R avancé

Data Table

Nicolas Klutchnikoff

2018-2019

Le package `data.table`

Le package `data.table`

Introduction

Objectifs

Pallier certains défauts des data-frames en étant :

- plus **rapide** dans la manipulation des données ;
- moins **gourmand** en utilisation mémoire.

Enfin, la **syntaxe** de manipulation des données est différente pour se rapprocher un peu des principes de manipulation de SQL.

La fonction `fread()` (1)

```
f_temp <- function(n){
  set.seed(1234)
  dt <- data.table(
    x = sample(1:1000, n, replace = TRUE),
    y = runif(n),
    z = rnorm(n),
    group1 = sample(c("A", "B"), n, replace=TRUE),
    group2 = sample(c("C", "D"), n, replace=TRUE)
  )
  write.csv(dt, file = "data/dt.csv")
  cat("Taille en (MB):", round(file.info("data/dt.csv")$size/1024^2),"\n")
}
```

La fonction `fread()` (2)

```
f_temp(100)
```

```
## Taille en (MB): 0
```

```
system.time(read.csv("data/dt.csv"))
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
system.time(fread("data/dt.csv"))
```

```
##    user  system elapsed
##   0.002   0.000   0.003
```

La fonction `fread()` (3)

```
f_temp(1e5)
```

```
## Taille en (MB): 5
```

```
a <- system.time(read.csv("data/dt.csv")); a
```

```
##    user  system elapsed
##   0.428   0.014   0.444
```

```
b <- system.time(fread("data/dt.csv")); b
```

```
##    user  system elapsed
##   0.039   0.002   0.035
```

Facteur d'accélération : 10.974359

La fonction `fread()` (4)

Sur cet exemple simple on voit les optimisations apportées à la fonction `fread()`.

- Plus *rapide* que la fonction de base `read.csv()`. C'est encore plus flagrant sur des gros jeux de données !
- Plus *intelligente* : reconnaissance du type de fichiers (ici `.csv`)

Très utilisée, même en dehors du package `data.frame`.

La bonne façon de faire : `data.table::fread()`.

Syntaxe de base

```
dt[i, j, by]
```

- `i` : sélection de lignes
- `j` calcul sur les colonnes (opération, sélection, manipulation, etc.)
- `by` : groupement par *catégories*

Pour la suite :

```
n <- 100
dt <- data.table(
  x = runif(n),
  y = rnorm(n),
  group1 = sample(c("A", "B"), n, replace=TRUE),
  group2 = sample(c("C", "D"), n, replace=TRUE)
)
```

Exemple typique

```
class(dt)
```

```
## [1] "data.table" "data.frame"
```

```
dt[group1 == "A", mean(x), by = group2]
```

```
##    group2        V1
## 1:      D 0.5401284
## 2:      C 0.5119101
```

Pour tous les individus dont la variable `group1` présente la modalité `A`, la moyenne de la variable `x` est calculée le long des différentes modalités de la variable `group2`.

Le résultat de l'opération est le **data-table** affiché.

Le package `data.table`

Filtrer des individus

Principe

Le filtrage sur les individus se fait en utilisant la partie `i` de la syntaxe. Il existe plusieurs façons de s'y prendre :

- en utilisant directement les numéros de lignes à sélectionner ;
- en utilisant un vecteur de booléens comportant TRUE si l'on souhaite sélectionner la ligne et FALSE sinon.

Numéros de lignes (1)

```
dt[2,,]
```

```
##           x         y group1 group2
## 1: 0.516298 0.2080352      B      D
```

Les virgules finales sont optionnelles :

```
dt[2]
```

```
##           x         y group1 group2
## 1: 0.516298 0.2080352      B      D
```

Numéros de lignes (2)

```
dt[c(1,5)]
```

```
##            x         y group1 group2
## 1: 0.2009551  1.741912      B      C
## 2: 0.4018199 -2.747668      B      C
```

```
dt[1:4]
```

```
##            x          y group1 group2
## 1: 0.2009551  1.7419116      B      C
## 2: 0.5162980  0.2080352      B      D
## 3: 0.6805172 -0.5880803      A      D
## 4: 0.1789758 -0.2117536      B      C
```

Numéros de lignes (3)

```
dt[order(y)][1:5]
```

```
##           x         y group1 group2
## 1: 0.4018199 -2.747668      B      C
## 2: 0.4541626 -2.220570      B      C
## 3: 0.4809908 -2.218247      B      D
## 4: 0.6464605 -2.170958      B      D
## 5: 0.7690887 -1.877262      A      D
```

Remarques :

- La fonction `order()` renvoie bien un vecteur d'indices ;
- Il a été inutile d'écrire `dt$y` à l'intérieur des crochets comme avec des data-frames ;
- Nous avons **chaîné** deux filtrages successifs.

Test (1)

Le principe est le suivant :

```
mytest <- c(TRUE, TRUE, rep(FALSE, 98))
dt[mytest]
```

```
##            x         y group1 group2
## 1: 0.2009551 1.7419116      B      C
## 2: 0.5162980 0.2080352      B      D
```

Le vecteur `mytest` possède 100 éléments booléens, autant que `dt` possède de lignes. Les lignes correspondant à la valeur TRUE sont sélectionnées.

Test (2)

```
dt[y>0][1:5]
```

```
##            x         y group1 group2
## 1: 0.2009551 1.7419116      B      C
## 2: 0.5162980 0.2080352      B      D
## 3: 0.2317091 1.4276071      A      C
## 4: 0.9692728 0.3216744      B      D
## 5: 0.5424793 1.7834104      B      D
```

On remarque que :

- ici encore à l'intérieur des crochets on écrit `y` et pas `dt$y` ;
- on a réalisé un test portant l'une des variables du data-table. On peut ainsi sélectionner des individus qui possèdent certaines propriétés communes.

Test (3)

```
dt[x<=.5 & y>0][1:5]
```

```
##            x          y group1 group2
## 1: 0.2009551 1.74191161      B      C
## 2: 0.2317091 1.42760714      A      C
## 3: 0.3519434 0.09156552      B      D
## 4: 0.1924503 1.18221697      B      C
## 5: 0.2923867 0.15773065      A      D
```

Bilan

Comme pour les data-frames, il est possible de filtrer les individus :

- soit en utilisant des numéros de lignes ;
- soit en effectuant des tests.

La syntaxe est un peu plus simple (la lisibilité est améliorée) : `x` au lieu de `dt$x`.

ATTENTION ! Il n'est pas possible de filtrer des individus à l'aide du nom de la ligne. Cette notion n'existe pas pour les data-tables !

Le package `data.table`

Sélectionner des variables

Principes

La sélection de variables se fait en utilisant la partie `j` de la syntaxe générale. Il existe au moins trois façons de s'y prendre, en utilisant :

- des numéros de colonne ;
- des chaînes de caractères de noms de variables ;
- des vecteurs de variables.

Numéros de colonnes (1)

```
dt[, 1, ]
```

```
##              x
##   1: 0.20095513
##   2: 0.51629797
##   3: 0.68051717
##   4: 0.17897578
##   5: 0.40181986
##   6: 0.23170910
##   7: 0.96927277
##   8: 0.54247928
##   9: 0.35738144
##  10: 0.35194336
##  11: 0.33665128
##  12: 0.92807520
##  13: 0.46925895
##  14: 0.53500958
##  15: 0.57515620
##  16: 0.22934963
```

Numéros de colonnes (2)

Attention ! data-frame/data-table : une différence notable.

```
as.data.frame(dt)[, 1]
```

```
##   [1] 0.20095513 0.51629797 0.68051717 0.17897578 0.40181986 0.23170910
##   [7] 0.96927277 0.54247928 0.35738144 0.35194336 0.33665128 0.92807520
##  [13] 0.46925895 0.53500958 0.57515620 0.22934963 0.13726358 0.51402387
##  [19] 0.19245033 0.96923869 0.29238672 0.13646836 0.91623377 0.21571646
##  [25] 0.99145923 0.24811327 0.83610944 0.84537896 0.04172308 0.45416264
##  [31] 0.60703509 0.15198264 0.49449173 0.36070892 0.22491746 0.63606174
##  [37] 0.43579928 0.93589065 0.65945989 0.05034630 0.65437957 0.28016047
##  [43] 0.74026887 0.62570915 0.49479853 0.94231888 0.53461595 0.86130312
##  [49] 0.58683497 0.72524894 0.06265517 0.70495757 0.79758342 0.63253444
##  [55] 0.81733301 0.69259651 0.83093545 0.46793270 0.39367215 0.64561374
##  [61] 0.47640326 0.51526935 0.64646049 0.33015867 0.18095600 0.77848532
##  [67] 0.48099079 0.12895680 0.68056504 0.64068408 0.74874239 0.13135693
##  [73] 0.73122507 0.71631643 0.07104298 0.94434828 0.43579621 0.25619306
##  [79] 0.65930635 0.85278377 0.28563657 0.94821825 0.35198213 0.67803418
##  [85] 0.21707374 0.78810341 0.20328391 0.63690719 0.97352914 0.52478698
```

Numéros de colonnes (3)

```
dt[, c(1, 2)]
```

```
##                x           y
##   1: 0.20095513  1.74191161
##   2: 0.51629797  0.20803517
##   3: 0.68051717 -0.58808030
##   4: 0.17897578 -0.21175359
##   5: 0.40181986 -2.74766842
##   6: 0.23170910  1.42760714
##   7: 0.96927277  0.32167444
##   8: 0.54247928  1.78341044
##   9: 0.35738144 -0.08694961
##  10: 0.35194336  0.09156552
##  11: 0.33665128 -1.11542859
##  12: 0.92807520  0.67803583
##  13: 0.46925895 -0.85015917
##  14: 0.53500958 -0.43322732
##  15: 0.57515620 -1.20906002
##  16: 0.22934963 -0.44531166
```

Nom des variables

```
dt[, "x"]
```

```
##              x
##   1: 0.20095513
##   2: 0.51629797
##   3: 0.68051717
##   4: 0.17897578
##   5: 0.40181986
##   6: 0.23170910
##   7: 0.96927277
##   8: 0.54247928
##   9: 0.35738144
##  10: 0.35194336
##  11: 0.33665128
##  12: 0.92807520
##  13: 0.46925895
##  14: 0.53500958
##  15: 0.57515620
##  16: 0.22934963
```

Liste des variables (1)

```
dt[, list(x, y)]
```

```
##                x           y
##   1: 0.20095513  1.74191161
##   2: 0.51629797  0.20803517
##   3: 0.68051717 -0.58808030
##   4: 0.17897578 -0.21175359
##   5: 0.40181986 -2.74766842
##   6: 0.23170910  1.42760714
##   7: 0.96927277  0.32167444
##   8: 0.54247928  1.78341044
##   9: 0.35738144 -0.08694961
##  10: 0.35194336  0.09156552
##  11: 0.33665128 -1.11542859
##  12: 0.92807520  0.67803583
##  13: 0.46925895 -0.85015917
##  14: 0.53500958 -0.43322732
##  15: 0.57515620 -1.20906002
##  16: 0.22934963 -0.44531166
```

Liste des variables (2)

Il faut toujours utiliser une liste, même avec une seule variable !

```
dt[1:2, .(x)]
```

```
##            x
## 1: 0.2009551
## 2: 0.5162980
```

Sinon le résultat obtenu est différent

```
dt[1:2, x]
```

```
## [1] 0.2009551 0.5162980
```

Cette possibilité n'est pas recommandée par les auteurs de `data.table`

Liste des variables (3)

Notons au passage qu'on peut renommer les variables à l'aide des listes :

```
dt[1:2, .(xx = x, yy = y)]
```

```
##           xx        yy
## 1: 0.2009551 1.7419116
## 2: 0.5162980 0.2080352
```

Le data-table n'est pas modifié pour autant comme on peut le constater :

```
dt[1:2]
```

```
##            x         y group1 group2
## 1: 0.2009551 1.7419116      B      C
## 2: 0.5162980 0.2080352      B      D
```

Le package `data.table`

Manipuler des variables

Copie ou référence ?

En informatique, il existe au moins deux façons de *copier* des tableaux :

- on peut en faire une **copie** intégrale et indépendante. Si `df1` est un tableau, et si `df2` en est une copie, la modification de `df2` n'aura aucun effet sur `df1` et inversement.
- on peut **faire référence** à ce tableau. Si `dt1` fait référence à `dt2`, toute modification de l'un modifie l'autre. Les deux variables **pointent** sur le même emplacement en mémoire vive.

Les utilisateurs de R sont habitués au premier comportement. En utilisant les data-tables il faut comprendre le deuxième, plus économe en mémoire vive !

Modifier une variable

```
dt[, y := x**2]
dt[1:5]
```

```
##             x          y group1 group2
## 1: 0.2009551 0.04038296      B      C
## 2: 0.5162980 0.26656360      B      D
## 3: 0.6805172 0.46310362      A      D
## 4: 0.1789758 0.03203233      B      C
## 5: 0.4018199 0.16145920      B      C
```

La variable `y` a été modifiée par référence. Aucune copie n'a été crée. C'est très efficace...

Créer des variables (1)

On peut créer une nouvelle variable très simplement

```
dt[, z := sqrt(x**2 + y**2)]
dt[1:5]
```

```
##             x          y group1 group2         z
## 1: 0.2009551 0.04038296      B      C 0.2049726
## 2: 0.5162980 0.26656360      B      D 0.5810506
## 3: 0.6805172 0.46310362      A      D 0.8231455
## 4: 0.1789758 0.03203233      B      C 0.1818197
## 5: 0.4018199 0.16145920      B      C 0.4330454
```

Créer des variables (2)

On peut en créer plusieurs simultanément :

```
dt[, `:=`(
  a = 1,
  b = z - 1)]
dt[1:3, .(a, b)]
```

```
##    a          b
## 1: 1 -0.7950274
## 2: 1 -0.4189494
## 3: 1 -0.1768545
```

```
dt[, c("a", "b") := .(1, z-1)]
dt[1:3, .(a, b)]
```

```
##    a          b
## 1: 1 -0.7950274
## 2: 1 -0.4189494
## 3: 1 -0.1768545
```

Supprimer des variables

C'est facile :

```
dt[, z := NULL]
dt[1:5]
```

```
##             x          y group1 group2 a          b
## 1: 0.2009551 0.04038296      B      C 1 -0.7950274
## 2: 0.5162980 0.26656360      B      D 1 -0.4189494
## 3: 0.6805172 0.46310362      A      D 1 -0.1768545
## 4: 0.1789758 0.03203233      B      C 1 -0.8181803
## 5: 0.4018199 0.16145920      B      C 1 -0.5669546
```

Le package `data.table`

Faire des calculs

Calculs itermédiaires (1)

Il est possible d'utiliser des blocs `{...}` afin d'effectuer des calculs intermédiaires et de retourner seulement la liste voulue :

```
dt[, {
  a <-  x**2
  b <-  y**2
  z <-  a+b
  t <-  a-b
  .(z = z, t = t)
}]
```

```
##                z           t
##   1: 0.042013748 0.038752180
##   2: 0.337619747 0.195507445
##   3: 0.677568588 0.248638657
##   4: 0.033058399 0.031006259
##   5: 0.187528277 0.135390128
##   6: 0.056571628 0.050806588
##   7: 1.822130629 0.056848794
```

Calculs itermédiaires (2)

```
dt[, {
  distance <- x**2+y**2
  oo <- order(distance)
  z <-  x[oo]
  t <-  y[oo]
  .(x = z, y = t, distance = distance[oo])
}]
```

```
##                x           y    distance
##    1: 0.04172308 0.001740815 0.001743845
##    2: 0.05034630 0.002534750 0.002541175
##    3: 0.06265517 0.003925670 0.003941081
##    4: 0.07104298 0.005047105 0.005072578
##    5: 0.12895680 0.016629857 0.016906409
##    6: 0.13135693 0.017254642 0.017552365
##    7: 0.13646836 0.018623612 0.018970451
##    8: 0.13726358 0.018841290 0.019196284
##    9: 0.15198264 0.023098722 0.023632273
##   10: 0.15398931 0.023712708 0.024275001
```

Groupement (1)

Il est possible d'effectuer certains calculs en fonctions de modalités de variables catégorielles. Par exemple si l'on souhaite calculer la moyenne en fonction des modalités de la variable `group1` il suffit d'utiliser l'argument `by` :

```
dt[, .(mx = mean(x)), by = group1]
```

```
##    group1        mx
## 1:      B 0.5243393
## 2:      A 0.5229959
```

Les modalités sont ordonnées en fonction de leur apparition lors du parcours des lignes. Pour des raisons d'efficacité numérique.

Groupement (2)

Cas de plusieurs variables catégorielles

```
dt[1:5,
   .(mx = mean(x)),
   by = .(group1, group2)]
```

```
##    group1 group2        mx
## 1:      B      C 0.2605836
## 2:      B      D 0.5162980
## 3:      A      D 0.6805172
```

```
dt[1:5,
   .(mx = mean(x)),
   by = c("group1", "group2")]
```

```
##    group1 group2        mx
## 1:      B      C 0.2605836
## 2:      B      D 0.5162980
## 3:      A      D 0.6805172
```

Groupement (3)

Très souvent, on cherche à connaître l'effectif de chaque classe résultant d'un groupement. L'opérateur `.N` est là pour ça :

```
dt[, .N, by = .(G1 = group1, G2 = group2)]
```

```
##    G1 G2  N
## 1:  B  C 23
## 2:  B  D 21
## 3:  A  D 22
## 4:  A  C 34
```

Remarquons au passage qu'on peut aussi renommer les variables catégorielles dans la même instruction !

Utilisation de .SD

```
dt[1:5,
   lapply(data.table(x=x,y=y), mean),
   by = .(group1, group2)]
```

```
##    group1 group2         x          y
## 1:      B      C 0.2605836 0.07795817
## 2:      B      D 0.5162980 0.26656360
## 3:      A      D 0.6805172 0.46310362
```

```
dt[1:5,
   lapply(.SD, mean),
   by = .(group1, group2)]
```

```
##    group1 group2         x          y a          b
## 1:      B      C 0.2605836 0.07795817 1 -0.7267208
## 2:      B      D 0.5162980 0.26656360 1 -0.4189494
## 3:      A      D 0.6805172 0.46310362 1 -0.1768545
```

Utilisation de `.SDcols`

Lorsqu'on fait un regroupement par rapport à certaines variables catégorielles (pas toutes) et qu'on veut appliquer une fonction d'agrégation à toutes les variables quantitatives, on peut préciser ce que contient `.SD` à l'aide de `.SDcols` :

```
dt[1:5,
   lapply(.SD, mean),
   by = .(group1),
   .SDcols = -c("group2")]
```

```
##    group1         x         y a          b
## 1:      B 0.3245122 0.1251095 1 -0.6497780
## 2:      A 0.6805172 0.4631036 1 -0.1768545
```

Le package `data.table`

En vrac

Décallage

```
dt[1:4,
   .(x = x, y = shift(x, type = "lead", fill = NA))]
```

```
##            x         y
## 1: 0.2009551 0.5162980
## 2: 0.5162980 0.6805172
## 3: 0.6805172 0.1789758
## 4: 0.1789758        NA
```

```
dt[1:4,
   .(x = x, y = shift(x, type = "lag", fill = NA, n = 2L))]
```

```
##            x         y
## 1: 0.2009551        NA
## 2: 0.5162980        NA
## 3: 0.6805172 0.2009551
## 4: 0.1789758 0.5162980
```

Quelques fonctions utiles (1)

```
tables() # liste des data-table
```

```
##    NAME NROW NCOL MB                    COLS KEY
## 1:   dt  100    6  0 x,y,group1,group2,a,b
## Total: 0MB
```

```
setnames(dt, old = c("x", "y"), new = c("xx", "yy"))
dt[1:4]
```

```
##           xx         yy group1 group2 a          b
## 1: 0.2009551 0.04038296      B      C 1 -0.7950274
## 2: 0.5162980 0.26656360      B      D 1 -0.4189494
## 3: 0.6805172 0.46310362      A      D 1 -0.1768545
## 4: 0.1789758 0.03203233      B      C 1 -0.8181803
```

Quelques fonctions utiles (2)

```
setcolorder(dt, c("group1", "group2")); dt[1:2]
```

```
##    group1 group2        xx         yy a          b
## 1:      B      C 0.2009551 0.04038296 1 -0.7950274
## 2:      B      D 0.5162980 0.26656360 1 -0.4189494
```

```
setorder(dt, xx); dt[1:2]
```

```
##    group1 group2         xx          yy a          b
## 1:      A      D 0.04172308 0.001740815 1 -0.9582406
## 2:      A      C 0.05034630 0.002534750 1 -0.9495899
```

```
setorder(dt, -xx); dt[1:2]
```

```
##    group1 group2        xx        yy a         b
## 1:      A      C 0.9914592 0.9829914 1 0.3961603
## 2:      A      D 0.9859064 0.9720113 1 0.3844917
```

Quelques fonctions utiles (3)

```
dt1 <- data.table(x = 1:2, y = 1:2)
dt2 <- data.table(x = 1:3, y = 1:3)
rbindlist(list(dt1, dt2))
```

```
##    x y
## 1: 1 1
## 2: 2 2
## 3: 1 1
## 4: 2 2
## 5: 3 3
```

```
dt1 <- data.table(x = 1:2, y = 1:2)
dt2 <- data.table(x = 1:3, y = 1:3, z = 1:3)
rbindlist(list(dt1, dt2))
```

Produit une erreur !

Les deux data-tables n'ont pas le même nombre de colonnes.

Indexation (1)

Le **filtrage** d'une table selon des colonnes est nettement plus rapide si celles-ci sont pré-triées dans un ordre compatible avec le filtrage.

Cette idée est exploitée de manière efficace dans les bases de données sous le nom d'*indexation par clés*. Le package `data.table` dispose d'une telle indexation où les clés sont des colonnes choisies par l'utilisateur. Le tableau est alors ordonné suivant ces clés (il peut y en avoir plusieurs), et les filtres sur ces clés deviennent beaucoup plus rapides, avec un appel simplifié, en respectant l'ordre des clés.

Indexation (2)

```
setkeyv(dt, "group1")
dt["A"][1:2] # au lieu de dt[group1 == "A"][1:2]
```

```
##    group1 group2        xx        yy a         b
## 1:      A      C 0.9914592 0.9829914 1 0.3961603
## 2:      A      D 0.9859064 0.9720113 1 0.3844917
```

```
setkey(dt, group1, group2)
dt[list("A", "C")][1:2]
```

```
##    group1 group2        xx        yy a         b
## 1:      A      C 0.9914592 0.9829914 1 0.3961603
## 2:      A      C 0.9443483 0.8917937 1 0.2988801
```

Indexation (3)

```
tables()
```

```
##    NAME NROW NCOL MB                       COLS           KEY
## 1:   dt  100    6  0 group1,group2,xx,yy,a,b group1,group2
## 2:  dt1    2    2  0                       x,y
## 3:  dt2    3    2  0                       x,y
## Total: 0MB
```

```
setkey(dt, NULL)
tables()
```

```
##    NAME NROW NCOL MB                       COLS KEY
## 1:   dt  100    6  0 group1,group2,xx,yy,a,b
## 2:  dt1    2    2  0                       x,y
## 3:  dt2    3    2  0                       x,y
## Total: 0MB
```