Reshaping Data in R

Hadley Wickham. http://had.co.nz/cast

2005-08-25

1 Introduction

This article discusses a way to think about data reshaping and describes an implementation of these principles in a R package, `reshape`. By reshaping I mean both aggregation and restructuring. Aggregation is a common task where data is reduced into a smaller, more convenient form. One commonly used aggregation procedure is Excel's Pivot tables. Restructuring involves a similar rearrangement, but preserves all original information. Where aggregation reduces many cells in the original data set to one cell in the new dataset, reshaping preserves a one-to-one connection.

There are a number of existing R functions that can aggregate data, for example `tapply`, `by` and `aggregate`, and a function specifically for reshaping data, `reshape`. Each of these functions deals well with one or two specific scenarios, but each requires slightly different input arguments and in practice, careful thought is required to create the sequence of operations that arrange the data the way that you want. The `reshape` package overcomes these problems by using the framework described below to solve a general set of problems using just two functions, `cast` and `melt`. As you might guess from these names, the metaphor the `reshape` package uses is to melt your data into a flexible form, and then cast it into the exact form that you want.

2 Framework

To help us think about all the ways we might rearrange a data set it is useful to think about data in a somewhat unusual fashion. Usually, we think about data in terms of a matrix or data frame, where we have observations in the rows and variables in the columns. In this form it is difficult to investigate relationships between other facets of the data: between subjects, or treatments, or replicates. Reshaping the data allows to explore these other relationships while still being able to use the familiar tools that operate on columns. Reshaping is an important (but often unrecognised) part of practical data analysis and is often necessary when exploring, displaying and analysing data.

For the purposes of reshaping, we can divide the variables into two groups: identifier and measured variables.

1. Identifier, or id, variables identify the unit that measurements take place on. Id variables are usually discrete, and are typically fixed by design. In ANOVA notation (Y_ijk), id variables are the indices on the variables (i, j, k).

2. Measured variables represent what is measured on that unit (Y).

It is possible to take this abstraction a step further and say there are only id variables and a value, where the id variables now also identify what measured variable the value represents. For example, we could represent this table:

Subject	Time	Age	Weight	Height
John Smith	1	50	90	1.80
Mary Smith	1	NA	NA	1.70

as:

Subject	Time	Variable	Value
John Smith	1	Age	50
John Smith	1	Height	90
John Smith	1	Weight	1.80
Mary Smith	1	Height	1.7

Now each row represents one observation of one variable. This is what I will refer to as "molten" data. Compared to the original data set, it has a new id variable "variable", and a new column "value", which represents the value of that observation. We now have the data in a form in which there is no distinction between our original observed variables and other id variables.

3 Use

We first "melt" our data into the form described above (this step can be skipped, see `?recast` for details), and then "cast" it into the form we want. A natural way to specify how we want the result to look is to specify which variables should form the columns and which should form the rows. In the usual data frame, the "variable" id variable forms the columns, while all other id variables form the rows. When the selected variables do not uniquely identify a row, aggregation occurs, we need an aggregation function to reduce a vector to a single value. Examples later will make this concrete.

The order that the row and column variables are specified in is very important. As with a contingency table there are many possible ways of displaying the same variables, and the way they are organised reveals different patterns in the data. Variables specified first vary slowest, and those specified last vary fastest. Because comparisons between adjacent cells are made most easily, the variable you are most interested in making comparisons between should be specified last, and the early variables should be thought of as conditioning variables. An additional constraint is that displays have limited width but essentially infinite length, so variables with many levels must be specified as row variables. It is also desirable to adhere to common conventions, so where possible, "variable" should appear in the column specification.

4 Example

The reshape package is available on CRAN and can be installed using the R command `install.packages("reshape`
This section will work through some techniques using the reshape package with an example data set (`french_fries`). The data is from a sensory experiment investigating the effect of different

frying oils on the taste of french fries over time. There are three different types of frying oils (treatment), each in two different fryers (rep), tested by 12 people (subject) on 10 different days (time). The sensory attributes recorded, in order of desirability, are potato, buttery, grassy, rancid, painty flavours. The first few rows of the data look like:

	time	treatment	subject	rep	potato	buttery	grassy	rancid	painty
61	1	1	3	1.00	2.90	0.00	0.00	0.00	5.50
25	1	1	3	2.00	14.00	0.00	0.00	1.10	0.00
62	1	1	10	1.00	11.00	6.40	0.00	0.00	0.00
26	1	1	10	2.00	9.90	5.90	2.90	2.20	0.00
63	1	1	15	1.00	1.20	0.10	0.00	1.10	5.10
27	1	1	15	2.00	8.80	3.00	3.60	1.50	2.30

One of the first things we might be interested in is how balanced this design is, and whether there are many different missing values. We can investigate this using length as our aggregation function:

```
ff_d <- melt(french_fries, id=1:4)
cast(ff_d, subject ~ time, length)
```

```
subject X1 X2 X3 X4 X5 X6 X7 X8 X9 X10
      3 30 30 30 30 30 30 30 30 30  NA
     10 30 30 30 30 30 30 30 30 30  30
     15 30 30 30 30 25 30 30 30 30  30
     16 30 30 30 30 30 30 30 29 30  30
     19 30 30 30 30 30 30 30 30 30  30
     31 30 30 30 30 30 30 30 30 NA  30
     51 30 30 30 30 30 30 30 30 30  30
     52 30 30 30 30 30 30 30 30 30  30
     63 30 30 30 30 30 30 30 30 30  30
     78 30 30 30 30 30 30 30 30 30  30
     79 30 30 30 30 30 30 29 28 30  NA
     86 30 30 30 30 30 30 30 30 NA  30
```

We can also easily see the range of values that each variable takes:

```
cast(ff_d, variable ~ ., function(x) c(min=min(x), max=max(x)))
```

```
subject X1 X2 X3 X4 X5 X6 X7 X8 X9 X10
      3  0  0  0  0  0  0  0  0  0  NA
     10  0  0  0  0  0  0  0  0  0   0
     15  0  0  0  0  5  0  0  0  0   0
     16  0  0  0  0  0  0  0  1  0   0
     19  0  0  0  0  0  0  0  0  0   0
     31  0  0  0  0  0  0  0  0 NA   0
```

```
51  0  0  0  0  0  0  0  0  0   0
52  0  0  0  0  0  0  0  0  0   0
63  0  0  0  0  0  0  0  0  0   0
78  0  0  0  0  0  0  0  0  0   0
79  0  0  0  0  0  0  1  2  0  NA
86  0  0  0  0  0  0  0  0 NA   0
```

Since the data is fairly well balanced, we can do some (crude) investigation as to the effects of the different treatments. For example, we can calculate the overall means for each sensory attribute for each treatment:

```
cast(ff_d, treatment ~ variable, mean,
margins=c("grand_col", "grand_row"))
```

```
variable  max min
 buttery 11.2   0
  grassy 11.1   0
  painty 13.1   0
  potato 14.9   0
  rancid 14.9   0
```

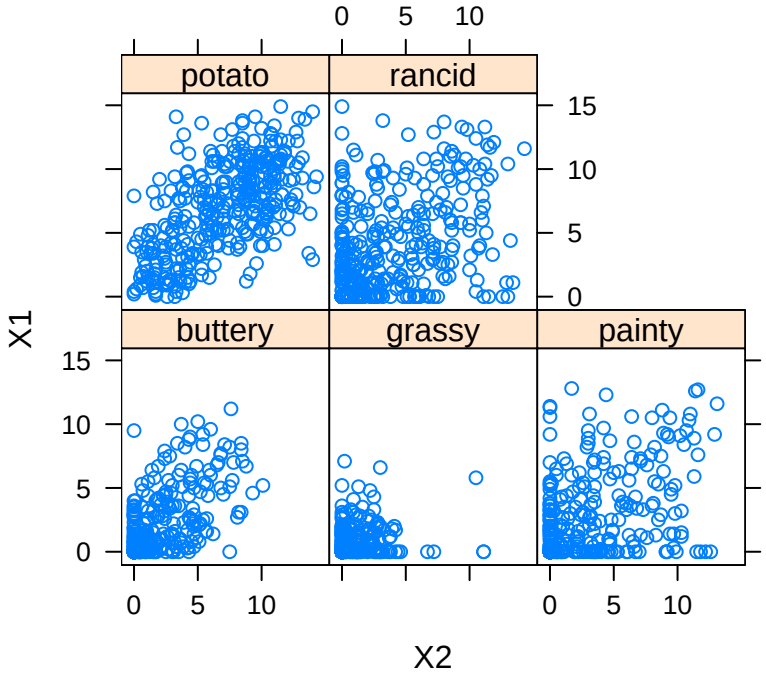
Note the row and column margins. We can also produce margins at different levels. The following example shows the results broken down for subjects 3 and 11, with both overall means and means for each subject:

```
cast(ff_d, treatment + subject ~ variable, mean,
margins="treatment", subset=subject %in% c(3,10))
```

```
treatment potato buttery grassy rancid painty  NA.
        1   6.89    1.78  0.649   4.07   2.58 3.19
        2   7.00    1.97  0.663   3.62   2.46 3.15
        3   6.97    1.72  0.681   3.87   2.53 3.15
        .   6.95    1.82  0.664   3.85   2.52 3.16
```

Finally, since we have repetition over treatments, we might be interested in how reliable each subject is – are the scores for the two reps highly correlated? We can explore this graphically by reshaping the data and using a lattice plot. Our graphical tools work best when the things we want to compare are in different columns, so we'll cast the data so we now have a column for each rep.

```
xyplot(X1   X2 | variable, cast(ff_d, ...     rep), aspect="iso")
```



If we wanted to explore the relationships between subjects or times or treatments we could follow similar steps.

5 Conclusion

This paper has presented a useful framework with which to think about reshaping data, and an intuitive implementation in R. Future work includes generalising the algorithms to deal better with non-numeric data, and large data sets. Other interesting opportunities include producing graphical summaries, and creating a GUI to make reshaping data easier. It would also be useful to be able to link in to relational databases so that as much aggregation as possible can be pushed off to a program that doesn't require all the data to fit in memory.