Rcookbook

Andreas

29 OCT 2018

Cookbook for R

Manipulating Data Converting data between wide and long format

Converting data between wide and long format

Problem Solution Sample data tidyr From wide to long From long to wide reshape2 From wide to long From long to wide

Problem

You want to do convert data from a wide format to a long format.

Many functions in R expect data to be in a long format rather than a wide format. Programs like SPSS, however, often use wide-formatted data. Solution

There are two sets of methods that are explained below: gather() and spread() from the tidyr package. This is a newer interface to the reshape2 package. melt() and dcast() from the reshape2 package.

There are a number of other methods which aren't covered here, since they are not as easy to use: The reshape() function, which is confusingly not part of the reshape2 package; it is part of the base install of R. stack() and unstack()

Sample data

These data frames hold the same data, but in wide and long formats. They will each be converted to the other format below.

```
#Wide data
olddata_wide <- read.table(header=TRUE, text='
                           subject sex control cond1 cond2
                           1   M     7.9  12.3  10.7
                           2   F     6.3  10.6  11.1
                           3   F     9.5  13.1  13.8
                           4   M    11.5  13.4  12.9
                           ')
# Make sure the subject column is a factor
olddata_wide$subject <- factor(olddata_wide$subject)

#Long data
olddata_long <- read.table(header=TRUE, text='
                           subject sex condition measurement
                           1   M   control         7.9
                           1   M     cond1        12.3
                           1   M     cond2        10.7
                           2   F   control         6.3
                           2   F     cond1        10.6
                           2   F     cond2        11.1
                           3   F   control         9.5
                           3   F     cond1        13.1
                           3   F     cond2        13.8
                           4   M   control        11.5
                           4   M     cond1        13.4
```

```
                          4   M     cond2        12.9
                          ')
# Make sure the subject column is a factor
olddata_long$subject <- factor(olddata_long$subject)
```

tidyr

From wide to long

Use gather:

```
# olddata_wide
#>   subject sex control cond1 cond2
#> 1       1   M     7.9  12.3  10.7
#> 2       2   F     6.3  10.6  11.1
#> 3       3   F     9.5  13.1  13.8
#> 4       4   M    11.5  13.4  12.9

library(tidyr)

# The arguments to gather():
# - data: Data object
# - key: Name of new key column (made from names of data columns)
# - value: Name of new value column
# - ...: Names of source columns that contain values
# - factor_key: Treat the new key column as a factor (instead of character vector)
data_long <- gather(olddata_wide, condition, measurement, control:cond2, factor_key=TRUE)
data_long
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 2        2   F   control         6.3
#> 3        3   F   control         9.5
#> 4        4   M   control        11.5
#> 5        1   M     cond1        12.3
#> 6        2   F     cond1        10.6
#> 7        3   F     cond1        13.1
#> 8        4   M     cond1        13.4
#> 9        1   M     cond2        10.7
#> 10       2   F     cond2        11.1
#> 11       3   F     cond2        13.8
#> 12       4   M     cond2        12.9
```

In this example, the source columns that are gathered are specified with control:cond2. This means to use all the columns, positionally, between control and cond2. Another way of doing it is to name the columns individually, as in: gather(olddata_wide, condition, measurement, control, cond1, cond2)

If you need to use gather() programmatically, you may need to use variables containing column names. To do this, you should use the gather_() function instead, which takes strings instead of bare (unquoted) column names.

```
keycol <- "condition"
valuecol <- "measurement"
gathercols <- c("control", "cond1", "cond2")

gather_(olddata_wide, keycol, valuecol, gathercols)
```

Optional: Rename the factor levels of the variable column, and sort.

```
# Rename factor names from "cond1" and "cond2" to "first" and "second"

levels(data_long$condition)[levels(data_long$condition)=="cond1"] <- "first"
levels(data_long$condition)[levels(data_long$condition)=="cond2"] <- "second"


# Sort by subject first, then by condition
data_long <- data_long[order(data_long$subject, data_long$condition), ]
data_long
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 5        1   M     first        12.3
#> 9        1   M    second        10.7
#> 2        2   F   control         6.3
#> 6        2   F     first        10.6
#> 10       2   F    second        11.1
#> 3        3   F   control         9.5
#> 7        3   F     first        13.1
#> 11       3   F    second        13.8
#> 4        4   M   control        11.5
#> 8        4   M     first        13.4
#> 12       4   M    second        12.9
```

From long to wide

Use spread:

```
#olddata_long
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 2        1   M     cond1        12.3
#> 3        1   M     cond2        10.7
#> 4        2   F   control         6.3
#> 5        2   F     cond1        10.6
#> 6        2   F     cond2        11.1
#> 7        3   F   control         9.5
#> 8        3   F     cond1        13.1
#> 9        3   F     cond2        13.8
#> 10       4   M   control        11.5
#> 11       4   M     cond1        13.4
#> 12       4   M     cond2        12.9

library(tidyr)

# The arguments to spread():
# - data: Data object
# - key: Name of column containing the new column names
# - value: Name of column containing values
data_wide <- spread(olddata_long, condition, measurement)
data_wide
#>   subject sex cond1 cond2 control
#> 1       1   M  12.3  10.7     7.9
```

```
#> 2       2   F  10.6  11.1     6.3
#> 3       3   F  13.1  13.8     9.5
#> 4       4   M  13.4  12.9    11.5
```

Optional: A few things to make the data look nicer.

```
# Rename cond1 to first, and cond2 to second
names(data_wide)[names(data_wide)=="cond1"] <- "first"
names(data_wide)[names(data_wide)=="cond2"] <- "second"

# Reorder the columns
data_wide <- data_wide[, c(1,2,5,3,4)]
data_wide
#>   subject sex control first second
#> 1       1   M     7.9  12.3   10.7
#> 2       2   F     6.3  10.6   11.1
#> 3       3   F     9.5  13.1   13.8
#> 4       4   M    11.5  13.4   12.9
```

The order of factor levels determines the order of the columns. The level order can be changed before reshaping, or the columns can be re-ordered afterward.

reshape2

From wide to long

Use melt:

```
#olddata_wide
#>   subject sex control cond1 cond2
#> 1       1   M     7.9  12.3  10.7
#> 2       2   F     6.3  10.6  11.1
#> 3       3   F     9.5  13.1  13.8
#> 4       4   M    11.5  13.4  12.9

library(reshape2)

# Specify id.vars: the variables to keep but not split apart on
melt(olddata_wide, id.vars=c("subject", "sex"))
#>    subject sex variable value
#> 1        1   M  control   7.9
#> 2        2   F  control   6.3
#> 3        3   F  control   9.5
#> 4        4   M  control  11.5
#> 5        1   M    cond1  12.3
#> 6        2   F    cond1  10.6
#> 7        3   F    cond1  13.1
#> 8        4   M    cond1  13.4
#> 9        1   M    cond2  10.7
#> 10       2   F    cond2  11.1
#> 11       3   F    cond2  13.8
#> 12       4   M    cond2  12.9
```

There are options for melt that can make the output a little easier to work with:

```
data_long <- melt(olddata_wide,
                  # ID variables - all the variables to keep but not split apart on
```

```
                  id.vars=c("subject", "sex"),
                  # The source columns
                  measure.vars=c("control", "cond1", "cond2" ),
                  # Name of the destination column that will identify the original
                  # column that the measurement came from
                  variable.name="condition",
                  value.name="measurement")
data_long
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 2        2   F   control         6.3
#> 3        3   F   control         9.5
#> 4        4   M   control        11.5
#> 5        1   M     cond1        12.3
#> 6        2   F     cond1        10.6
#> 7        3   F     cond1        13.1
#> 8        4   M     cond1        13.4
#> 9        1   M     cond2        10.7
#> 10       2   F     cond2        11.1
#> 11       3   F     cond2        13.8
#> 12       4   M     cond2        12.9
```

If you leave out the measure.vars, melt will automatically use all the other variables as the id.vars. The reverse is true if you leave out id.vars.

If you don't specify variable.name, it will name that column "variable", and if you leave out value.name, it will name that column "measurement".

Optional: Rename the factor levels of the variable column.

```
# Rename factor names from "cond1" and "cond2" to "first" and "second"
levels(data_long$condition)[levels(data_long$condition)=="cond1"] <- "first"
levels(data_long$condition)[levels(data_long$condition)=="cond2"] <- "second"

# Sort by subject first, then by condition
data_long <- data_long[ order(data_long$subject, data_long$condition), ]
data_long
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 5        1   M     first        12.3
#> 9        1   M    second        10.7
#> 2        2   F   control         6.3
#> 6        2   F     first        10.6
#> 10       2   F    second        11.1
#> 3        3   F   control         9.5
#> 7        3   F     first        13.1
#> 11       3   F    second        13.8
#> 4        4   M   control        11.5
#> 8        4   M     first        13.4
#> 12       4   M    second        12.9
```

From long to wide

The following code uses dcast to reshape the data. This function is meant for data frames; if you are working with arrays or matrices, use acast instead.

```
#olddata_long
```

```
#>    subject sex condition measurement
#> 1        1   M   control         7.9
#> 2        1   M     cond1        12.3
#> 3        1   M     cond2        10.7
#> 4        2   F   control         6.3
#> 5        2   F     cond1        10.6
#> 6        2   F     cond2        11.1
#> 7        3   F   control         9.5
#> 8        3   F     cond1        13.1
#> 9        3   F     cond2        13.8
#> 10       4   M   control        11.5
#> 11       4   M     cond1        13.4
#> 12       4   M     cond2        12.9

# From the source:
# "subject" and "sex" are columns we want to keep the same
# "condition" is the column that contains the names of the new column to put things in
# "measurement" holds the measurements
library(reshape2)

data_wide <- dcast(olddata_long, subject + sex ~ condition, value.var="measurement")
data_wide
#>   subject sex cond1 cond2 control
#> 1       1   M  12.3  10.7     7.9
#> 2       2   F  10.6  11.1     6.3
#> 3       3   F  13.1  13.8     9.5
#> 4       4   M  13.4  12.9    11.5
```

Optional: A few things to make the data look nicer.

```
# Rename cond1 to first, and cond2 to second
names(data_wide)[names(data_wide)=="cond1"] <- "first"
names(data_wide)[names(data_wide)=="cond2"] <- "second"

# Reorder the columns
data_wide <- data_wide[, c(1,2,5,3,4)]
data_wide
#>   subject sex control first second
#> 1       1   M     7.9  12.3   10.7
#> 2       2   F     6.3  10.6   11.1
#> 3       3   F     9.5  13.1   13.8
#> 4       4   M    11.5  13.4   12.9
```

The order of factor levels determines the order of the columns. The level order can be changed before reshaping, or the columns can be re-ordered afterward.