RStudio Blog

# Introducing tidyr

July 22, 2014 in **Packages**, **Uncategorized**

tidyr is new package that makes it easy to “tidy” your data. Tidy data is data that’s easy to work with: it’s easy to munge (with dplyr), visualise (with ggplot2 or ggvis) and model (with R’s hundreds of modelling packages). The two most important properties of tidy data are:

Each column is a variable.
Each row is an observation.

Arranging your data in this way makes it easier to work with because you have a consistent way of referring to variables (as column names) and observations (as row indices). When use tidy data and tidy tools, you spend less time worrying about how to feed the output from one function into the input of another, and more time answering your questions about the data.

To tidy messy data, you first identify the variables in your dataset, then use the tools provided by tidyr to move them into columns. tidyr provides three main functions for tidying your messy data: `gather()`, `separate()` and `spread()`.

`gather()` takes multiple columns, and gathers them into key-value pairs: it makes “wide” data longer. Other names for gather include melt (reshape2), pivot (spreadsheets) and fold (databases). Here’s an example how you might use `gather()` on a made-up dataset. In this experiment we’ve given three people two different drugs and recorded their heart rate:

```
library(tidyr)
library(dplyr)

messy <- data.frame(
  name = c("Wilbur", "Petunia", "Gregory"),
  a = c(67, 80, 64),
  b = c(56, 90, 50)
)
messy
#>      name  a  b
#> 1  Wilbur 67 56
#> 2 Petunia 80 90
#> 3 Gregory 64 50
```

We have three variables (name, drug and heartrate), but only name is currently in a column. We use `gather()` to gather the a and b columns into key-value pairs of drug and heartrate:

```
messy %>%
  gather(drug, heartrate, a:b)
#>      name drug heartrate
#> 1  Wilbur    a        67
#> 2 Petunia    a        80
#> 3 Gregory    a        64
#> 4  Wilbur    b        56
#> 5 Petunia    b        90
#> 6 Gregory    b        50
```

Sometimes two variables are clumped together in one column. `separate()` allows you to tease them apart (`extract()` works similarly but uses regexp groups instead of a splitting pattern or position). Take this example from stackoverflow (modified slightly for brevity). We have some measurements of how much time people spend on their phones, measured at two locations (work and home), at two times. Each person has been randomly assigned to either treatment or control.

```
set.seed(10)
messy <- data.frame(
  id = 1:4,
  trt = sample(rep(c('control', 'treatment'), each = 2)),
  work.T1 = runif(4),
  home.T1 = runif(4),
  work.T2 = runif(4),
  home.T2 = runif(4)
)
```

To tidy this data, we first use `gather()` to turn columns `work.T1`, `home.T1`, `work.T2` and `home.T2` into a key-value pair of key and time. (Only the first eight rows are shown to save space.)

```
tidier <- messy %>%
  gather(key, time, -id, -trt)
tidier %>% head(8)
#>   id       trt     key    time
#> 1  1 treatment work.T1 0.08514
#> 2  2   control work.T1 0.22544
#> 3  3 treatment work.T1 0.27453
#> 4  4   control work.T1 0.27231
#> 5  1 treatment home.T1 0.61583
#> 6  2   control home.T1 0.42967
#> 7  3 treatment home.T1 0.65166
#> 8  4   control home.T1 0.56774
```

Next we use `separate()` to split the key into location and time, using a regular expression to describe the character that separates them.

```
tidy <- tidier %>%
  separate(key, into = c("location", "time"), sep = "\\.")
tidy %>% head(8)
#>   id       trt location time    time
#> 1  1 treatment     work   T1 0.08514
#> 2  2   control     work   T1 0.22544
#> 3  3 treatment     work   T1 0.27453
#> 4  4   control     work   T1 0.27231
#> 5  1 treatment     home   T1 0.61583
#> 6  2   control     home   T1 0.42967
#> 7  3 treatment     home   T1 0.65166
#> 8  4   control     home   T1 0.56774
```

The last tool, `spread()`, takes two columns (a key-value pair) and spreads them in to multiple columns, making "long" data wider. Spread is known by other names in other places: it's cast in reshape2, unpivot in spreadsheets and unfold in databases. `spread()` is used when you have variables that form rows instead of columns. You need `spread()` less frequently than `gather()` or `separate()` so to learn more, check out the documentation and the demos.

Just as reshape2 did less than reshape, tidyr does less than reshape2. It's designed specifically for tidying data, not general reshaping. In particular, existing methods only work for data frames, and tidyr never aggregates. This makes each function in tidyr simpler: each function does one thing well. For more complicated operations you can string together multiple simple tidyr and dplyr functions with `%>%`.

You can learn more about the underlying principles in my tidy data paper. To see more examples of data tidying, read the vignette, `vignette("tidy-data")`, or check out the demos, `demo(package = "tidyr")`. Alternatively, check out some of the great stackoverflow answers that use tidyr. Keep up-to-date with development at http://github.com/hadley/tidyr, report bugs at http://github.com/hadley/tidyr/issues and get help with data manipulation challenges at https://groups.google.com/group/manipulatr. If you ask a question specifically about tidyr on stackoverflow, please tag it with tidyr and I'll make sure to read it.

## SHARE THIS:

Reddit More

Reblog Like

12 bloggers like this.

## RELATED

tidyr 0.6.0
In "Packages"

tidyr 0.4.0
In "Packages"

tidyr 0.3.0
In "Packages"

## 5 comments

Introducing tidyr | Claudia Mihai
[…] Source: blog.rstudio.org […]

jhmaind

I like the idea expounded here. Why have I spent so much time wrestling with the use of functions for working with tables, when everything becomes so much simpler conceptually when working from the data frame representation.

I am though uncomfortable with referring to the particular form of data frame that is in mind as "tidy". The word "tidy" really does carry a lot of semantic baggage from its usage in other contexts. The table representation strikes me as even "tidier". It is just that the table representation is a much less satisfactory starting point for most types of further calculations.

My preference would be to talk of a "canonical" or "normal" representation.

Building pipelines to facilitate data analysis – O'Reilly Radar
[…] tidyr, pipelines to convert messy data into tidy (normalised) data that's easy to wrangle, visualise and model. […]

来自Data Science Central的44份资源 | 数盟
[…] Introducing tidyr […]

tidyr and pandas: Gather and

Melt | Connor Johnson
[…] or variable, and each row represents a unique record, or observation. The RStudio Blog has a great introduction to the tidyr functions gather(), separate(), and spread(). The gather() function is used to convert […]