Introducing tidyr

Mats Hansson

28 augusti 2016

Data tidying

It is often said that 80% of data analysis is spent on the cleaning and preparing data. And it's not just a first step, but it must be repeated many times over the course of analysis as new problems come to light or new data is collected. To get a handle on the problem, this paper focuses on a small, but important, aspect of data cleaning that I call data tidying: structuring datasets to facilitate analysis.

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.

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. Here's an example how you might use gather() on a made-up dataset.

```
library(tidyr)
suppressMessages(library(dplyr))
mtcars=tbl_df(mtcars)
mtcars$car <- rownames(mtcars)
mtcars <- mtcars[, c(12, 1:11)]
```

gather()

```
args(gather)
```

```
## function (data, key, value, ..., na.rm = FALSE, convert = FALSE,
##     factor_key = FALSE)
## NULL
```

```
mtcars %>%
  gather(attribute, value, -car)
```

```
## # A tibble: 352 x 3
##                   car attribute value
##                 <chr>     <chr> <dbl>
## 1           Mazda RX4       mpg  21.0
## 2       Mazda RX4 Wag       mpg  21.0
```

```
## 3         Datsun 710       mpg  22.8
## 4     Hornet 4 Drive       mpg  21.4
## 5  Hornet Sportabout       mpg  18.7
## 6             Valiant       mpg  18.1
## 7          Duster 360       mpg  14.3
## 8           Merc 240D       mpg  24.4
## 9            Merc 230       mpg  22.8
## 10           Merc 280       mpg  19.2
## # ... with 342 more rows
```

if we only use the gather function then is will combine all variable in the data frame.

if we only want to put two variable togeather then we will use. You will also see that all the other variable is still there.

```
mtcars %>%
  gather(combine_variable, value, cyl, disp, -car)
```

```
## # A tibble: 64 x 12
##                   car   mpg    hp  drat    wt  qsec    vs    am  gear  carb
##                 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1           Mazda RX4  21.0   110  3.90 2.620 16.46     0     1     4     4
## 2       Mazda RX4 Wag  21.0   110  3.90 2.875 17.02     0     1     4     4
## 3          Datsun 710  22.8    93  3.85 2.320 18.61     1     1     4     1
## 4      Hornet 4 Drive  21.4   110  3.08 3.215 19.44     1     0     3     1
## 5   Hornet Sportabout  18.7   175  3.15 3.440 17.02     0     0     3     2
## 6             Valiant  18.1   105  2.76 3.460 20.22     1     0     3     1
## 7          Duster 360  14.3   245  3.21 3.570 15.84     0     0     3     4
## 8           Merc 240D  24.4    62  3.69 3.190 20.00     1     0     4     2
## 9            Merc 230  22.8    95  3.92 3.150 22.90     1     0     4     2
## 10           Merc 280  19.2   123  3.92 3.440 18.30     1     0     4     4
## # ... with 54 more rows, and 2 more variables: combine_variable <chr>,
## #   value <dbl>
```

To just select some variable in the data fram and combine them, then we can first select the variable.

```
mtcars %>%
  select(car, disp, am) %>%
  gather(combine_variable, value,-car)
```

```
## # A tibble: 64 x 3
##                   car combine_variable value
##                 <chr>            <chr> <dbl>
## 1           Mazda RX4             disp 160.0
## 2       Mazda RX4 Wag             disp 160.0
## 3          Datsun 710             disp 108.0
## 4      Hornet 4 Drive             disp 258.0
## 5   Hornet Sportabout             disp 360.0
## 6             Valiant             disp 225.0
## 7          Duster 360             disp 360.0
## 8           Merc 240D             disp 146.7
## 9            Merc 230             disp 140.8
## 10           Merc 280             disp 167.6
## # ... with 54 more rows
```

Save it in a new data frame

```
new_cars=mtcars %>%
  select(car, disp, am) %>%
  gather(combine_variable, value,-car)
```

To combine more variable in a row,thenwe can use :

```
mtcars %>%
  gather(key, value, disp:drat)
```

```
## # A tibble: 96 x 11
##                   car   mpg   cyl    wt  qsec    vs    am  gear  carb   key
##                 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
## 1           Mazda RX4  21.0     6 2.620 16.46     0     1     4     4  disp
## 2       Mazda RX4 Wag  21.0     6 2.875 17.02     0     1     4     4  disp
## 3          Datsun 710  22.8     4 2.320 18.61     1     1     4     1  disp
## 4      Hornet 4 Drive  21.4     6 3.215 19.44     1     0     3     1  disp
## 5   Hornet Sportabout  18.7     8 3.440 17.02     0     0     3     2  disp
## 6             Valiant  18.1     6 3.460 20.22     1     0     3     1  disp
## 7          Duster 360  14.3     8 3.570 15.84     0     0     3     4  disp
## 8           Merc 240D  24.4     4 3.190 20.00     1     0     4     2  disp
## 9            Merc 230  22.8     4 3.150 22.90     1     0     4     2  disp
## 10           Merc 280  19.2     6 3.440 18.30     1     0     4     4  disp
## # ... with 86 more rows, and 1 more variables: value <dbl>
```

spread

To get back to the old format in the data frame `new_cars`, then we will use `spread`

```
args(spread)
```

```
## function (data, key, value, fill = NA, convert = FALSE, drop = TRUE,
##     sep = NULL)
## NULL
```

```
new_cars %>%
  spread(combine_variable, value)
```

```
## # A tibble: 32 x 3
##                    car    am  disp
## *                <chr> <dbl> <dbl>
## 1          AMC Javelin     0 304.0
## 2   Cadillac Fleetwood     0 472.0
## 3           Camaro Z28     0 350.0
## 4    Chrysler Imperial     0 440.0
## 5           Datsun 710     1 108.0
## 6     Dodge Challenger     0 318.0
## 7           Duster 360     0 360.0
## 8         Ferrari Dino     1 145.0
## 9             Fiat 128     1  78.7
## 10           Fiat X1-9     1  79.0
## # ... with 22 more rows
```

unite

How to work with date variable

```
 ## make some data
args(unite)
```

```
## function (data, col, ..., sep = "_", remove = TRUE)
## NULL
```

```
date <- as.Date('2016-01-01') + 0:14
hour <- sample(1:24, 15)
min <- sample(1:60, 15)
second <- sample(1:60, 15)
event <- sample(letters, 15)
data <- data.frame(date, hour, min, second, event)

data_new=data %>%
  unite(datehour, date, hour,sep=' ') %>%
  unite(datatime, datehour, min, second, sep=":")

print(data_new)
```

```
##               datatime event
## 1   2016-01-01 21:16:1     l
## 2    2016-01-02 7:45:9     q
## 3   2016-01-03 10:19:4     m
## 4  2016-01-04 12:18:37     x
## 5  2016-01-05 16:56:53     h
## 6  2016-01-06 20:40:14     b
## 7    2016-01-07 1:4:52     z
## 8  2016-01-08 14:38:13     u
## 9    2016-01-09 9:6:44     v
## 10 2016-01-10 19:59:31     j
## 11   2016-01-11 6:2:34     f
## 12  2016-01-12 4:39:27     p
## 13 2016-01-13 17:34:33     e
## 14  2016-01-14 5:46:41     w
## 15 2016-01-15 22:13:49     r
```

separate

we can get back to the original data we creted:

```
args(separate)
```

```
## function (data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
##     convert = FALSE, extra = "warn", fill = "warn", ...)
## NULL
```

```
data_new %>%
  separate(datatime, c("date", "time"), sep=" " )%>%
  separate(time, c("hour", "min", "second"))
```

```
##          date hour min second event
## 1  2016-01-01   21  16      1     l
## 2  2016-01-02    7  45      9     q
## 3  2016-01-03   10  19      4     m
## 4  2016-01-04   12  18     37     x
## 5  2016-01-05   16  56     53     h
## 6  2016-01-06   20  40     14     b
## 7  2016-01-07    1   4     52     z
## 8  2016-01-08   14  38     13     u
## 9  2016-01-09    9   6     44     v
## 10 2016-01-10   19  59     31     j
## 11 2016-01-11    6   2     34     f
## 12 2016-01-12    4  39     27     p
## 13 2016-01-13   17  34     33     e
## 14 2016-01-14    5  46     41     w
## 15 2016-01-15   22  13     49     r
```

FILL()

The new fill function fills in missing observations from the last non-missing value. This is useful if you're getting data from Excel users who haven't read Karl Broman's excellent data organisation guide and leave cells blank to indicate that the previous value should be carried forward:

```
df <- tbl_df(data.frame(
  year = c(2015, NA, NA, NA),
  trt = c("A", NA, "B", NA)))


df  %>%
  fill(year, trt)
```

```
## # A tibble: 4 x 2
##    year    trt
##   <dbl> <fctr>
## 1  2015      A
## 2  2015      A
## 3  2015      B
## 4  2015      B
```

replace_na() and complete()

replace_na() makes it easy to replace missing values on a column-by-column basis:

```
df <- dplyr::data_frame(x = c(1, 2, NA), y = c("a", NA, "b"))
df %>% replace_na(list(x = 0, y = "unknown"))
```

```
## # A tibble: 3 x 2
##       x       y
##   <dbl>   <chr>
## 1     1       a
## 2     2 unknown
## 3     0       b
```

```
df%>%
  replace_na(list(x=0, y="unkown"))
```

```
## # A tibble: 3 x 2
##       x      y
##   <dbl>  <chr>
## 1     1      a
## 2     2 unkown
## 3     0      b
```

It is particularly useful when called from `complete()`, which makes it easy to fill in missing combinations of your data:

```
df <- dplyr::data_frame(group = c(1:2, 1),
                        item_id = c(1:2, 2),
                        item_name = c("a", "b", "b"),
                        value1 = 1:3,
                        value2 = 4:6)


print(df)
```

```
## # A tibble: 3 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl>     <chr>  <int>  <int>
## 1     1       1         a      1      4
## 2     2       2         b      2      5
## 3     1       2         b      3      6
```

```
args(complete)
```

```
## function (data, ..., fill = list())
## NULL
```

```
df %>%
  complete(group, nesting(item_id, item_name))
```

```
## # A tibble: 4 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl>     <chr>  <int>  <int>
## 1     1       1         a      1      4
## 2     1       2         b      3      6
## 3     2       1         a     NA     NA
## 4     2       2         b      2      5
```

```
df %>%
  complete(group, nesting(item_id, item_name), fill=list(value1=0))
```

```
## # A tibble: 4 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl>     <chr>  <dbl>  <int>
## 1     1       1         a      1      4
## 2     1       2         b      3      6
## 3     2       1         a      0     NA
## 4     2       2         b      2      5
```

```
df %>%
  complete(group, nesting(item_id, item_name), fill=list(value1=0, value2=1))
```

```
## # A tibble: 4 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl>     <chr>  <dbl>  <dbl>
## 1     1       1         a      1      4
## 2     1       2         b      3      6
## 3     2       1         a      0      1
## 4     2       2         b      2      5
```

Note how I've grouped item_id and item_name together with c(item_id, item_name). This treats them as nested, not crossed, so we don't get every combination of group, item_id and item_name, as we would otherwise:

```
df%>%
  complete(group, item_id, item_name, fill=list(value1=0, value2=0))
```

```
## # A tibble: 8 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl>     <chr>  <dbl>  <dbl>
## 1     1       1         a      1      4
## 2     1       1         b      0      0
## 3     1       2         a      0      0
## 4     1       2         b      3      6
## 5     2       1         a      0      0
## 6     2       1         b      0      0
## 7     2       2         a      0      0
## 8     2       2         b      2      5
```

unnest()

`unnest()` is out of beta, and is now ready to help you unnest columns that are lists of vectors. This can occur when you have hierarchical data that's been collapsed into a string:

```
df <- dplyr::data_frame(x = 1:2,
                        y = c("1,2", "3,4,5,6,7"))

df %>%
  mutate(y = strsplit(y, ","))
```

```
## # A tibble: 2 x 2
##       x         y
##   <int>    <list>
## 1     1 <chr [2]>
## 2     2 <chr [5]>
```

```
df %>%
  mutate(y = strsplit(y, ",")) %>%
  unnest()
```

```
## # A tibble: 7 x 2
##       x     y
##   <int> <chr>
## 1     1     1
## 2     1     2
## 3     2     3
## 4     2     4
## 5     2     5
## 6     2     6
## 7     2     7
```