Réorganisation des données par concaténation et séparation d'observations

Mohammed Amine Ennajeh et Anas Koubaa

2016-05-08

Table des matières

Introduction

La concaténation et la séparation d'observations de variables sont deux manipulations courantes dans la réorganisation des données, servant à refaçonner les jeux de données afin de mieux les exploiter.
La concaténation consiste à rattacher les observations de plusieurs variables, généralement des chaînes de caractères, et les regrouper en une seule variable.
La séparation consiste par contre à "détacher" l'information contenue dans une variable pour la répartir sur plusieurs autres variables.
Ces deux manipulations peuvent être réalisées sous R suivant différentes approches. On présentera dans cette fiche deux manières de concaténer/séparer des observations dans un jeu de données :

- Fonctions de base R
- Package {tidyr}

Description du jeu de données utilisé

Le jeu de données utilisé est un tableau de détails concernant 114 matchs de tennis en simple masculin du tournoi de Wimbledon 2013, dans lequel on trouve, entre autres, les variables suivantes :

Variable	Description
Player1	Nom du premier joueur
Player2	Nom du second joueur
Result	Résultat du match (1 pour la victoire du premier joueur, 0 pour sa défaite)
STx.1	Nombre de jeux remportés au set x (1,2,3,4,5) par le premier joueur
STx.2	Nombre de jeux remportés au set x (1,2,3,4,5) par le second joueur

On commence par importer le jeu de données dans la variable `tennis` :

```
tennis = read.csv2(file = "Wimbledon-men-2013.csv", header = TRUE, sep = ",")
```

Puis on crée une nouvelle variable `Score` à partir de notre jeu de données `tennis`, contenant la série des scores de sets de chaque match, en commençant par ceux du joueur gagnant :

```
for (i in 1:5){
  ss <- ifelse(is.na(tennis[[paste0("ST", i, ".1")]]), NA,
               ifelse(tennis$Result,
                      paste0(tennis[[paste0("ST", i, ".1")]], "-", tennis[[paste0("ST", i, ".2")]]),
                      paste0(tennis[[paste0("ST", i, ".2")]], "-", tennis[[paste0("ST", i, ".1")]])))
  Score <- if (i == 1) ss else ifelse(is.na(ss), Score, paste0(Score, ", ", ss))
}
```

Voici quelques observations de nos variables d'interêt :

```
##     Player1         Player2 Result ST1.1 ST2.1 ST3.1 ST4.1 ST5.1 ST1.2
## 1  B.Becker        A.Murray      0     4     3     2    NA    NA     6
## 2    J.Ward          Y-H.Lu      0     7     4     6     6    NA     6
## 3   N.Mahut         J.Hajek      1     6     6     6    NA    NA     2
## 4 T.Robredo A.Bogomolov Jr.      1     6     6     6    NA    NA     2
## 5   R.Haase       M.Youzhny      0     4     5     5    NA    NA     6
## 6 M.Gicquel      V.Pospisil      0     3     2     6    NA    NA     6
##   ST2.2 ST3.2 ST4.2 ST5.2              Score
## 1     6     6    NA    NA      6-4, 6-3, 6-2
## 2     6     7     7    NA 6-7, 6-4, 7-6, 7-6
## 3     4     3    NA    NA      6-2, 6-4, 6-3
## 4     2     4    NA    NA      6-2, 6-2, 6-4
## 5     7     7    NA    NA      6-4, 7-5, 7-5
## 6     6     7    NA    NA      6-3, 6-2, 7-6
```

Concaténation/séparation avec les fonctions de base R

Concaténation

Pour concaténer les éléments de deux vecteurs en un seul, on peut utiliser les fonctions `paste` ou `paste0`, qui sont les deux fonctions de base R les plus courantes pour cette action.
La fonction `paste` permet de concaténer deux vecteurs (élément par élément), en intercalant une chaîne de caractères passée en argument `sep` entre chaque deux éléments de même position.
La fonction `paste0` est une variante de `paste`, permettant de concaténer directement deux vecteurs, sans aucun séparateur. Voici des spécifications des deux fonctions inspirées de la documentation R :

```
paste(..., sep, collapse)
paste0(..., collapse)

# Arguments :
# ... : Les objets à concaténer
# sep : La chaîne de caractères qui séparera les termes concaténés
# collapse : La chaîne de caractères séparant les résultats de concaténation terme à terme, le tout reg
```

On voudrait par exemple créer à partir de notre jeu de données `tennis` un nouveau vecteur `opponents`, qui contiendra les noms des deux joueurs de chaque match en une seule chaîne de caractères, séparés par un "

vs ". La fonction paste pourrait être utilisée à cet effet :

```
##     Player1         Player2
## 1  B.Becker        A.Murray
## 2    J.Ward          Y-H.Lu
## 3   N.Mahut         J.Hajek
## 4 T.Robredo A.Bogomolov Jr.
## 5   R.Haase       M.Youzhny
## 6 M.Gicquel      V.Pospisil
```

```
opponents = paste(tennis$Player1, tennis$Player2, sep = " vs ")
```

Le vecteur opponents contiendra donc les noms des deux joueurs du match. En voici un extrait :

```
## [1] "B.Becker vs A.Murray"         "J.Ward vs Y-H.Lu"
## [3] "N.Mahut vs J.Hajek"           "T.Robredo vs A.Bogomolov Jr."
## [5] "R.Haase vs M.Youzhny"         "M.Gicquel vs V.Pospisil"
```

Séparation

La fonction la plus courante pour la séparation d'un élément en plusieurs sous-éléments est la fonction strsplit, qui ne tient compte que des vecteurs de type caractère.
Cette fonction permet de séparer les éléments des chaînes de caractères d'un vecteur suivant un ou plusieurs caractères entrés en argument split, et retourne une liste de toutes les sous-chaînes obtenues. En voici quelques spécifications inspirées aussi de la documentation R :

```
strsplit(x, split, fixed, perl, useBytes)

# Arguments :
# x : Chaîne de caractères à séparer
# split : Caractères à utiliser pour séparer la chaîne x
# fixed : Si FALSE, split est interprété comme une expression régulière
# perl : Si TRUE, split est interprété comme une expression régulière compatible avec perl
# useBytes : Si TRUE, l'association est effectuée bit par bit plutôt que caractère par caractère
```

On veut maintenant séparer chaque série de scores selon cinq vecteurs, lesquels contiendront les scores des sets et, pour les matchs n'ayant pas abouti à cinq sets, des valeurs manquantes pour les sets non disputés. On utilise donc la fonction strsplit, en prenant en compte la séparation par virgule entre chaque paire de scores consécutifs.

```
Score_split = strsplit(x = Score, split = ", ")
```

On obtient la liste de sous-chaînes des scores de chaque set par match. En voici un extrait :

```
## [[1]]
## [1] "6-4" "6-3" "6-2"
##
## [[2]]
## [1] "6-7" "6-4" "7-6" "7-6"
##
## [[3]]
## [1] "6-2" "6-4" "6-3"
##
## [[4]]
```

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

Le problème qui se pose maintenant est que l'on ne peut pas directement affecter nos résultats à des vecteurs de scores par set, car il faudra d'abord compléter les sets non disputés de chaque élément de la liste des matchs par des valeurs manquantes, puis extraire les scores de chaque match (114 vecteurs de longueur 5) aux cinq vecteurs de sets.
Une manière de contourner ce blocage est de recourir à la fonction `mapply`, qui permet d'appliquer à la fois une fonction aux éléments d'une ou plusieurs listes.
On utilisera donc `mapply` pour completer les sets non disputés par des valeurs manquantes dans un premier temps, et pour concaténer le résultat à la liste `Score_split` :

```
Score_split_completed = mapply(FUN = c,
                               Score_split,
                               mapply(NA, FUN = rep, 5 - mapply(Score_split, FUN = length)),
                               SIMPLIFY = FALSE)
```

On termine notre manipulation par un regroupement des scores dans un data frame représentant les résultats de chaque set :

```
Score_set = data.frame(matrix(unlist(Score_split_completed), nrow = length(Score_split), byrow = TRUE))
names(Score_set) = paste0("ST", 1:5)
```

Voici un aperçu sur la variable `Score_set` résultante :

```
##   ST1 ST2 ST3  ST4  ST5
## 1 6-4 6-3 6-2 <NA> <NA>
## 2 6-7 6-4 7-6  7-6 <NA>
## 3 6-2 6-4 6-3 <NA> <NA>
## 4 6-2 6-2 6-4 <NA> <NA>
## 5 6-4 7-5 7-5 <NA> <NA>
## 6 6-3 6-2 7-6 <NA> <NA>
```

Une manière plus simple aurait été d'utiliser la fonction `str_split_fixed` du package {stringr}. Ce package contient un nombre de fonctions permettant la manipulation simple des chaînes de caractères.
On aurait donc pu séparer la variable `Score` comme suit :

```
library(stringr)
Score_split_stringr = str_split_fixed(string = Score, pattern = ", ", n = 5)
Score_split_stringr[Score_split_stringr == ""] = NA
colnames(Score_split_stringr) = paste0("ST", 1:5)
```

```
##      ST1   ST2   ST3   ST4   ST5
## [1,] "6-4" "6-3" "6-2" NA    NA
## [2,] "6-7" "6-4" "7-6" "7-6" NA
## [3,] "6-2" "6-4" "6-3" NA    NA
## [4,] "6-2" "6-2" "6-4" NA    NA
## [5,] "6-4" "7-5" "7-5" NA    NA
## [6,] "6-3" "6-2" "7-6" NA    NA
```

Le package {tidyr}

Le package a été créé par Hadley Wickham, la première fois en juillet 2014, et a subi plusieurs modifications depuis. La dernière version 0.4.1 de ce package est disponible sur le site https://cran.r-project.org/web/packages/tidyr/index.html

Concaténation/séparation avec {tidyr}

Les manipulations faites auparavant peuvent être effectuées à l'aide des fonctions offertes par le package `tidyr` de R. Ce package contient plusieurs fonctions utiles et pratiques pour arranger un jeu de données, afin de faciliter aussi bien la réorganisation et le remaniement des données en changeant la disposition des ces dernières, que la fusion et l'extraction d'observations.

Les deux propriétés importantes des jeux de données ordonnés sont :

- Chaque colonne est une variable.
- Chaque ligne est une observation.

Pour l'installation du dit package il faut soumettre l'instruction suivante :

```
install.packages("tidyr")
```

Pour appeler le package :

```
library(tidyr)
```

La fonction `unite`

La fonction qui nous interesse est `unite`, cette dernière permet de concaténer plusieurs colonnes d'un jeu de données en une seule colonne, ce qui s'avére très pratique en termes de lecture et d'affichage de données.

```
unite(data, col, ..., sep, remove)
```

On trouve dans la documentation R les détails de ces arguments :

```
#  data   : Le data frame qui contient les colonnes qu'on veut concaténer.
#   col   : Le nom de la nouvelle colonne à créer.
#   ...   : Les noms des colonnes à concaténer.
#   sep   : Le séparateur à utliser entre le contenu des variables à concaténer.
#  remove : Si TRUE, les colonnes concaténées n'apparaissent pas dans notre data frame.
```

Voici un exemple de l'utilisation de la fonction `unite` : on va concaténer les deux premières colonnes `Player1` et `Player2` du jeu de données `tennis` en utilisant le séparateur `" vs "`.

```
opponents = unite(tennis, opponents, 1:2, sep=" vs ", remove=FALSE)
```

On peut affecter d'autres fonctions à l'argument ..., à savoir les fonctions `num_range` ou `matches` du package dplyr (à charger préalablement) :

```
library(dplyr)

opponents = unite(tennis, opponents, num_range("Player", 1:2), sep = " vs ")
```

```
# num_range : permet de sélectionner une plage de variables à la fois.

opponents = unite(tennis, opponents, matches("Player"), sep = " vs ")
# matches("Player") : selectionne toutes les variables dont le nom contient l'expression "Player".
```

Pour davantage d'informations sur l'utilisation de ces options, une documentation plus détaillée sur la fonction `select` du package {dplyr} figure dans la documentation R.

```
help(select)
```

La fonction separate

La fonction `separate` permet de diviser une colonne en plusieurs colonnes. Cette fonction, à l'inverse des manœuvres avec fonctions de base R, permet d'effectuer cette tâche en une seule étape sans avoir recours à un traitement préalable. Ainsi on s'apercoit de l'utilité de cette fonction qui nous permet de manipuler les jeux de données avec subtilité.

```
separate(data, col, into, sep, remove, convert, extra, fill)
```

Les détails des arguments, inspiré de la documentation R, sont les suivants :

```
# data    : data frame.
# col     : la colonne à séparer.
# into    : les noms de nouvelles colonnes éclatées crées.
# sep     : la chaîne de caractères de séparation ou le nombre représentant la position dans laquelle l
# remove  : Si TRUE alors on supprime la colonne à séparer du jeu de données.
# convert : Si TRUE alors elle transformera automatiquement les valeurs en un type logique, numerique,
# extra   : Si le contenu de la variable à séparer est plus grand que le nombre des nouvelles colonnes
#     Elle prend trois valeurs possibles :
#     -"warn"( valeur par défault) : émet un message d'alerte et omet les valeurs en extra
#     -"drop" : omet les valeurs en extra sans émettre de message d'alerte.
#     -"merge" : sépare au plus length(into) fois.
# fill    : Si le contenu de la variable à séparer ne suffit pas pour remplir les champs des colonnes c
#     Elle prend trois valeurs possibles :
#     -"warn"(valeur par défault) : émet un message d'alerte et remplit les colonnes partant de droite.
#     -"right" : complète l'insuffisance avec des valeurs manquantes en partant de droite.
#     -"left" : complète l'insuffisance avec des valeurs manquantes en partant de gauche.
```

Ainsi, dans notre cas, on va séparer la variable `Score` en cinq variables contenant chacune le résultat d'un set, en complétant les valeurs manquantes par des `NA` grâce à l'argument `fill`. Voici un aperçu des variables d'intérêt du nouveau jeu de données :

```
Tennis = separate(cbind(tennis,Score), col = Score, into = sprintf("ST%d",1:5),
                  sep=', ', fill = "right", remove = FALSE)
head(Tennis)[c("Score", paste0("ST",1:5))]
```

```
##                Score ST1 ST2 ST3  ST4  ST5
## 1      6-4, 6-3, 6-2 6-4 6-3 6-2 <NA> <NA>
## 2 6-7, 6-4, 7-6, 7-6 6-7 6-4 7-6  7-6 <NA>
## 3      6-2, 6-4, 6-3 6-2 6-4 6-3 <NA> <NA>
## 4      6-2, 6-2, 6-4 6-2 6-2 6-4 <NA> <NA>
## 5      6-4, 7-5, 7-5 6-4 7-5 7-5 <NA> <NA>
## 6      6-3, 6-2, 7-6 6-3 6-2 7-6 <NA> <NA>
```

Comparaison des deux approches

Les fonctions de base R, dont on a présenté quelques unes dans cette fiche, sont généralement suffisantes pour réaliser des concaténations ou des séparations de données. Toutefois, les manipulations utilisant les fonctions de base R présentent des difficultés en termes d'efficience et de lisibilité du code.
La motivation derrière la création du package {tidyr} a été de rendre le réarrangement de données plus rapide et efficace, en particulier pour la création de nouvelles variables à partir de données concaténées ou séparées. Comme on a vu dans la présente fiche, les deux fonctions `unite` et `separate` se sont montrées d'une grande utilité, puisqu'on a abouti au résultat voulu avec un traitement moindre (pratiquement une ligne de code par manipulation). En effet on peut, en plus de la concaténation / séparation, à la fois désigner les nouvelles variables à créer, choisir de garder ou non la/les variable(s) source, de convertir ou non les données réarrangées, et déterminer quels objets remplaceront les valeurs manquantes, ce qui aurait coûté un traitement supplémentaire avec les fonctions de base R.
Le seul incovéniant par rapport aux fonctions de base R, qu'on reprocherait d'ailleurs à toute fonction n'appartenant pas à ce dernier, est l'indisponibilité par défaut sur R ou RStudio ; si par malheur on se retrouve quelque part sans connexion internet pour télécharger le package...

Note concernant l'opérateur %>%

L'opérateur `%>%` passe l'objet se trouvant à gauche comme premier argument de la fonction se trouvant à droite, cet opérateur permet d'économiser du temps et de rendre le code plus lisible quand il s'agit d'imbriquer des fonctions dans d'autres.
Pour illustrer l'utilité de cet opérateur, on va essayer de séparer notre variable `Score`, puis d'extraire ces variables éclatées et afficher les trois premières lignes en appliquant simultanément les fonctions `separate`, `select` et `head`.
Avant d'appliquer la fonction `select` il faut d'abord faire appel au package {dplyr}.
L'utilisation de cet opérateur dans notre cas se fait de la manière suivante :

```
library(dplyr)

cbind(tennis,Score) %>%
  separate(Score, into = sprintf("ST%d",1:5), sep=', ', fill="right" ,remove=FALSE) %>%
  select(num_range("ST",1:5)) %>%
  head(3)
```

```
##   ST1 ST2 ST3  ST4  ST5
## 1 6-4 6-3 6-2 <NA> <NA>
## 2 6-7 6-4 7-6  7-6 <NA>
## 3 6-2 6-4 6-3 <NA> <NA>
```

Références

- Documentation sur les fonctions `paste` et `paste0` : https://stat.ethz.ch/R-manual/R-devel/library/base/html/paste.html
- Package {stringr} : https://cran.r-project.org/web/packages/stringr/stringr.pdf
- Documentation officielle du package {tidyr} https://cran.r-project.org/web/packages/tidyr/tidyr.pdf
- Documentation plus détaillée : https://rpubs.com/bradleyboehmke/data_wrangling
- Pour les fans des vidéos : https://www.rstudio.com/resources/webinars/data-wrangling-with-r-and-rstudio/
- Une fiche résumée des fonctions de {dplyr} et {tidyr} : https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

- La fonction `select` : http://www.rdocumentation.org/packages/dplyr/functions/select