Tuto Package Data.table

Claire MAZZUCATO

21/12/2020

Figure 1: PSB website.

Data.table

Qu'est-ce que c'est ?

`data.table` fournit une version haute performance du `data.frame` de base R avec des améliorations de syntaxe et de fonctionnalités pour la facilité d'utilisation, la commodité et la vitesse de programmation.

Le data.table est une alternative au data.frame par défaut de R pour traiter les données tabulaires.

La raison pour laquelle il est si populaire est la rapidité d'exécution sur des données plus volumineuses et la syntaxe laconique. Ainsi, vous tapez effectivement moins de code et obtenez une vitesse beaucoup plus rapide. C'est l'un des package les plus téléchargés dans R.

À la fin de ce guide, vous comprendrez la syntaxe fondamentale de data.table et la structure qui la sous-tend. Toutes les fonctions de manipulation des données de base de data.table, dans quels scénarios elles sont utilisées et comment les utiliser, avec également quelques trucs et astuces avancés.

Installation

L'installation du package `data.table` n'est pas différente des autres packages R. Il est recommandé d'exécuter `install.packages()` pour obtenir la dernière version.

```
#install.packages('data.table')
```

Usage

L'usage de data.tables est différente de la façon dont vous travailleriez avec data.frames.

Le `fread()`, abréviation de `fast read`, est la version `data.tables` de `read.csv()`.

Comme `read.csv()`, il fonctionne pour un fichier sur votre ordinateur local ainsi que pour un fichier hébergé sur Internet.

Importons le jeu de données `mtcars` stocké dans un fichier csv.

```
library(data.table)
mt <- fread("https://raw.githubusercontent.com/selva86/datasets/master/mtcars.csv")
head(mt)
```

```
##         mpg cyl disp  hp drat    wt  qsec vs am gear carb fast
## 1: 4.582576   6  160 110 3.90 2.620 16.46  0  1    4    4    1
## 2: 4.582576   6  160 110 3.90 2.875 17.02  0  1    4    4    1
## 3: 4.774935   4  108  93 3.85 2.320 18.61  1  1    4    1    1
## 4: 4.626013   6  258 110 3.08 3.215 19.44  1  0    3    1    1
## 5: 4.324350   8  360 175 3.15 3.440 17.02  0  0    3    2    1
## 6: 4.254409   6  225 105 2.76 3.460 20.22  1  0    3    1    1
##                  cars           carname
## 1:         Mazda RX4         Mazda RX4
## 2:     Mazda RX4 Wag     Mazda RX4 Wag
## 3:        Datsun 710        Datsun 710
## 4:    Hornet 4 Drive    Hornet 4 Drive
## 5: Hornet Sportabout Hornet Sportabout
## 6:           Valiant           Valiant
```

```
class(mt)
```

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

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

Les données importées sont stockées directement sous forme de data.table.

Dans le résultat ci-dessus, la `data.table` hérite d'une classe de data.frame et est donc un data.frame en soi. Ainsi, les fonctions qui acceptent un data.frame fonctionneront très bien sur la data.table également.

Comme l'ensemble de données que nous avons importé est petit, la vitesse de lecture du fichier csv() était satisfaisante Cependant, le gain de vitesse devient évident lorsque l'on importe un grand ensemble de données (des millions de lignes). L'execution est 20x plus rapide.

Conversion

Il est possible de convertir n'importe quel "data.frame" en "data.table" en utilisant une des deux approches :

- data.table(df) ou as.data.table(df)
- setDT(df)

La différence entre les deux approches est la suivante : La fonction `data.table(df)` créera une copie de df et la convertira en `data.table`.

Alors que la fonction `setDT(df)` le convertit en un `data.table` en place. Cela signifie que le "df" lui-même est converti en un "data.table" et que vous n'avez pas besoin de l'affecter à un objet différent.

Important : Le fichier `data.table()` ne comporte pas de nom de ligne. Donc si le `data.frame` a des noms d'emprunt, vous devez le stocker dans une colonne séparée avant de le convertir en `data.table`.

```
mtcars$carname <- rownames(mtcars)
mtcars_dt <- as.data.table(mtcars)
class(mtcars_dt)
```

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

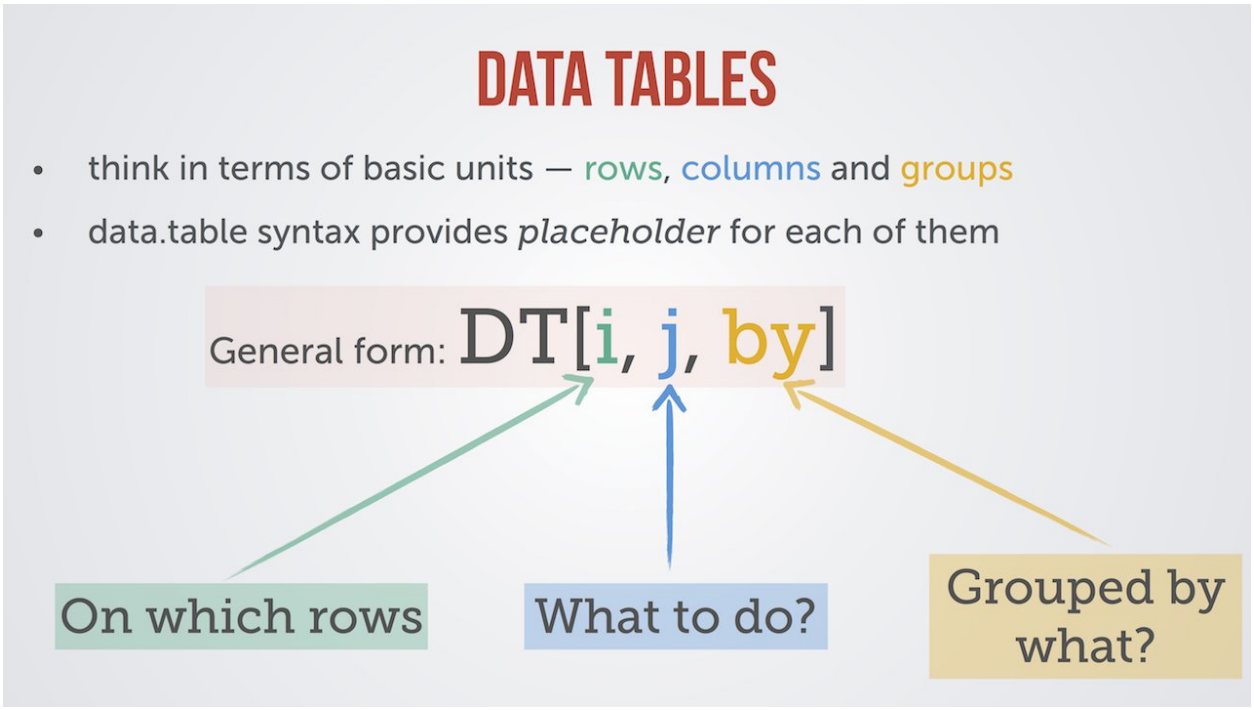
```
mtcars_copy <- copy(mtcars)
setDT(mtcars_copy)
class(mtcars_copy)
```

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

Filtrer

La principale différence avec `data.frame` est que `data.table` reconnait le nom de colonnes.

Ainsi, lors du filtrage, il suffit de faire passer les noms des colonnes entre les crochets.




```
mtcars[mtcars$cyl == 6 & mtcars$gear == 4, ]
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb       carname
```

```
## Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4     Mazda RX4
## Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4 Mazda RX4 Wag
## Merc 280      19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4      Merc 280
## Merc 280C     17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4     Merc 280C
```

```
mtcars_dt[cyl==6 & gear==4, ]
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb       carname
## 1: 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4     Mazda RX4
## 2: 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4 Mazda RX4 Wag
## 3: 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4      Merc 280
## 4: 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4     Merc 280C
```

Séléctionner des colonnes

Avec `data.table`,vous ne pouvez pas sélectionner une colonne par sa position numérotée dans un `data.table`.

Par exemple, vous pouvez vous attendre à ce que les éléments suivants fonctionnent dans un `data.frame`.

```
mtcars[, 1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Mais cela ne ferait que renvoyer “1” dans un `data.table`

```
mtcars_dt[, 1]
```

```
##      mpg
##  1: 21.0
##  2: 21.0
##  3: 22.8
##  4: 21.4
##  5: 18.7
##  6: 18.1
##  7: 14.3
##  8: 24.4
##  9: 22.8
## 10: 19.2
## 11: 17.8
## 12: 16.4
## 13: 17.3
## 14: 15.2
## 15: 10.4
## 16: 10.4
## 17: 14.7
## 18: 32.4
## 19: 30.4
## 20: 33.9
## 21: 21.5
```

```
## 22: 15.5
## 23: 15.2
## 24: 13.3
## 25: 19.2
## 26: 27.3
## 27: 26.0
## 28: 30.4
## 29: 15.8
## 30: 19.7
## 31: 15.0
## 32: 21.4
##      mpg
```

Si vous voulez obtenir cette colonne uniquement par position, vous devez ajouter un argument supplémentaire, avec=FALSE.

```
mtcars_dt[, 1, with=F]
```

```
##      mpg
##  1: 21.0
##  2: 21.0
##  3: 22.8
##  4: 21.4
##  5: 18.7
##  6: 18.1
##  7: 14.3
##  8: 24.4
##  9: 22.8
## 10: 19.2
## 11: 17.8
## 12: 16.4
## 13: 17.3
## 14: 15.2
## 15: 10.4
## 16: 10.4
## 17: 14.7
## 18: 32.4
## 19: 30.4
## 20: 33.9
## 21: 21.5
## 22: 15.5
## 23: 15.2
## 24: 13.3
## 25: 19.2
## 26: 27.3
## 27: 26.0
## 28: 30.4
## 29: 15.8
## 30: 19.7
## 31: 15.0
## 32: 21.4
##      mpg
```

Le résultat obtenu est un data.table à une colonne.

Une autre façon et une meilleure pratique consiste à passer le nom de la colonne réelle.

```
mtcars_dt[, mpg]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Séléctionner plusieurs colonnes

Que faire si le nom de la colonne est présent sous forme de chaîne dans une autre variable (vecteur) ?

Dans ce cas, vous ne pouvez pas utiliser mpg directement. Vous devez en outre passer avec=FALSE.

```
myvar <- "mpg"
mtcars_dt[, myvar, with=F]
```

```
##      mpg
##  1: 21.0
##  2: 21.0
##  3: 22.8
##  4: 21.4
##  5: 18.7
##  6: 18.1
##  7: 14.3
##  8: 24.4
##  9: 22.8
## 10: 19.2
## 11: 17.8
## 12: 16.4
## 13: 17.3
## 14: 15.2
## 15: 10.4
## 16: 10.4
## 17: 14.7
## 18: 32.4
## 19: 30.4
## 20: 33.9
## 21: 21.5
## 22: 15.5
## 23: 15.2
## 24: 13.3
## 25: 19.2
## 26: 27.3
## 27: 26.0
## 28: 30.4
## 29: 15.8
## 30: 19.7
## 31: 15.0
## 32: 21.4
##      mpg
```

Renommer des colonnes

La fonction `setnames()` est utilisée pour renommer les colonnes.

Elle prend le `data.table` (ou data.frame), le nom actuel et le nouveau nom comme arguments et change les noms des colonnes en place sans aucune copie des données.

```
setnames(mtcars_dt, 'vs', 'engine_type')
colnames(mtcars_dt)
```

```
##  [1] "mpg"         "cyl"         "disp"        "hp"          "drat"
##  [6] "wt"          "qsec"        "engine_type" "am"          "gear"
## [11] "carb"        "carname"
```

On note ici que la colonne a été renommée pour engine_type.

```
DT <- data.table(A=1:5)
DT[ , X := shift(A, 1, type="lag")]
DT[ , Y := shift(A, 1, type="lead")]
```

Regrouper

En base R, le regroupement est effectué à l'aide de la fonction `aggregate()`. C'est un peu lourd et difficile de se souvenir de la syntaxe. Toutes les fonctionnalités peuvent être facilement réalisées en utilisant l'argument "by" entre crochets. Par exemple, dans les données `mtcars`, on peut obtenir le kilométrage moyen pour chaque type de cylindre en utilisant 'cyl' entre les crochets.

```
mtcars_dt[, .(mean_mileage=mean(mpg)), by=cyl]
```

```
##    cyl mean_mileage
## 1:   6     19.74286
## 2:   4     26.66364
## 3:   8     15.10000
```

Joindre des tables

Le package `data.table` fournit une implémentation plus rapide de la fonction `merge()`. La syntaxe est à peu près la même que celle de la fonction `merge()` du R de base.

```
dt1 <- mtcars_dt[5:25,.(carname, mpg, cyl)]
dt2 <- mtcars_dt[1:10, .(carname, gear)]
dt3 <- mtcars_dt[2:12, .(carname, disp)]
```

```
# Inner Join
merge(dt1, dt2, by='carname')
```

```
##              carname  mpg cyl gear
## 1:        Duster 360 14.3   8    3
## 2: Hornet Sportabout 18.7   8    3
## 3:          Merc 230 22.8   4    4
## 4:         Merc 240D 24.4   4    4
## 5:          Merc 280 19.2   6    4
## 6:          Valiant 18.1   6    3
```

```r
# Left Join
merge(dt1, dt2, by='carname', all.x = T)
```

```
##                 carname  mpg cyl gear
##  1:         AMC Javelin 15.2   8   NA
##  2:  Cadillac Fleetwood 10.4   8   NA
##  3:          Camaro Z28 13.3   8   NA
##  4:   Chrysler Imperial 14.7   8   NA
##  5:    Dodge Challenger 15.5   8   NA
##  6:          Duster 360 14.3   8    3
##  7:            Fiat 128 32.4   4   NA
##  8:         Honda Civic 30.4   4   NA
##  9:   Hornet Sportabout 18.7   8    3
## 10: Lincoln Continental 10.4   8   NA
## 11:            Merc 230 22.8   4    4
## 12:           Merc 240D 24.4   4    4
## 13:            Merc 280 19.2   6    4
## 14:           Merc 280C 17.8   6   NA
## 15:          Merc 450SE 16.4   8   NA
## 16:          Merc 450SL 17.3   8   NA
## 17:         Merc 450SLC 15.2   8   NA
## 18:    Pontiac Firebird 19.2   8   NA
## 19:      Toyota Corolla 33.9   4   NA
## 20:       Toyota Corona 21.5   4   NA
## 21:             Valiant 18.1   6    3
##                 carname  mpg cyl gear
```

```r
# Outer Join
merge(dt1, dt2, by='carname', all = T)
```

```
##                 carname  mpg cyl gear
##  1:         AMC Javelin 15.2   8   NA
##  2:  Cadillac Fleetwood 10.4   8   NA
##  3:          Camaro Z28 13.3   8   NA
##  4:   Chrysler Imperial 14.7   8   NA
##  5:           Datsun 710   NA  NA    4
##  6:    Dodge Challenger 15.5   8   NA
##  7:          Duster 360 14.3   8    3
##  8:            Fiat 128 32.4   4   NA
##  9:         Honda Civic 30.4   4   NA
## 10:      Hornet 4 Drive   NA  NA    3
## 11:   Hornet Sportabout 18.7   8    3
## 12: Lincoln Continental 10.4   8   NA
## 13:           Mazda RX4   NA  NA    4
## 14:       Mazda RX4 Wag   NA  NA    4
## 15:            Merc 230 22.8   4    4
## 16:           Merc 240D 24.4   4    4
## 17:            Merc 280 19.2   6    4
## 18:           Merc 280C 17.8   6   NA
## 19:          Merc 450SE 16.4   8   NA
## 20:          Merc 450SL 17.3   8   NA
## 21:         Merc 450SLC 15.2   8   NA
## 22:    Pontiac Firebird 19.2   8   NA
```

```
## 23:      Toyota Corolla 33.9   4   NA
## 24:       Toyota Corona 21.5   4   NA
## 25:             Valiant 18.1   6    3
##                carname  mpg cyl gear
```

Tableaux croisés dynamiques

La fonction `dcast.data.table()` est utilisée pour faire des opérations de type tableau croisé dynamique comme on le voit sur Microsoft Excel ou Google Spreadsheets.

L'avantage est que la fonction `dcast.data.table()` fonctionne aussi bien avec `data.frame`.

Créons un tableau croisé dynamique montrant le kilométrage moyen (`mpg`) des cylindres par rapport au carburateur (`Carb`)

```
dcast.data.table(mtcars_dt, cyl ~ carb, fun.aggregate = mean, value.var = 'mpg')
```

```
##    cyl     1     2    3     4    6   8
## 1:   4 27.58 25.90  NaN   NaN  NaN NaN
## 2:   6 19.75   NaN  NaN 19.75 19.7 NaN
## 3:   8   NaN 17.15 16.3 13.15  NaN  15
```

Sources :

Source CRAN

Source Machine Learning Plus