Ordonner un data frame

par Charles Bordet et Marie-Hélène Simard

2014-03-16

Table des matières

Il est souvent nécessaire de classer un tableau de données selon les valeurs d'une ou de plusieurs de ses colonnes. Cela permet en général d'avoir une meilleure vision du jeu de données afin de mieux les comprendre, les organiser et rechercher l'information précise que l'on cherche. Cette manipulation est utile dans de nombreux cas, par exemple :

- **Classer une liste de noms** (auprès desquels on a recueilli des informations) par ordre alphabétique, afin de pouvoir retrouver plus tard un nom précis (et ses informations) plus facilement.
- **Classer des données par ordre chronologique**. C'est indispensable si on veut pouvoir afficher et visualiser ensuite l'évolution temporelle de nos variables.

D'abord une petite illustration.

Si on ne dispose que d'un simple vecteur, alors la fonction `sort` fait très bien l'affaire, comme on peut le voir sur l'exemple suivant :

```
x <- c(2, 3, 12, -3, 4, 1, 0)
sort(x)
```

```
## [1] -3  0  1  2  3  4 12
```

Néanmoins, dès qu'on veut ordonner plus qu'une variable (c'est-à-dire dès qu'on dispose d'un data frame), `sort` ne suffit plus. Nous allons alors utiliser `order` qui permet d'ordonner un data frame. Cette fonction a une démarche différente de la fonction `sort` dans le sens où elle ne permet pas directement d'ordonner le jeu de données. Dans un premier temps, elle va sortir la permutation qui va nous permettre de savoir *comment* sont ordonnées les données, et à l'aide de ces permutations on va pouvoir réarranger notre data frame. Observons la fonction en action sur notre premier exemple :

```
x <- c(2, 3, 12, -3, 4, 1, 0)
(permutation <- order(x))
```

```
## [1] 4 7 6 1 2 5 3
```

```
# La plus petite valeur est en 4e position, la deuxième plus petite en 7e position,
# et ainsi de suite...
x[permutation]
```

```
## [1] -3  0  1  2  3  4 12
```

```
# Une manipulation plus rapide et facile à se rappeler :
x[order(x)]
```

```
## [1] -3  0  1  2  3  4 12
```

Et notre vecteur est ordonné !

La puissance de `order` !

Pour illustrer l'utilisation de cette fonction sur un data frame, nous allons utiliser le jeu de données `iris` fournit directement avec R. On y trouve les mesures en centimètres de la longueur et de la largeur des pétales et des sépales de cinquante fleurs pour trois espèces différences d'iris.

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 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
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Imaginons que nous voulions classer le tableau de données en fonction de la longueur des sépales, alors suivant le même procédé que précédemment, on écrira :

```
permutation <- order(iris$Sepal.Length)
head(iris[permutation,])
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 14          4.3         3.0          1.1         0.1  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 4           4.6         3.1          1.5         0.2  setosa
```

Et si on voulait dans l'ordre décroissant ? Il suffit simplement de le préciser en option dans la fonction en ajoutant `decreasing = TRUE` :

```
permutation <- order(iris$Sepal.Length, decreasing = TRUE)
head(iris[permutation,])
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 132          7.9         3.8          6.4         2.0 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 119          7.7         2.6          6.9         2.3 virginica
## 123          7.7         2.8          6.7         2.0 virginica
## 136          7.7         3.0          6.1         2.3 virginica
## 106          7.6         3.0          6.6         2.1 virginica
```

Objectif réussi ! Notre data frame est classé par ordre croissant de la longueur des pétales. On remarque une propriété de la fonction `order` :

> En cas d'égalité, la fonction `order` laisse les observations dans leur ordre d'origine par défaut.

Et si on voulait résoudre ces cas d'égalité en spécifiant une autre colonne pour les départager ? `order` le permet en ajoutant autant de colonnes qu'on veut à l'algorithme de tri. S'il y a un cas d'égalité dans la première qu'on spécifie, alors il essaie de départager dans la deuxième, puis dans la troisième, et ainsi de

suite... Si jamais un cas d'égalité n'est pas résolu, alors on garde l'ordre d'origine. Essayons par exemple de départager nos cas d'égalité par la largeur des sépales :

```
permutation <- order(iris$Sepal.Length, iris$Sepal.Width, decreasing = TRUE)
head(iris[permutation,]) # Il n'y a plus aucun cas d'égalité.
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 132          7.9         3.8          6.4         2.0 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 136          7.7         3.0          6.1         2.3 virginica
## 123          7.7         2.8          6.7         2.0 virginica
## 119          7.7         2.6          6.9         2.3 virginica
## 106          7.6         3.0          6.6         2.1 virginica
```

Sauf qu'en utilisant l'option `decreasing = TRUE`, on classe chaque colonne par ordre décroissant. Si on veut départager nos cas d'égalité par ordre croissant de la largeur des sépales, il suffit de placer un signe `-` devant la colonne que l'on veut ranger par ordre décroissant (puisqu'ainsi sa permutation va être inversée) :

```
permutation <- order(-iris$Sepal.Length, iris$Sepal.Width)
head(iris[permutation,]) # Il n'y a plus aucun cas d'égalité.
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 132          7.9         3.8          6.4         2.0 virginica
## 119          7.7         2.6          6.9         2.3 virginica
## 123          7.7         2.8          6.7         2.0 virginica
## 136          7.7         3.0          6.1         2.3 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 106          7.6         3.0          6.6         2.1 virginica
```

Utilisation de `arrange {plyr}`

`plyr` est un package offrant tout un panel de fonctions permettant globalement de partitionner un jeu de données, puis de travailler séparément sur chaque partition avant d'à nouveau regrouper tout ensemble le jeu de données. La documentation est disponible à l'adresse suivante : http://cran.r-project.org/web/packages/plyr/plyr.pdf

La fonction `arrange` du package `plyr` permet de faire exactement la même chose que `order` mais de manière beaucoup plus simple pour l'utilisateur. En effet, on n'a plus besoin de générer un vecteur de permutation qu'on doit ensuite utiliser pour ordonner notre data frame, `arrange` fait tout ça en arrière-plan et nous sort directement le résultat voulu, c'est-à-dire le data frame ordonné. Reprenons notre exemple de base :

```
library(plyr)
```

```
x <- c(2, 3, 12, -3, 4, 1, 0)
arrange(x)
```

```
## Error: is.data.frame(df) is not TRUE
```

```
# Il faut absolument mettre un data frame en entrée.
x <- data.frame(colonne = c(2, 3, 12, -3, 4, 1, 0))
arrange(x, colonne)
```

```
##   colonne
## 1      -3
## 2       0
```

```
## 3       1
## 4       2
## 5       3
## 6       4
## 7      12
```

Que remarque-t-on ? D'abord que la fonction `arrange` ne peut pas prendre de vecteur en entrée, contrairement à `order`, mais c'est un bien léger défaut dans la mesure où pour un vecteur, on va généralement utiliser `sort`. Si on définit `x` comme un data frame, alors il suffit de préciser à la fonction le data frame en premier argument, puis les colonnes ensuite, et on se retrouve avec un data frame classé !

Reprenons l'exemple avec le jeu de données `iris` où on voulait d'abord classer les fleurs par la longueur des sépales par ordre décroissant :

```
head(arrange(iris, desc(Sepal.Length)))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          7.9         3.8          6.4         2.0 virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.7         2.6          6.9         2.3 virginica
## 4          7.7         2.8          6.7         2.0 virginica
## 5          7.7         3.0          6.1         2.3 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

Plusieurs choses à remarquer :

- D'abord les données sont renumérotées dans l'ordre de classement. Ce n'était pas le cas avec `order` où les observations gardaient leur numéro d'origine.
- Néanmoins, on observe le même classement qu'avec `order`, cela signifie qu'à nouveau les cas d'égalité sont traités en gardant l'ordre d'origine.
- Pour classer par ordre décroissant, il n'existe pas d'option comme pour `order`, mais on peut utiliser `desc` (provenant aussi du package `plyr` sur chaque colonne qu'on veut ordonner par ordre décroissant.

Comme pour `order`, on peut ajouter des colonnes pour traiter les cas d'égalité à un second niveau, ainsi :

```
head(arrange(iris, desc(Sepal.Length), Sepal.Width))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          7.9         3.8          6.4         2.0 virginica
## 2          7.7         2.6          6.9         2.3 virginica
## 3          7.7         2.8          6.7         2.0 virginica
## 4          7.7         3.0          6.1         2.3 virginica
## 5          7.7         3.8          6.7         2.2 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

On obtient le résultat voulu !