TOPITO

LA VIE, DU CÔTÉ TOP

Top 10 des raisons de passer à data.table

(parmi des milliers …)

Par Océane Jossomme - EDF
Titouan Robert - Datastorm
Fanny Meyer - dreamRs

2k18
MAI

Catégorie : Raddict

Matt Dowle

Créateur

- Hacker & Data Warrior -

7 245

posts sur SO

436

Packages ont data.table en dépendance

Avril 2006

1ère version sur le CRAN

90

Fonctions

Arun Srinivasan

Co Auteur

49

Contributeurs sur Github

V.1.11.2

12,9 M

Téléchargement au total

359 000

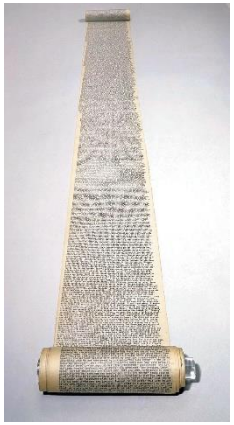
Téléchargement / Mois

NE DEPEND DE PERSONNE

Je peux pousser en prod?

Liste des dépendances

Package XXX



Package data.table

data.table

- Ne dépend que de R-base
- Importe uniquement le package « methods »
- Facilite l'évolution de version

LIRE, ECRIRE DES DONNEES

© Can Stock Photo

Lecture de données

Csv de 11,5 M de lignes et 13 colones (2.59Go)

Sur 4 cores et 8Go de RAM

data.table  tidyverse

```
data.table::fread(
        file = "./connexion_dt.csv“
)
```

51

secondes*

```
readr::read_csv(
        file = "./connexion_dt.csv“
)
```

87

secondes*

*** microbenchmark:: 20x (moyenne)**

Visualiser les données

```
options(datatable.print.class = TRUE)
data.table::as.data.table(iris)
```

```
#>      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#>             <num>       <num>        <num>       <num>    <fctr>
#>   1:          5.1         3.5          1.4         0.2    setosa
#>   2:          4.9         3.0          1.4         0.2    setosa
#>   3:          4.7         3.2          1.3         0.2    setosa
#>   4:          4.6         3.1          1.5         0.2    setosa
#>  ---
#> 146:          6.7         3.0          5.2         2.3 virginica
#> 147:          6.3         2.5          5.0         1.9 virginica
#> 148:          6.5         3.0          5.2         2.0 virginica
#> 149:          6.2         3.4          5.4         2.3 virginica
#> 150:          5.9         3.0          5.1         1.8 virginica
```

← ça affiche la classe des colonnes

← ça affiche uniquement les 5 1ère …

← … et les 5 dernières lignes

Ecriture de données

Csv de 11,5 M de lignes et 13 colones (2.59Go)

Sur 4 cores et 8Go de RAM

data.table VS tidyverse

```
data.table::fwrite(
        connexion_dt,
        file = "./connexion_dt.csv"
)
```

6

secondes*

```
readr::write_csv(
        connexion_dt,
        path = "./connexion_dt.csv"
)
```

128

secondes*

fwrite parallélisée depuis la v.1.9.7

*** microbenchmark:: 20x (moyenne)**

ELEGANCE & SIMPLICITE

UNE SYNTAXE PENSÉE EN TERMES D'UNITES BASIQUES

R	DT[	i ,	j ,	by]
SQL	FROM	WHERE	SELECT	GROUP BY
	"Prends DT,	filtre par i,	calcule j,	groupe par by"

2 DIMENSIONS CLASSIQUES : lignes & colonnes

Sélection de lignes

```
DT[id!="a"]
```

Sélection de colonnes

```
DT[,val]
```

Calcul sur les colonnes

```
DT[,mean(val)]
```

Sélection de lignes & les colonnes

```
DT[id!="a", mean(val)]
```

3ème DIMENSION *VIRTUELLE* : group by

Calculs utilisant les groupes

```
DT[id!="a",mean(val) , by = id]
```

DT	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

```
3.5
```

```
4
```

```
b 5
c 3
```

CRÉATION DE VARIABLES

R	DT[	i ,	j ,	by]
SQL	FROM	WHERE	SELECT	GROUP BY
	”Prends DT,	filtre par i,	calcule j,	groupe par by”

AJOUT DE COLONNE SIMPLE

```
DT[, list(id, new = mean(val))]
```

```
DT[, new := mean(val)]
```

COUPLÉ AU FILTRE

```
DT[id!="a", new := mean(val)]
```

COUPLÉ AU BY

```
DT[id!="a", new := mean(val) , by = id]
```

DT	id	val	new
1:	b	4	5
2:	a	2	NA
3:	a	3	NA
4:	c	1	3
5:	c	5	3
6:	b	6	5

4

FONCTIONEMENT PAR REFENCE

Sans référence

$

- Redéclare un nouvel emplacement mémoire

DT$value <- 0

Group	Value
A	1
A	2
B	3
B	4
A	5
B	6

Group	Value	Value
A	1	0
A	2	0
B	3	0
B	4	0
A	5	0
B	6	0

Il faut refaire tout depuis le départ

Avec référence

:=

- Utilise un pointeur.

Ne redéfinis aucun emplacement mémoire

DT[, value := 0]

Group	Value
A	1
A	2
B	3
B	4
A	5
B	6

Group	Value
A	0
A	0
B	0
B	0
A	0
B	0

Il suffit d'updater l'information

5

FONCTIONS SET

DES FONCTIONS AGISSANT PAR REFERENCE

RENOMMER UNE/PLUSIEURS COLONNE(S)

```
setnames(DT, "val", "val2") # par nom
setnames(DT, 2, "val2") # par position
setnames(DT, 1:2, c("A","val3")) # multiple
setnames(DT, c("id","val2"), c("A","val3"))
setnames(DT, c("A","val3")) # remplace tout si le nb de noms = ncol(DT)
```

	A	val3
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

PAS DE COPIE !!

> La modification est opérée directement sur l'objet

DES FONCTIONS AGISSANT PAR REFERENCE

TRIER LES LIGNES PAR UNE/PLUSIEURS COLONNE(S)

```
setorder(DT, id, -val, na.last=FALSE)
setorderv(DT, c("val","id"), c(1, -1) , na.last=FALSE)
```

CHANGER L'ORDRE DES COLONNES

```
setcolorder(DT, c("val","id"))
```

PASSER D'UN DF À UN DT – OU L'INVERSE

```
setDT(DF, keep.rownames=FALSE, key=NULL, check.names=FALSE)
setDF(DT, rownames=LETTERS[1:5])
```

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

val	id
3	a
2	a
6	b
4	b
5	c
1	c

LA DOUBLE IDENTITE DU BY

Sans référence

- Fait un calcul par groupe
- Ressort un nouveau data.table

```
DT[, list(value = sum(value)), by = Group]
```

Group	value
A	1
A	2
B	3
B	4
A	5
B	6

Group	value
A	8
B	13

Avec référence

- Fait un calcul par groupe
- Calcul une colonne ayant la même valeur pour tous les individus du groupe

```
DT[, value := sum(value), by = Group]
```

Group	value
A	1
A	2
B	3
B	4
A	5
B	6

Group	value
A	8
A	8
B	13
B	13
A	8
B	13

Autre spécificité du by

```
DT[, list(Sum = sum(value)),
by = list(Group1, Group2)]
```

- Plus facile à écrire

```
DT[, list(Sum = sum(value)),
by = c('Group1', 'Group2')]
```

- Plus facile à intégrer dans une fonction

- Déclarer des colonnes à la volé (pratique pour les dates!)

```
DT[, list(Sum = sum(value)),
by = list(hour(date))]
```

7

LES FONCTIONS « OUTILS »

DES OUTILS POUR TOUT !

DES FILTRES A LA POINTE

```
# un exemple avec %in%
DT[a %in% 1:2]
# un exemple avec %chin%
DT[c %chin% c("v", "w")]

# un exemple avec %between%
DT[a %between% c(4, 5)]

# un exemple avec %like%
DT[c %like% "^x"]
```

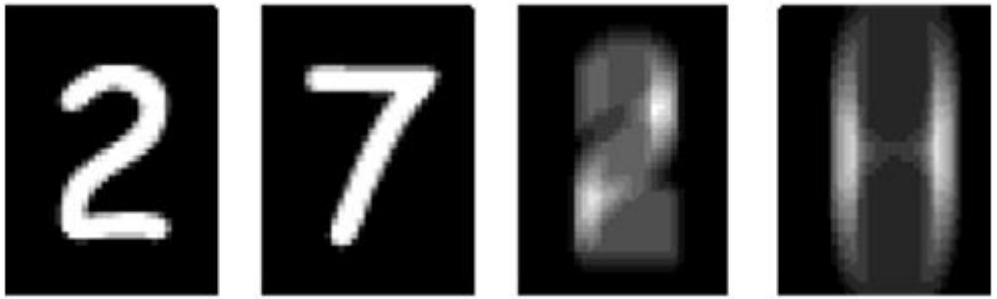
a	b	c
1	3	v
2	10	w
3	8	x
4	2	y
5	1	z

DES COMPTAGES A DIFFERENTES MAILLES

```
# un exemple avec .N
Contrat[,.N, by = type] # nombre de contrats Elec/Gaz

# un exemple avec uniqueN
Contrat[, uniqueN(PDL), by= type] # nombre de locaux Elec/Gaz
```

8

LES COMPTEURS



Compteur par client et par type de campagne

campagne

	id	type	date	Cpt_camp
1:	1	WP	2017-12-12	1
2:	1	VŒUX	2018-01-02	1
3:	1	WP	2018-01-12	2
4:	1	WP	2018-02-12	3
5:	1	FROID	2018-02-13	1
6:	1	NEW	2018-01-16	1
7:	1	WP	2018-03-12	4
8:	2	WP	2018-01-06	1
9:	2	OFFRE	2018-02-07	1
10:	2	WP	2018-02-08	2

```
setorder(campagne, id, date)
campagne[,
        cpt_camp := seq_len(.N),
        by = list(id, type)
]
```

OU

```
setorder(campagne, id, date)
campagne[,
        cpt_camp := rowid(id, type)
]
```

- Syntaxe plus courte
- ~ 3x plus rapide (sur 11,5M de ligne)

Compteur par client et par type de campagne

campagne

	id	type	date	Cpt_camp	Cpt_time
1:	1	WP	2017-12-12	1	1
2:	1	VŒUX	2018-01-02	1	1
3:	1	WP	2018-01-12	2	2
4:	1	WP	2018-02-12	3	1
5:	1	FROID	2018-02-13	1	1
6:	1	NEW	2018-01-16	1	1
7:	1	WP	2018-03-12	4	1
8:	2	WP	2018-01-06	1	1
9:	2	OFFRE	2018-02-07	1	2
10:	2	WP	2018-02-08	2	1

```
setorder(campagne, id, date)
campagne[,
        cpt_time := rowid(type),
        by = list(id, rleid(type))
]
```

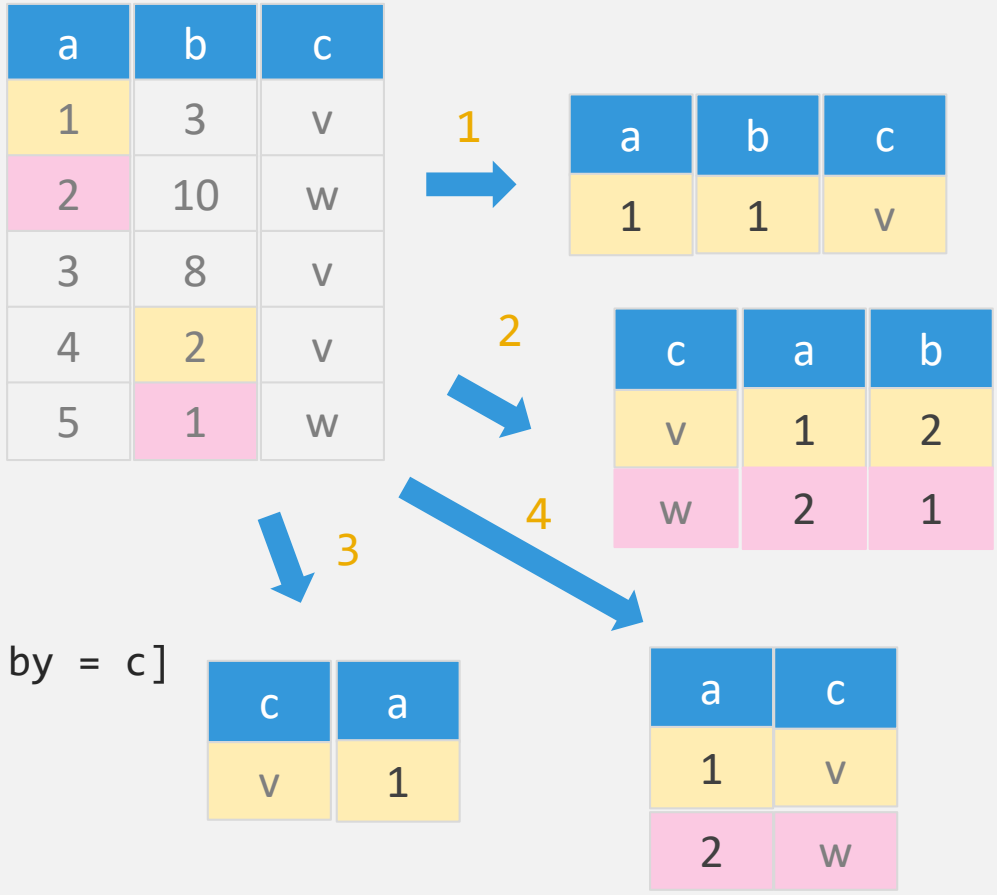
.SD + lapply

Merci JP ☺

LA RÉVOLUTION DU .SD

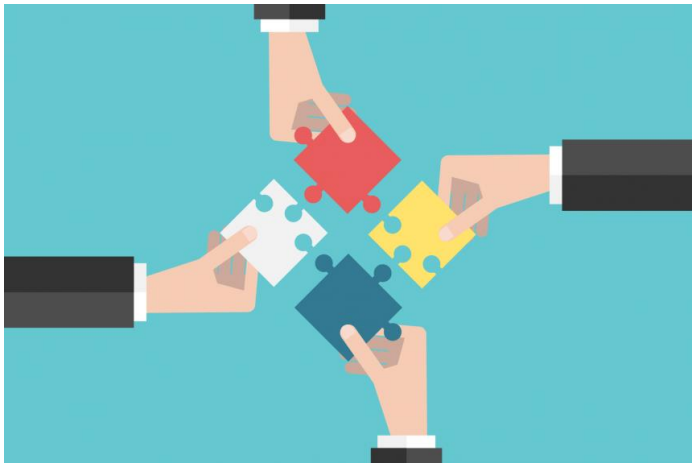
POUR APPLIQUER UN MÊME CALCUL À PLUSIEURS COLONNE

```
# 1. minimum de a et b par c
DT[, lapply(.SD, min)]

# 2. minimum de a et b par c
DT[, lapply(.SD, min), by = c]

# 3. minimum de a par c
DT[, lapply(.SD, min), .SDcol = "a"]

# 4. minimum de a par c
DT[, lapply(.SD, min), .SDcol = "a", by = c]
```



10

DCAST/MELT/MERGE/UPDATE



Use case dcast

connexions

11,5M de connexions

	id	canal	date
1:	1	web	2018-04-12
2:	2	web	2018-04-12
3:	1	app	2018-04-15
4:	1	web	2018-04-15
5:	2	web_m	2018-04-17

users_connexions

1,1M de users

	id	web	app	web_m
1:	1	2	1	0
2:	2	1	0	1
3:	3	...	...	...

data.table

```
data.table::dcast(
            data = connexions_dt,
            id ~  canal,
           fun.aggregate = length
)
```

4

secondes*

VS

reshape2

```
reshape2::dcast(
            data = connexions_df,
            id ~  canal,
           fun.aggregate = length
)
```

30,5

secondes*

* microbenchmark:: 20x (moyenne)

Use case dcast

connexions

11,5M de connexions

	id	canal	date
1:	1	web	2018-04-12
2:	2	web	2018-04-12
3:	1	app	2018-04-15
4:	1	web	2018-04-15
5:	2	web_m	2018-04-17

Agrégation par id et canal

users_connexions

1,1M de users

	id	web	app	web_m
1:	1	2	1	0
2:	2	1	0	1
3:	3	...	...	...

data.table

```
con_ag <- connexions_dt[
                        list(value = .N),
                        by = list(id, canal)
]
data.table::dcast(
            data = con_ag,
            id ~  canal,
            var.value = 'value'
)
```

3.8

secondes*

VS

tidyverse

```
connexion_df %>%
  group_by(user_id, canal) %>%
  summarise(value = n()) %>%
  spread(canal, value)
```

75

secondes*

* microbenchmark:: 20x (moyenne)

Use case merge/update par référence

connexions_annee

	id	nb_co
1:	1	400
2:	2	2
3:	3	356
4:	4	78
5:	5	122

connexions_jour

	id	nb_co_jour
1:	1	1
2:	2	10
3:	5	3

connexions_annee

	id	nb_co
1:	1	401
2:	2	12
3:	3	356
4:	4	78
5:	5	125

```
connexions_annee[
        connexions_jour,
        on = list(id),
        nb_co := nb_co + nb_co_jour
]
```

Ce « merge/update » est fait par référence donc il est rapide et concis

Use case merge/update par référence

connexions_annee

	id	nb_co
1:	1	400
2:	2	2
3:	3	356
4:	4	78
5:	5	122

connexions_jour

	id	nb_jour
1:	1	1
2:	2	10
3:	5	3

nb_co

connexions_annee

	id	nb_co
1:	1	401
2:	2	12
3:	3	356
4:	4	78
5:	5	125

```
connexions_annee[
        connexions_jour,
        on = list(id),
        nb_co := nb_co + nb_jour      i.nb_co
]
```

On peut aussi imaginer faire des conditions sur la jointure directement dans le « on », sur une date par exemple.

bonus

Mettre des data.table

dans des data.table

Group	Value	Value2
A	1	7
A	2	8
B	3	9
B	4	10
A	5	11
B	6	12
B	6	12
A	2	8

```
DT[, list(list(.SD)), by = Group]
```

Group	V1
A	Value / Value2 1 / 7 2 / 8 2 / 8 5 / 11
B	Value / Value2 3 / 9 4 / 10 6 / 12 6 / 12

- **Facilite la parallélisation :**
- V1 est une liste, il est donc possible de Lancer des traitements en parralèle dessus
- Exemple lancer une kmeans par groupe en parallèle

```
cl <- makeCluster(2)
```

```
DT[, clust := list(parLapplyLB(cl, V1 , function(X){kmeans(X,2)$cluster}))]
```

Group	V1 : Liste de data.table	clust : Liste de vecteur
A	Value / Value2 1 / 7 2 / 8 2 / 8 5 / 11	Résultat A 1 2 2 1
B	Value / Value2 3 / 9 4 / 10 6 / 12 6 / 12	Résultat B 1 1 2 2