Transformer un jeu de données en forme large ou longue : base R vs le package tidyr

Hans Ivers & Mohammadsadegh Shokrizadeh

2016-04-08

Table des matières

Introduction

Formes large et longue

Les bases de données peuvent représenter leur contenu d'information sous deux formats : (a) la forme **large**, où chaque variable est localisée dans une colonne distincte, et (b) la forme **longue**, également connue sous le nom de modèle *Entité - attribut - valeur*, où chaque ligne représente une combinaison de l'identifiant unique de l'observation, de l'attribut (la variable) ainsi que sa valeur (voir Figure 1).

La première forme est fréquemment utilisée dans les logiciels statistiques comme SAS (Delwiche & Slaughter, 2012), ou dans les bases de données relationnelles, alors que la seconde forme se retrouve à la base de la représentation des données sous le formalisme NoSQL (Risch, 2013). Cette seconde forme présente de nombreux avantages, comme une flexibilité supérieure pour ajouter de nouvelles variables et une rapidité plus grande pour mettre à jour ce type de format, au détriment d'une possible perte d'information car le type de variable n'est pas nécessairement représenté dans la combinaison "attribut-valeur" (dans les cas comme le présent exemple où plusieurs variables distinctes sont transposées).

Objectifs de ce document

Les objectifs de ce document d'information sont de présenter deux méthodes pour passer d'une mise en forme à une autre : (a) une méthode incluse par défaut dans R base, soit la procédure *reshape*, et (b) une méthode disponible via le package *tidyr* (Wickham, 2016) qui présente, selon notre évaluation, des avantages intéressants. Chaque méthode sera illustrée puis un comparatif de ces méthodes sera discuté.

Format large

ID	Age	Poids	Grandeur
1	24	90	1.82
2		75	1.70
3	35	62	

Format long

ID	Attribut	Valeur
1	Age	24
1	Poids	90
1	Grandeur	1.82
2	Poids	75
2	Grandeur	1.7
3	Age	35
3	Poids	62

FIGURE 1 – Représentation des formes large et longue.

Pour illustrer ces procédures, un jeu de données est tiré d'une base incluant tous les matchs de tennis en simple masculin ayant eu lieu lors du tournoi de Wimbledon en 2013. Les données sont accessibles dans le fichier *Wimbledon-men-2013.csv*, téléchargé à l'URL suivant : https://archive.ics.uci.edu/ml/machine-learning-databases/00300/ et la description des attributs est présentée à l'URL suivant : https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics.

Méthode de base pour changer de mise en forme

Illustration de la base de données

Afin d'illustrer le passage de la forme large à la forme longue à l'aide de la méthode *reshape*, la base de données (par défaut en forme large) est d'abord chargée en mémoire et une clé primaire pour identifier chaque match de façon unique (variable *id*) est ajoutée.

```
data.raw <- read.csv2(file = "Wimbledon-men-2013.csv", header = TRUE, na.strings = "", sep = ",")
id <- as.integer(rownames(data.raw))
data.raw <- cbind(data.raw,id)
```

Voici un extrait des premières lignes de cette base, pour les variables d'intérêt (noms des deux joueurs, ronde, résultat, et les scores de chaque joueur à chacun des cinq sets possibles (noms originaux des variables : Player1, Player2, Round, Result, ST1.1 à ST5.1, et ST1.2 à ST5.2) :

```
##   id   Player1         Player2 Round Result ST1.1 ST2.1 ST3.1 ST4.1 ST5.1
```

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

Passage de la forme large à la forme longue

La procédure *reshape* permet de passer d'une forme large à une forme longue, et vice-versa. Pour cette première démonstration, nous allons d'abord passer vers la forme longue.

Pour se faire, différents arguments doivent être spécifiés :

- **data** = nom du data frame qui doit être transposé
- **idvar** = la clé primaire (simple ou composée) de la base, soit l'identifiant unique de chaque ligne
- **v.names** = nom(s) que prendra la(les) variable(s) une fois transposée(s). Dans le cas où il y a plus qu'une variable à transposer, la liste des variables doit être transmise dans un vecteur de chaines de caractères (avec la fonction c()).
- **varying** = liste des variables en forme large qui devront être transposées. L'ordre des variables dans cette liste est très important : notez bien que les deux variables relatives au premier set sont listées, ensuite les deux variables relatives au second set, etc. Cet ordre est nécessaire car la transposition est réalisée par set, et non par joueur.
- **timevar** = variable en format long qui permet de différencier les multiples observations liées à la même idvar.
- **times** = valeur de la variable timevar pour chaque nouvelle observation
- **new.row.names** = (optionnel) pour indiquer le nom de chaque ligne de la forme longue
- **direction** = "long" (soit la direction de la transposition).

Selon la base de données retenue pour illustrer cette procédure, les arguments de la procédure *reshape* prendront donc les valeurs suivantes :

```
long.1 <- reshape(data = wide,
            idvar = "id",
            v.names = c("ST.1","ST.2"),
            varying = c("ST1.1","ST1.2","ST2.1","ST2.2","ST3.1","ST3.2","ST4.1","ST4.2","ST5.1","ST5.2"
            timevar = "set",
            times = 1:5,
            new.row.names = 1:570,
            direction = "long")
```

Voici un extrait des 10 dernières lignes (i.e., les deux derniers matchs) du fichier de données après transposition en forme longue.

```
##      id    Player1     Player2 Round Result set ST.1 ST.2
## 113 113 N.Djokovic J.Del Potro     6      1   1    7    5
## 227 113 N.Djokovic J.Del Potro     6      1   2    4    6
## 341 113 N.Djokovic J.Del Potro     6      1   3    7    6
```

```
## 455 113 N.Djokovic J.Del Potro     6      1   4    6    7
## 569 113 N.Djokovic J.Del Potro     6      1   5    6    3
## 114 114 N.Djokovic    A.Murray     7      0   1    4    6
## 228 114 N.Djokovic    A.Murray     7      0   2    5    7
## 342 114 N.Djokovic    A.Murray     7      0   3    4    6
## 456 114 N.Djokovic    A.Murray     7      0   4   NA   NA
## 570 114 N.Djokovic    A.Murray     7      0   5   NA   NA
```

Passage de la forme longue à la forme large

Une façon simple de revenir d'une forme longue à une forme large est simplement de resoumettre la commande *reshape (nom du dataframe en forme longue)*, **pour autant que le dataframe a été obtenu initialement à l'aide de cette même commande**, tel qu'illustré dans l'exemple suivant :

```
wide.1 <- reshape(long.1)
```

Voici le résultat obtenu pour les deux derniers matchs :

```
##      id    Player1     Player2 Round Result ST1.1 ST1.2 ST2.1 ST2.2 ST3.1
## 113 113 N.Djokovic J.Del Potro     6      1     7     5     4     6     7
## 114 114 N.Djokovic    A.Murray     7      0     4     6     5     7     4
##     ST3.2 ST4.1 ST4.2 ST5.1 ST5.2
## 113     6     6     7     6     3
## 114     6    NA    NA    NA    NA
```

Toutefois, il est possible de noter que l'ordre des variables transposées (résultats des cinq sets pour les deux joueurs) n'est pas exactement le même que dans le fichier original. Il est ici en ordre de sets puis de joueur, alors qu'il était originalement en ordre de joueur, puis de sets.

Si le fichier à transposer n'était pas déjà produit par une procédure *reshape*, il est possible d'effectuer le passage d'une forme longue vers une forme large à l'aide de quatre principaux arguments :

- **data** = nom du data frame qui doit être transposé
- **idvar** = un vecteur de chaines de caractères qui inclut l'identifiant unique de chaque ligne ainsi que tous les autres variables qui ne varient pas pour la même observation
- **timevar** = variable en format long qui permet de différencier les multiples observations liées à la même idvar.
- **direction** = "wide"

```
wide.2 <- reshape(data = long.1,
            idvar = c("id","Player1","Player2","Round","Result"),
            timevar = "set",
            direction = "wide")
```

Voici le résultat obtenu pour les deux derniers matchs :

```
##      id    Player1     Player2 Round Result ST1.1 ST1.2 ST2.1 ST2.2 ST3.1
## 113 113 N.Djokovic J.Del Potro     6      1     7     5     4     6     7
## 114 114 N.Djokovic    A.Murray     7      0     4     6     5     7     4
##     ST3.2 ST4.1 ST4.2 ST5.1 ST5.2
## 113     6     6     7     6     3
## 114     6    NA    NA    NA    NA
```

Ici encore, bien que les données soient correctement transposées, il est possible de noter que l'ordre et le nom (sous le formalisme ST[joueur].[set]) des variables transposées n'est pas exactement le même que dans le fichier original (sous le formalisme ST[set].[joueur]). Il faudra donc une étape supplémentaire (renommer et

réordonner les variables) afin de reproduire la mise en forme exacte du fichier initial. Mais dans un contexte réel d'usage de cette commande, cette étape n'est pas nécessaire car, par définition, le fichier original en format long n'existe pas, notre objectif étant plutôt de le créer.

Méthode alternative à l'aide du package *tidyr*

Le package *tidyr*, développé et maintenu par Hadley Wickham (dernière version en mars 2016), est une amélioration du package *reshape2* par le même auteur (disponible depuis 2007), lui-même développé pour améliorer les fonctionnalités de la procédure *reshape* de R Base. Cette "troisième" génération de package vise également à inclure l'usage du puissant opérateur de redirection de flux (%>%), historiquement inventé pour les systèmes d'exploitation dérivés de Unix/Linux.

Le package *tidyr* offre 13 fonctions de manipulation de données, mais seules deux fonctions seront traitées dans le présent document en lien avec les objectifs, soit la procédure **gather()**, pour passer de la forme large vers la forme longue, et la procédure **spread()**, afin de compléter le passage contraire.

Avant d'utiliser ce package, il faut évidemment s'assurer qu'il soit bien installé et chargé dans l'environnement de travail.

```
library(tidyr)
```

Passage de la forme large à la forme longue

Notons qu'à l'instar de la procédure *reshape*, les procédures du package *tidyr* assument que chaque rangée est une observation et chaque colonne est une variable. Pour passer d'une forme large à une forme longue, la procédure *gather()* utilise les arguments suivants :

- **data** = nom du data frame qui doit être transposé
- **key** = nom de colonne en forme longue qui permet de différencier les multiples observations liées (par défaut, ce sera le nom des variables transposées)
- **value** = nom de colonne qui contient les valeurs qui prendra la nouvelle variable en forme longue
- **...** = nom de colonnes (variables) en forme large dont le contenu sera transposé

Si l'opération de redirection est utilisé, l'argument **data** devient inutile. Ainsi, l'appel de la commande suivante : gather(data,key,value), est équivalent à l'appel suivant : data %>% gather(key,value). Cet opérateur permet également d'enchaîner une série de commandes, évitant à chaque fois de devoir sauvegarder le résultat dans un tableau temporaire avant de passer ce tableau à la commande suivante.

Appliqué à la base de données en exemple, la commande *gather* prend la forme suivante :

```
long.2 <- wide %>%
         gather(Set, SetRes,ST1.1:ST5.2)
```

Toutefois, un examen du début de ce fichier démontre que la transposition vers la forme longue s'est effectuée pour l'ensemble des résultats des deux joueurs.

```
##      id  Player1  Player2 Round Result   Set SetRes
## 1     1 B.Becker A.Murray     1      0 ST1.1      4
## 115   1 B.Becker A.Murray     1      0 ST2.1      3
## 229   1 B.Becker A.Murray     1      0 ST3.1      2
## 343   1 B.Becker A.Murray     1      0 ST4.1     NA
## 457   1 B.Becker A.Murray     1      0 ST5.1     NA
## 571   1 B.Becker A.Murray     1      0 ST1.2      6
```

```
## 685   1 B.Becker A.Murray     1      0 ST2.2      6
## 799   1 B.Becker A.Murray     1      0 ST3.2      6
## 913   1 B.Becker A.Murray     1      0 ST4.2     NA
## 1027  1 B.Becker A.Murray     1      0 ST5.2     NA
```

Ainsi, un peu de travail est nécessaire pour extraire de la variable *Set*, dont le format est ST[set].[joueur], le numéro du set et celui du joueur. Pour se faire, une série d'opérations réalisées à l'aide des procédures *separate()* et *unite()* qui permettent (a) de séparer les deux chaines de caractères en utilisant le point comme séparateur, (b) d'extraire le numéro de set de la première chaine de caractères, puis (c) de fusionner la chaine "ST" avec le numéro du joueur. Les commandes utilisées sont présentées ici :

```
long.2 <- wide %>%
         gather(Set, SetRes,ST1.1:ST5.2) %>%
         separate(Set, c("Set", "Player")) %>%
         separate(Set, into = c("ST", "Set"), sep = 2) %>%
         unite(Player, ST, Player, sep = ".")
```

Et le résultat du traitement des chaines de caractères est le suivant :

```
##      id  Player1  Player2 Round Result Player Set SetRes
## 1     1 B.Becker A.Murray     1      0   ST.1   1      4
## 115   1 B.Becker A.Murray     1      0   ST.1   2      3
## 229   1 B.Becker A.Murray     1      0   ST.1   3      2
## 343   1 B.Becker A.Murray     1      0   ST.1   4     NA
## 457   1 B.Becker A.Murray     1      0   ST.1   5     NA
## 571   1 B.Becker A.Murray     1      0   ST.2   1      6
## 685   1 B.Becker A.Murray     1      0   ST.2   2      6
## 799   1 B.Becker A.Murray     1      0   ST.2   3      6
## 913   1 B.Becker A.Murray     1      0   ST.2   4     NA
## 1027  1 B.Becker A.Murray     1      0   ST.2   5     NA
```

A cette étape, il ne reste plus qu'à utiliser la procédure *spread()* qui, paradoxalement est utilisée pour passer d'une forme longue à une forme large, afin de créer une colonne pour les résultats de chaque joueur. L'appel de cette procédure est très simple, et deux principaux arguments doivent être inclus :

- **key** = nom de colonne en forme longue qui est utilisée pour nommer les nouvelles variables en forme large
- **value** = nom de colonne qui contient les valeurs que prendra les nouvelles variables en forme large

```
long.2 <- wide %>%
         gather(Set, SetRes,ST1.1:ST5.2) %>%
         separate(Set, c("Set", "Player")) %>%
         separate(Set, into = c("ST", "Set"), sep = 2) %>%
         unite(Player, ST, Player, sep = ".") %>%
         spread(Player, SetRes)
```

Il est possible de constater que l'ensemble de ces opérations produit une fichier en forme longue, selon les spécifications attendues :

```
##      id    Player1     Player2 Round Result Set ST.1 ST.2
## 561 113 N.Djokovic J.Del Potro     6      1   1    7    5
## 562 113 N.Djokovic J.Del Potro     6      1   2    4    6
## 563 113 N.Djokovic J.Del Potro     6      1   3    7    6
## 564 113 N.Djokovic J.Del Potro     6      1   4    6    7
## 565 113 N.Djokovic J.Del Potro     6      1   5    6    3
## 566 114 N.Djokovic    A.Murray     7      0   1    4    6
```

```
## 567 114 N.Djokovic    A.Murray     7      0   2    5    7
## 568 114 N.Djokovic    A.Murray     7      0   3    4    6
## 569 114 N.Djokovic    A.Murray     7      0   4   NA   NA
## 570 114 N.Djokovic    A.Murray     7      0   5   NA   NA
```

Passage de la forme longue à la forme large

Comme il a été illustré dans la section précédente, le package *tidyr* permet d'utiliser de façon flexible les outils de base pour effectuer toute opération nécessaire afin de réaliser une transposition vers une forme longue. Cette même approche sera retenue pour passer de la forme longue à la forme large. En clair, les mêmes opérations peuvent être utilisées pour effectuer la procédure inverse, seul l'ordre des opérations sera nécessairement à l'inverse.

Rappelons que la forme longue se présente comme suit pour la base à titre d'exemple :

```
##    id  Player1  Player2 Round Result Set ST.1 ST.2
## 1   1 B.Becker A.Murray     1      0   1    4    6
## 2   1 B.Becker A.Murray     1      0   2    3    6
## 3   1 B.Becker A.Murray     1      0   3    2    6
## 4   1 B.Becker A.Murray     1      0   4   NA   NA
## 5   1 B.Becker A.Murray     1      0   5   NA   NA
## 6   2   J.Ward   Y-H.Lu     1      0   1    7    6
## 7   2   J.Ward   Y-H.Lu     1      0   2    4    6
## 8   2   J.Ward   Y-H.Lu     1      0   3    6    7
## 9   2   J.Ward   Y-H.Lu     1      0   4    6    7
## 10  2   J.Ward   Y-H.Lu     1      0   5   NA   NA
```

Et l'objectif est d'obtenir une forme large similaire à :

```
##   id  Player1  Player2 Round Result ST1.1 ST2.1 ST3.1 ST4.1 ST5.1 ST1.2
## 1  1 B.Becker A.Murray     1      0     4     3     2    NA    NA     6
## 2  2   J.Ward   Y-H.Lu     1      0     7     4     6     6    NA     6
##   ST2.2 ST3.2 ST4.2 ST5.2
## 1     6     6    NA    NA
## 2     6     7     7    NA
```

En examinant la transposition précédente, la liste suivante des opérations a été complétée afin de permettre le passage à la forme longue :

1. Transposer les données originales en forme large vers une forme longue (fonction gather)
2. Isoler le numéro du joueur de la variable Set
3. Isoler le numéro du set de la variable Set
4. Combiner le numéro du joueur avec un préfixe "ST." pour créer le nom des variables à transposer
5. Transposer vers une forme large les deux variables de l'étape 4 pour obtenir une colonne de résultat par joueur

Ainsi, le passage de la forme longue vers la forme large pourra simplement être complété en réalisant à rebours de l'étape 5 vers l'étape 1 :

- Transposer vers une forme longue (fonction *gather*) les deux colonnes de résultats par joueur pour obtenir une seule colonne (10 lignes par match)
- Isoler le numéro du joueur
- Créer la variable Set en combinant le numéro de set avec un préfixe "ST"
- Créer le nom de la variable finale de Set en combinant la variable Set en (3) avec le suffixe du numéro du joueur
- Transposer les résultats de l'ensemble des sets d'un match d'une forme longue vers une forme large, en utilisant la variable en (2) pour nommer les nouvelles variables de la forme large (fonction *spread*)

```r
wide.3 <- long.2 %>%
  gather(Player,SetRes,ST.1:ST.2) %>%
  separate(Player,c("ST","Player")) %>%
  unite(Set,ST,Set, sep = "") %>%
  unite(SP,Set,Player, sep = ".") %>%
  spread(SP,SetRes)
```

Il est possible de constater que cet ensemble d'opérations à rebours a permis de réaliser correctement le passgage de la forme longue vers la forme large selon les spécifiations :

```
##   id  Player1  Player2 Round Result ST1.1 ST1.2 ST2.1 ST2.2 ST3.1 ST3.2
## 1  1 B.Becker A.Murray     1      0     4     6     3     6     2     6
## 2  2   J.Ward   Y-H.Lu     1      0     7     6     4     6     6     7
##   ST4.1 ST4.2 ST5.1 ST5.2
## 1    NA    NA    NA    NA
## 2     6     7    NA    NA
```

Comparaison des deux méthodes

Les illustrations réalisées dans les sections précédentes pour la procédure *reshape* de Base R et la combinaison des procédures *gather*, *spread*, *unite* et *separate* tirées du package *tidyr* permettent de constater plusieurs avantages de ce package. En effet, la philosophie de design derrière ce package est de rendre disponible à l'analyste une série d'outils simples, qui font une seule tâche. Toutefois, en permettant de combiner de façon flexible et originale ces différents outils, les manipulations possibles sont quasi illimitées. A l'inverse, la procédure de Base R est développée avec un nombre important d'arguments, afin de permettre certains changements de forme prévus à l'avance. L'analyste qui désire donc réaliser une certaine manipulation de données qui n'était pas prévue initialement par le concepteur de cette méthode risque d'être incapable d'atteindre son objectif.

A l'instar de notre expérience, Boehmke (2015) affirme que, comparativement aux méthodes contenues dans R Base, les méthodes du package *tidyr* présentent plusieurs avantages :

1. leur syntaxe est plus cohérente
2. leur code est plus efficace
3. la disponibilité de l'opération %>% réduit la longueur et la complexité des commandes, tout en favorisant la lisibilité du code
4. il est plus facile de combiner différentes instructions

De plus, à la lecture du manuel du package *tidyr* (Wickham, 2016), il est possible de constater que d'autres avantages sont présents, dont un des principaux est la gestion des données manquantes pour les combinaisons de variables indicatrices inexistantes.

Bref, la découverte de ce package a été très stimulante et nous ouvre plusieurs portes vers un usage plus simple, flexible et efficace du logiciel R.

Références

- Boehmke, B. (2015). *Data processing with dplyr & tidyr*. Référence en ligne : http://rpubs.com/bradleyboehmke/data_wrangling.

- Delwiche, L.D., & Slaughter, S.J. (2012). *The Little SAS Book : A Primer (5th ed.)*. Cary, NC : SAS Institute Publishing.
- Risch, T. (2013). *Introduction to NoSQL databases*. Uppsala University. Référence en ligne : http://user.it.uu.se/~torer/kurser/dbt/NoSQLDatabases.pdf.
- Wickham, H. (2016). *tidyr : Easily Tidy Data with 'spread()' and 'gather()' Functions (package manual, version 0.4.1)*. Référence en ligne : https://cran.r-project.org/web/packages/tidyr/tidyr.pdf.