mettre ? disposition de la communaut? via `github`, `bioconductor` ou d'autres lieux de partages.

Notons que le caract?re open source de R le rend tr?s mouvant, mais le CRAN a vocation ? ?tre le lieu de r?f?rence pour les librairies. Lorsqu'une librairie est pr?sente sur le site du CRAN, alors elle est fiable parce que v?rifi?e ? partir de crit?res tr?s strictes et sa maintenance est "quasiment" assur?e dans le temps.

L'installation des librairies se fait par l'instruction :

install.packages()

et l'appel ? ceette librairie se fait pendant l'ex?cution d'un programme par :

library() ou encore require()

```
> # installez les librairies : ggplot2, dplyr, reshape2
> install.packages('dplyr',dep=T)
> library(dplyr)
```

Certaines librairies, **surtout les plus r?centes** ne sont pas disponibles sur le cran. Il faut installer une librairies d'outils ? partir du CRAN : devtools afin d'avoir acc?s aux repos de github ou bioconductor.

```
> install.packages("devtools")
> install_github("rCharts","ramnath")
```

Concept 3 : La fonction Apply et ses d?clinaisons l'un des concepts les plus innovants de la programmation en R est la fonction apply.

Elle n'est pas intuitive, mais lorsqu'on a pris le pied, on ne fait jamais de boucle en R et on l'utilise quasiment ? toutes les sauces. Je vais dans ce qui suit essayer d'en donner une intuition

Petit rappel :

L'?l?ment de base de R est un tableau (matrice, data.frame). Il a donc des lignes et des colonnes. La fonction apply() permet d'appliquer une fonction (par exemple une moyenne, une racine carr?, etc) ? chaque ligne ou chaque colonne d'un tableau de donn?es.Cette fonction prend 3 arguments dans l'ordre suivant:

- nom du tableau de donn?es
- un nombre pour dire si la fonction doit s'appliquer aux lignes (1), aux colonnes (2) ou aux deux (c(1,2))
- le nom de la fonction ? appliquer

Si l'on consid?re les logs_sle par exemple, on veut connaitre la moyenne du nombre d'affichages et du nombre de clics

```
> apply(logs_sle[,2:3],2,mean)
```

```
## nbAffichages      nbClics
##      184.338        6.728
```

La fonction peut ?tre plus complexe

```
> f = function(d){
+   return(min(d) +max(d))
+ }
> apply(logs_sle[,2:3],2,f)
```

```
## nbAffichages      nbClics
##          543           66
```

L'avantage des fonctions apply est d'?viter de faire des boucles. Dans la programmation de tous les jours, ? chaque fois que vous pensez ? une boucle, dites vous que le apply fait l'affaire.

A retenir :

> Apply permet d'effectuer dans un tableau de donn?es les m?mes op?rations sur des lignes ou sur des colonnes
>
> Apply a plusiuers d?clinaisons : sapply, lapply, vapply, mapply

Je renvois au site de http://www.manio.org/blog/understanding-the-apply-functions-and-then-others-in-by-asking-questions/ pour une compr?hension approfondie de la fonction Apply ? travers des questions et des exemples ou encore ? l'excellent Blog de saunders pour une introduction approfondie.

Quelques exemples d'utilisation des fonctions apply

```
> list = list(df =head(logs_sle[,2:3]), vec = rnorm(10))
> lapply(list, mean)
```

```
## Warning: argument is not numeric or logical: returning NA
```

```
## $df
## [1] NA
##
## $vec
## [1] 0.2772
```

```
> attach(iris)
> # Moyenne des longueurs de p?tales par esp?ces
> tapply(iris$Petal.Length, Species, mean)
```

```
##     setosa versicolor  virginica
##      1.462      4.260      5.552
```

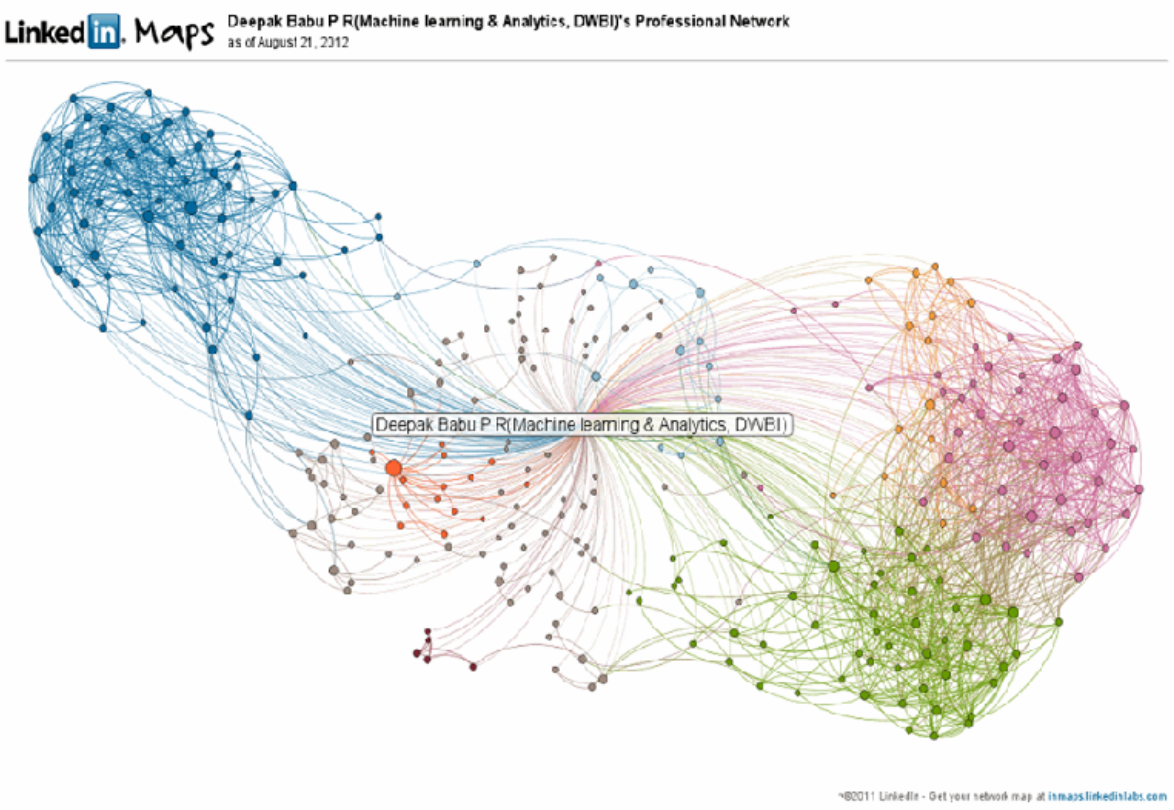
```
> l <- list(a = 1:10, b = 11:20)
> # Moyenne des valeurs d'une listes de numeric
> l.mean <- sapply(l, mean) : l.mean
```

```
## Error: object 'l.mean' not found
```

```
> # Quel type d'objet est retourn??
> class(l.mean)
```

```
## Error: object 'l.mean' not found
```

Concept 4 : Graphiques avec R La richersse de graphiques avec R est incomparable. Un exemple est cette mangnifique repr?sentation d'un clustering de r?seau social :



13 - 17 Avant Midi

17 - 20 Apres 20

ou encore ces comparisons cloud

la librairie qui a r?volutionn? la grammaire graphique dans un langage de haut niveau est *ggplot2* d?velopp? par Hadley Wickham, membre actif de la communaut? R.

On peut quasiment tout faire avec cette librairie. Tous les types de graphiques comme nous verrons plus loin.

- Quelques graphiques classiques

Nous utiliserons pour l'illustration le fichier sle_clicks qui contient les informations d'AB Testing, partenaire, type de page, type de clic, ou encore position du listing cliqu?, position de l'iframe dans la page du partenaire; pour les clics du mois de juin ou le fichier sle pr?c?demment vu

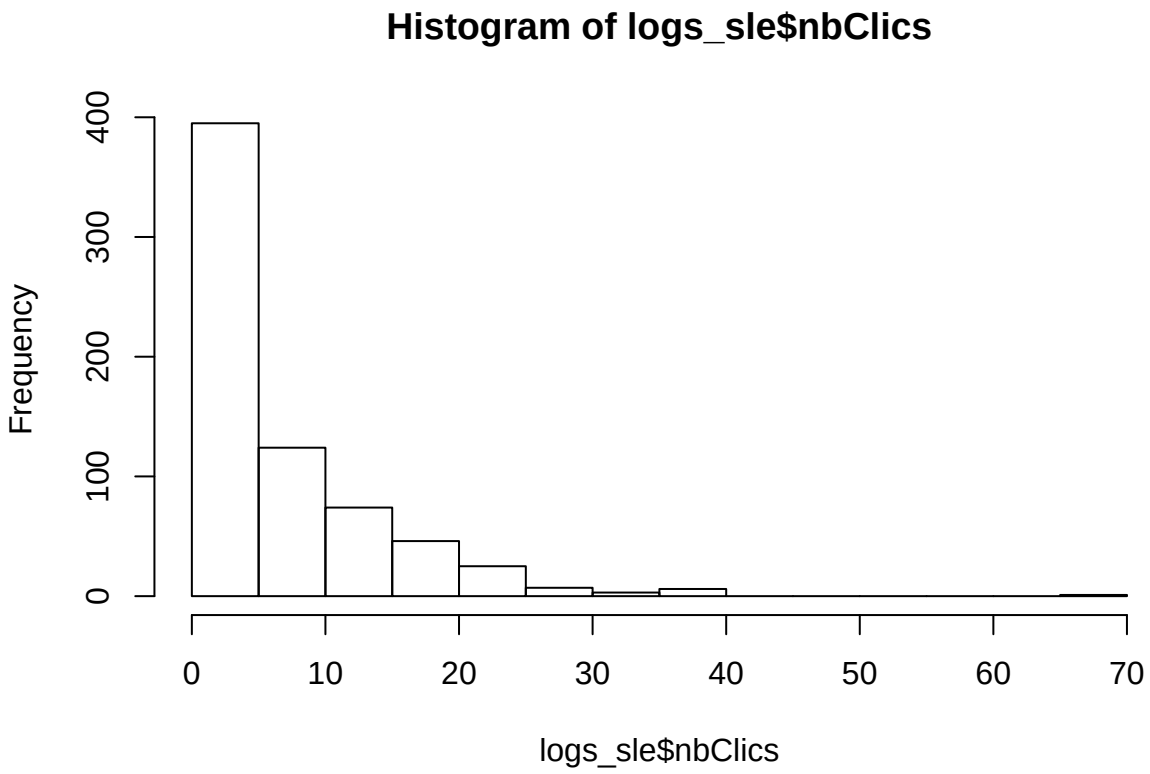
1. Histogramme : Graphique permettant de repr?senter la r?partition d'une variable continue. C'est le graphique le plus intuitif pour toutes personnes qui se frottent ? l'analyse des donn?es

```
> sle_clicks <-read.csv("./data/sle-clicks.csv",header = T,sep = ";",dec=".")
> # Premiers ?l?ments pour explorer le fichier
> head(sle_clicks)
```

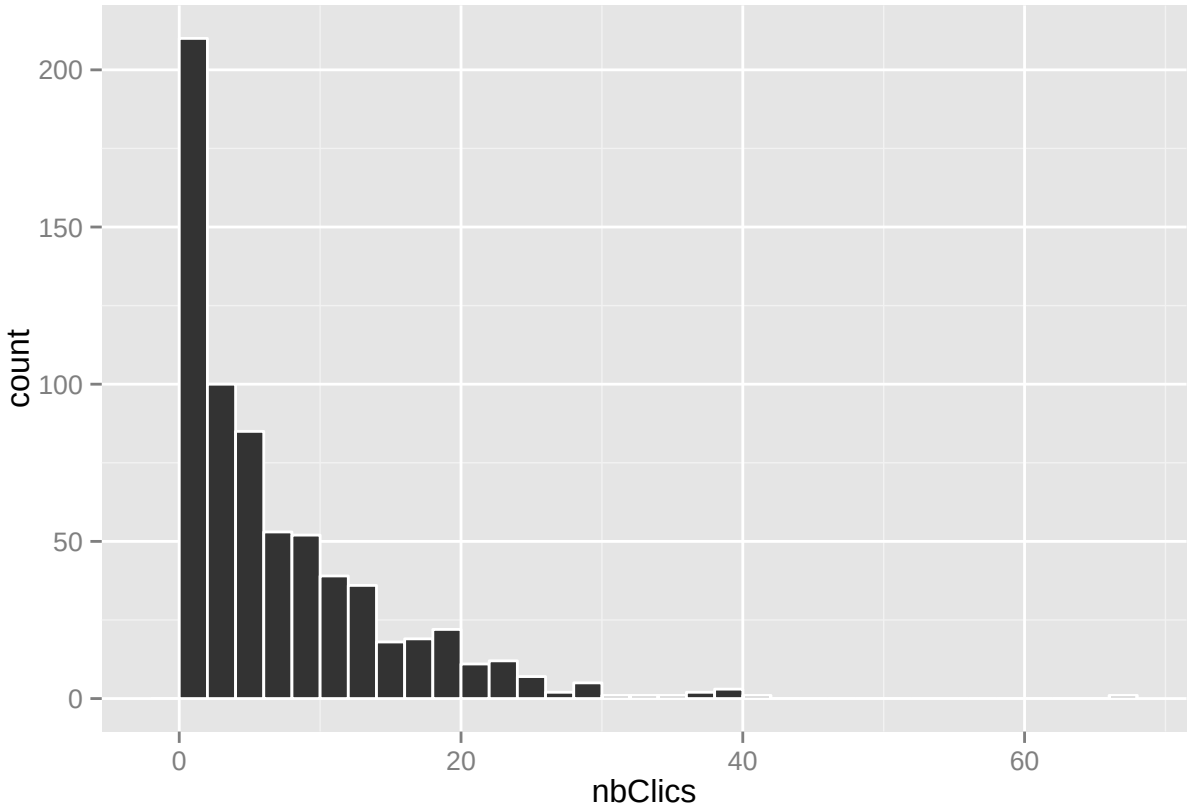
```
##          timestamp kameleoon_x atpartner atpagetype position eventtype
## 1 02/06/2014 20:58             Lesrestos         FD        1  clickurl
## 2 02/06/2014 20:58             Lesrestos         FD        1 dispphone
## 3 03/06/2014 04:14             Lesrestos         FD        3  clickurl
## 4 03/06/2014 10:55                 Local         FD        2 dispphone
## 5 03/06/2014 10:55                 Local         FD        1 dispphone
## 6 03/06/2014 10:55                 Local         FD        1  clickurl
##           target posleft postop
```

```
## 1            LVS       0      0
## 2       Fantomas       0      0
## 3            LVS       0      0
## 4       Fantomas     151    195
## 5       Fantomas     151    195
## 6 Raison_Sociale     151    195
```

```
> logs_sle <- read.table("D:/Blog/Data_Friday/data/sle_time_series",header=F,
+                        col.names = c("timeStamp","nbAffichages","nbClics"),stringsAsFactors = F,sep="
>
> # Distribution du nombre de clics avec les fonctions de basse de R
> hist(logs_sle$nbClics)
> # Distribution du nombre de clics avec la biblioth?que ggplot2
> library(ggplot2)
```



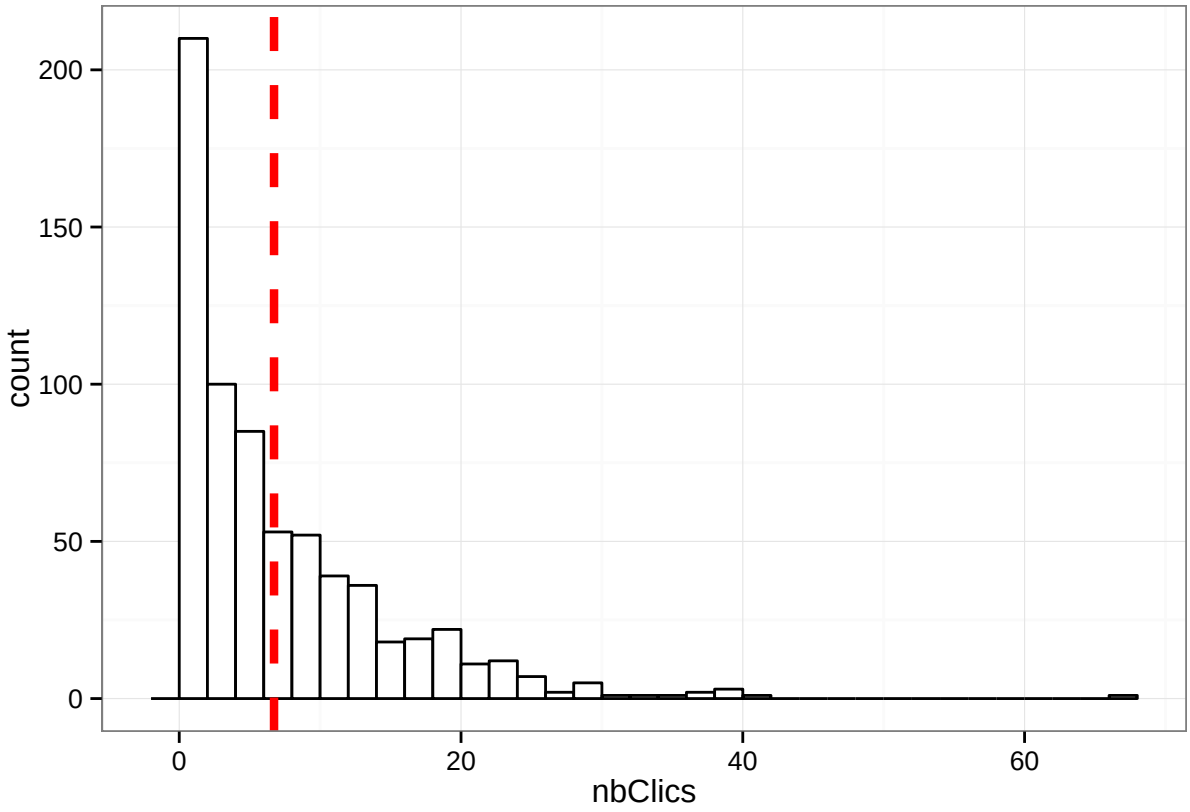

```
> ggplot(logs_sle, aes(x=nbClics)) + geom_histogram(binwidth=2, fill="gray20",colour="white")
```



Toute la richesse de cette biblioth?que est la personnalisation de graphiques symbolis?s par des couches que l'on superpose au plan initial. le symb?le + indique le rajout de couche

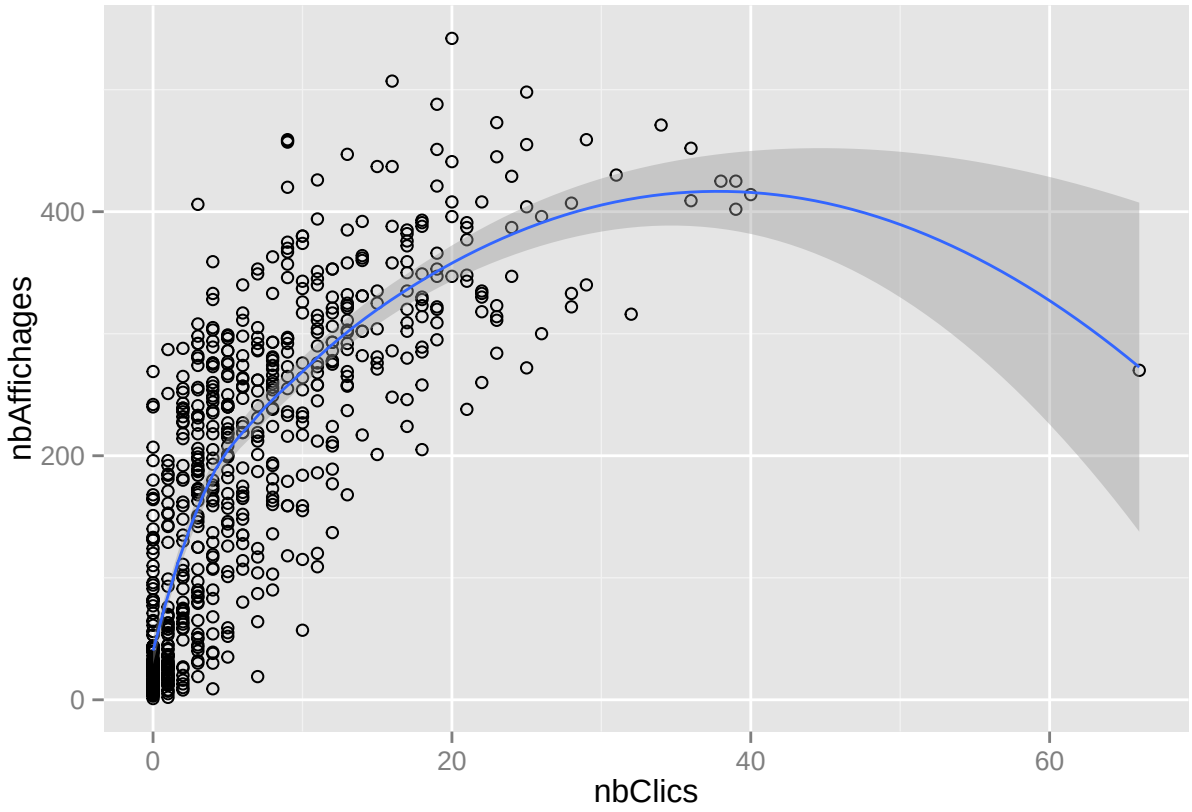
Exemple de personnalisation : Je souhaite voir o? se situe la moyenne dans la distribution

```
> ggplot(logs_sle, aes(x=nbClics)) + geom_histogram(binwidth=2, colour="black", fill="white") + theme_bw
+     geom_vline(aes(xintercept=mean(logs_sle$nbClics, na.rm=T)),color="red", linetype="dashed", size=1
```



2. Graphique de dispersion : G?n?ralement, on souhaite voir comment deux grandeurs ?voluent ensemble, on utilise alors des nuages de dispersions

```
> ggplot(logs_sle, aes(x=nbClics, y=nbAffichages)) +
+     geom_point(shape=1) +    #Cercle creux
+     geom_smooth(method=loess,   # Ajouter une courbe de tendance bas?e sur la m?thode lowess
+                 se=TRUE)    # Colorer l'intervalle de confiance
```



Dans ce graphique, globalement, on peut voir que nbAffichages = sqrt(nbClics). Autrement dit si on augmente l'affichage de 2, le nombre de clics augmente de 4

3. S?ries temporelles Si l'on veut par exemple visualiser par heure/jour/ le nombre d'affichages, cela est possible. Mais avant, un petit interm?de est indispensable **La gestion de la date est comme dans tous les langages un exercice tr?s difficile**

Une doc bien faite pour apprendre ? g?rer les dates en R est http://en.wikibooks.org/wiki/R_Programming/Times_and_Dates. j'invite quiconque aura ? g?rer les dates dans un projet ? s'en inspirer.

```
> # Quel est le format des variables dans mon fichier
> str(logs_sle)
```

```
## 'data.frame':    681 obs. of  3 variables:
##  $ timeStamp   : chr  "2014-06-02 15:00:00" "2014-06-02 16:00:00" "2014-06-02 17:00:00" "2014-06-02 
##  $ nbAffichages: int  99 131 131 165 124 80 164 105 82 29 ...
##  $ nbClics     : num  1 0 0 0 0 2 0 0 0 0 ...
```

timeStamp a un format character et pourtant, il s'agit d'une date au format datetime. On va donner ? R cette information

```
> # Modification du format du fichier
> # Quelle est ma zone g?o
> Sys.timezone()
```

```
## [1] "Europe/Paris"
```

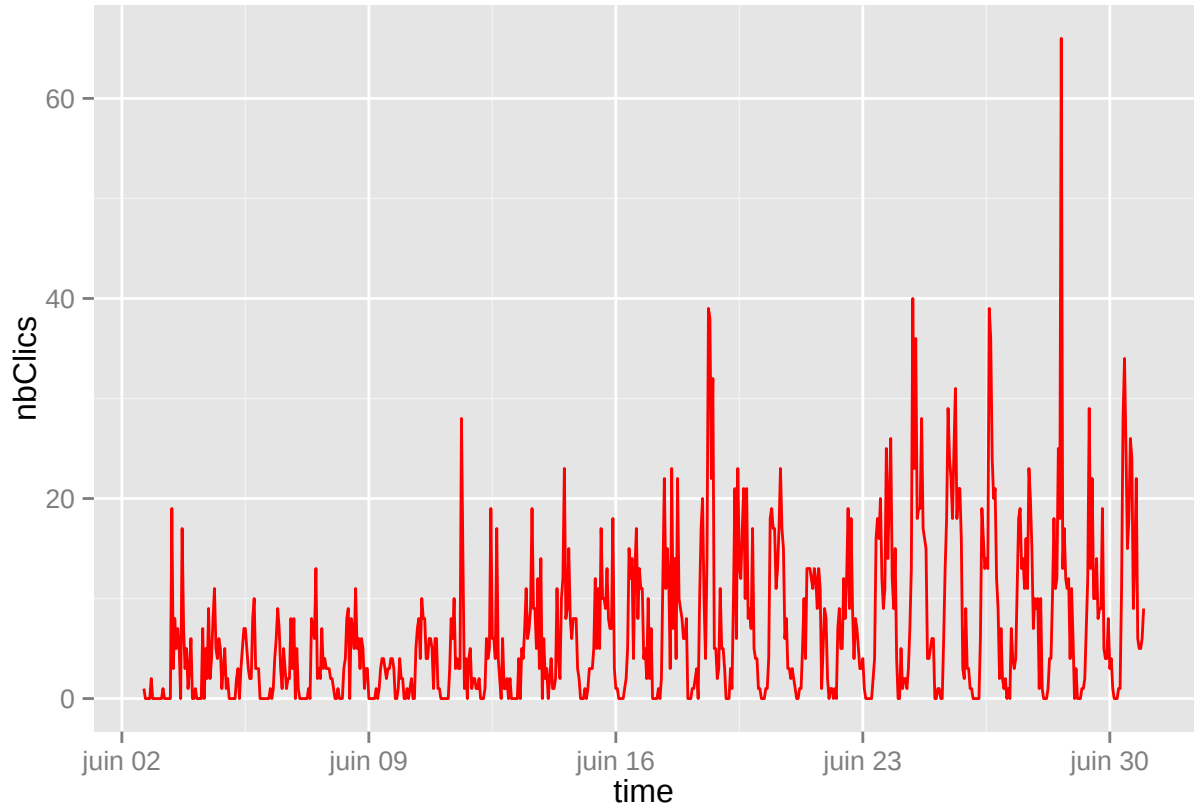
```
> #logs_sle$newtimeStamp = as.Date(strftime(as.POSIXct(logs_sle$timeStamp), format="%Y-%m-%d %H:%M:%S",
> logs_sle$newtimeStamp = as.POSIXct(x = logs_sle$timeStamp,tz = "EST")
> str(logs_sle)
```

```
## 'data.frame':    681 obs. of  4 variables:
##  $ timeStamp   : chr  "2014-06-02 15:00:00" "2014-06-02 16:00:00" "2014-06-02 17:00:00" "2014-06-02 
##  $ nbAffichages: int  99 131 131 165 124 80 164 105 82 29 ...
##  $ nbClics     : num  1 0 0 0 0 2 0 0 0 0 ...
##  $ newtimeStamp: POSIXct, format: "2014-06-02 15:00:00" "2014-06-02 16:00:00" ...
```

```
> # On peut biensur extraire les heures et les minutes par leur positions
> logs_sle$time = substr(logs_sle$newtimeStamp, 12, 16)
> # Ou encore et de mani?re plus native
> logs_sle$time <- as.POSIXct(logs_sle$newtimeStamp, format = "%H")
> head(logs_sle)
```

```
##             timeStamp nbAffichages nbClics        newtimeStamp
## 1 2014-06-02 15:00:00           99       1 2014-06-02 15:00:00
## 2 2014-06-02 16:00:00          131       0 2014-06-02 16:00:00
## 3 2014-06-02 17:00:00          131       0 2014-06-02 17:00:00
## 4 2014-06-02 18:00:00          165       0 2014-06-02 18:00:00
## 5 2014-06-02 19:00:00          124       0 2014-06-02 19:00:00
## 6 2014-06-02 20:00:00           80       2 2014-06-02 20:00:00
##                   time
## 1 2014-06-02 15:00:00
## 2 2014-06-02 16:00:00
## 3 2014-06-02 17:00:00
## 4 2014-06-02 18:00:00
## 5 2014-06-02 19:00:00
## 6 2014-06-02 20:00:00
```

```
> # Le graphique avec la fonction ggplot
> ggplot(logs_sle, aes(x=time, y=nbClics)) + geom_line(colour="red")
```



Une documentation compl?e de ggplot2 est disponible ici : http://docs.ggplot2.org/0.9.3.1

Attention :

Notez que cette biblioth?que graphique demande beaucoup d'entrainement pour une prise en main op?rationnelle. Ne pas h?sitez ? consulter les forums et l'aide en ligne pour personnaliser les fonctions. Ici, nous n'avons abord? qu'une infime partie de ce qui est possible ? partir de cette librairie.

Rappelons quand m?me que R dispose de fonctions de base pour les repr?sentations graphiques. `plot` qui contient des possibilit?s tr?s int?ressantes et plus avanc?es que tous les autres langages sans ajout de libairies compl?mentaires

Quelques fonctions d'un niveau 'un peu' avanc?es : melt, cast, rCharts

melt et cast (Excel am?lior?) To do

rCharts : d3 dans R To do

Shiny & co : Rstudio, html5 To do

R et Hadoop To do

R versus Python To do

Etude de cas compl?te : Exploration d'un fichier de bout en bout avec R

je propose ici de r?aliser une phase exploratoire compl?te avec R. De l'importation de fichier, ? la r?alisation de graphiques simples et intuitifs en passant par les phases de tri, de fusion, de s?lection de filtre et de comptages statistiques sur les fichiers sources

Etape 1 : Lecture du fichier Comme vu pr?c?demment, la lecture d'un fichier se fait ? partir des commandes de bases du langage : `read.csv`, `read.delim`, `read.table`, `read.xlsx`,`scan`, `readlines`,etc...

Pour cette premi?re ?tape nous allons

- Lire des fichier
- Changer le noms de certaines variables
- Mettre les noms des variables en minuscule
- Cherchez en quelques minutes comment importer des fichiers ? partir d'une URL
- Afficher les premiers ?l?ments des fichiers et les diff?rentes types de variables disponibles
- Changer le format des dates

```
> # Lecture ? partir d'une URL
> url_sle <- "https://www.dropbox.com/s/qwlx1zhcylwuxo1/sle_time_series"
> url_at <- "https://www.dropbox.com/s/z2v22tkczchmc61/sle-clicks.csv"
> Sys.timezone()
> #logs_sle$newtimeStamp = as.Date(strftime(as.POSIXct(logs_sle$timeStamp), format="%Y-%m-%d %H:%M:%S",
> download.file(url_sle,destfile = "fromDropBox1.csv")
> logs_sle <- read.table(url_sle,header=F,col.names = c("timeStamp","nbAffichages","nbClics"),stringsAs
> head(logs_sle)
> names(logs_sle)
> names(logs_sle)[1] <- "timeStampChange"
> head(logs_sle)
> names(logs_sle)<-tolower(names(logs_sle))
> head(logs_sle)
> str(logs_sle)
```

Etape 2 : Manipulation avec dPlyr, plyr, reshape2 Des librairies plyr, dPlyr, reshape2 sont des librairies faites en particulier pour des personnes qui viennent d'autres lanagees (SQL, SAS) et qui leur fournit un ensemble de focntions pratiques ? la manipulation des donn?es.

Je pr?sente ici quelques unes et renvoit ? la documentation pour des informations compl?mentaires.

Mais avant d'entrer dans le d?tail de ces fonctions, notons que bien qu'elles apportent une simplification en apparence, elles masquent en g?n?ral la base du langage R et certains aujourd'hui l'utilisent principalement pour ces fonctions pr?tes ? l'emploi. Il est toujours important, voire n?cessaire d'avoir une compr?hension au del? de simples fonctions pr?packag?es

Pour d?couvrir quelques fonctions, nous allons effectuer les op?rations suivantes

- S?lectionner une partie seulement du dataframe

- S?lectionner quelques variables seulement
- R?aliser des jointures
- Cr?er de nouvelles variables ? partir d'anciennes
- Utiliser les fonctions apply, table, summary, by, pour calculer des statistiques simples et courantes en phase exploratoire

```
> # Appel des librairies
> library(dplyr);library(reshape2)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
> # Dataframe de test
> set.seed(123)
> df1 = data.frame(var1 = sample(letters,15,replace = T), var2 =rnorm(15,2,3),var3 = c("A","B","C"))
> df2 = data.frame(varx1 = rep(c("a","b","c"),5), varx2 =rnorm(15,0,3),var3 = c("A","B","C"))
> head(df1) ; head(df2)
```

```
##   var1   var2 var3
## 1    h  5.842    A
## 2    u -3.182    B
## 3    k  7.071    C
## 4    w  3.511    A
## 5    y  9.585    B
## 6    b  3.647    C
```

```
##   varx1   varx2 var3
## 1     a -3.2571    A
## 2     b -0.2563    B
## 3     c  3.2118    C
## 4     a -0.4362    A
## 5     b -3.4966    B
## 6     c -2.4555    C
```

```
> # > * S?lectionner une partie seulement du dataframe
> filter(df1,var2>4)
```

```
##   var1  var2 var3
## 1    h 5.842    A
## 2    k 7.071    C
## 3    y 9.585    B
## 4    o 5.884    C
## 5    l 4.477    A
```

```
> filter(df2, var3=="A")
```

```
##   varx1   varx2 var3
## 1     a -3.2571    A
## 2     a -0.4362    A
## 3     a  2.0548    A
## 4     a -1.7988    A
## 5     a -0.4542    A
```

```
> filter(df2, (var3=="A" & varx2<0.2))
```

```
##   varx1   varx2 var3
## 1     a -3.2571    A
## 2     a -0.4362    A
## 3     a -1.7988    A
## 4     a -0.4542    A
```

```
> # > * S?lectionner quelques variables seulement
> select(df1,var1,var2)
```

```
##    var1    var2
## 1     h  5.8417
## 2     u -3.1818
## 3     k  7.0706
## 4     w  3.5114
## 5     y  9.5850
## 6     b  3.6473
## 7     n  2.7146
## 8     x -1.1467
## 9     o  5.8843
## 10    l  4.4766
## 11    y  1.8329
## 12    l -0.3531
## 13    r -0.2005
## 14    o  1.3524
## 15    c  0.9953
```

```
> select(filter(df1,var2>2),var1,var3)
```

```
##   var1 var3
## 1    h    A
## 2    k    C
## 3    w    A
## 4    y    B
## 5    b    C
## 6    n    A
## 7    o    C
## 8    l    A
```

```
> # > * R?aliser des jointures
> head(merge(df1,df2), 20)
```

```
##    var3 var1  var2 varx1   varx2
## 1     A    h 5.842     a -3.2571
## 2     A    h 5.842     a -1.7988
## 3     A    h 5.842     a  2.0548
## 4     A    h 5.842     a -0.4362
## 5     A    h 5.842     a -0.4542
## 6     A    l 4.477     a -3.2571
## 7     A    l 4.477     a -1.7988
## 8     A    l 4.477     a  2.0548
## 9     A    l 4.477     a -0.4362
## 10    A    l 4.477     a -0.4542
## 11    A    n 2.715     a -3.2571
## 12    A    n 2.715     a -1.7988
## 13    A    n 2.715     a  2.0548
## 14    A    n 2.715     a -0.4362
## 15    A    n 2.715     a -0.4542
## 16    A    w 3.511     a -3.2571
## 17    A    w 3.511     a -1.7988
## 18    A    w 3.511     a  2.0548
## 19    A    w 3.511     a -0.4362
## 20    A    w 3.511     a -0.4542
```

```
> head(inner_join(df1,df2),20)
```

```
## Joining by: "var3"
```

```
##    var1    var2 var3 varx1   varx2
## 1     h  5.8417    A     a -3.2571
## 2     w  3.5114    A     a -3.2571
## 3     n  2.7146    A     a -3.2571
## 4     l  4.4766    A     a -3.2571
## 5     r -0.2005    A     a -3.2571
## 6     u -3.1818    B     b -0.2563
## 7     y  9.5850    B     b -0.2563
## 8     x -1.1467    B     b -0.2563
## 9     y  1.8329    B     b -0.2563
## 10    o  1.3524    B     b -0.2563
## 11    k  7.0706    C     c  3.2118
## 12    b  3.6473    C     c  3.2118
## 13    o  5.8843    C     c  3.2118
## 14    l -0.3531    C     c  3.2118
## 15    c  0.9953    C     c  3.2118
## 16    h  5.8417    A     a -0.4362
## 17    w  3.5114    A     a -0.4362
## 18    n  2.7146    A     a -0.4362
## 19    l  4.4766    A     a -0.4362
## 20    r -0.2005    A     a -0.4362
```

```
> #left_join(df1,df2)
> # > * Cr?er de nouvelles variables ? partir d'anciennes
> # > * Utiliser les fonctions apply, table, summary, by, pour calculer des statistiques simples et cou
> table(df1$var1,df1$var3)
```

```
##
##     A B C
##   b 0 0 1
##   c 0 0 1
##   h 1 0 0
##   k 0 0 1
##   l 1 0 1
##   n 1 0 0
##   o 0 1 1
##   r 1 0 0
##   u 0 1 0
##   w 1 0 0
##   x 0 1 0
##   y 0 2 0
```

```
> summary(df2)
```

```
##  varx1     varx2         var3
##  a:5   Min.   :-9.682   A:5
##  b:5   1st Qu.:-2.856   B:5
##  c:5   Median :-0.454   C:5
##        Mean   :-1.214
##        3rd Qu.: 0.367
##        Max.   : 3.212
```

```
> apply(X = df1,2,length)
```

```
## var1 var2 var3
##   15   15   15
```

```
> apply(df2[,-2],2,as.factor)
```

```
##       varx1 var3
##  [1,] "a"   "A"
##  [2,] "b"   "B"
##  [3,] "c"   "C"
##  [4,] "a"   "A"
##  [5,] "b"   "B"
##  [6,] "c"   "C"
##  [7,] "a"   "A"
##  [8,] "b"   "B"
##  [9,] "c"   "C"
## [10,] "a"   "A"
## [11,] "b"   "B"
## [12,] "c"   "C"
## [13,] "a"   "A"
## [14,] "b"   "B"
## [15,] "c"   "C"
```

Conclusion partielle

J'ai essay? ici d'introduire les notions centrales de R en partant de ce que je consid?re comme les fondamentaux ? connaitre. L'aide en ligne est souvent trompeuse et masque mal la complexit? des op?rations ? l'oeuvre. Les biblioth?ques rapport?es sont indispensables, mais il faut toujours s'assurer de bien comprendre ce qui est fait.

J'ai par ailleurs insit? sur 4 librairies qui ont r?volutionn? la conception m?me de R ces deux derni?re ann?es, et sans se mettre ? jour r?guli?rement, on passe vite ? c?t? de l'?volution du langage. Ces quatre librairies sont tout de m?me tr?s sures et un projet vise m?me ? en faire des ?l?ments de base du langage.

N?anmoins, comme toute introduction, elle est partielle et partiale. Et parfois m?me, il y a certains partis pris sans doute li?s ? ma formation initiale de math?maticien et qui sont profond?ment discutables (du moins sur certaines choses que l'on attend d'un logiciel). La partialit? vient aussi du fait qu'il n'a pas du tout ?t? abord? (parce que le public ne s'y pr?te pas) le coeur de R ? travers la Data science d'une part et sa v?ritable r?volution autour de Shiny et de la mani?re dont la soci?t? Rstudio ont r?volutionn? l'approche de R d'autres parts. Par ailleurs, les possibilit?s graphiques sont aujourd'hui infinies avec Ramnath et la librairie rCharts, et le clou de la chose **Tout ce tutoriel a ?t? ?crit avec R** du d?but ? la fin.

? http://sciencendata.wordpress.com