

Reshaping data in R

Duncan Golicher

December 9, 2008

One of the most frustrating and time consuming parts of statistical analysis is shuffling data into a format for analysis. No one enjoys changing data formats. Researchers want to get results, finish the task, move on. Routine reformatting of data is made difficult by complications that could have been avoided. Students who are not instructed in data management, or those who ignore instruction, use spreadsheets to hold data. This practice leads to confusion. The reshape package in R not only help to rescue data lost in overly complex Excel spreadsheets, it changes what could be a boring cut and paste operation into an interesting intellectual exercise in its own right.

Students should be taught to collect and hold data in a standardized “long” form in which there is only one variable for each form of measurement from the start. This resolves most problems and saves hours of tutors time. However even if they do, they still need help in reshaping this data to their own needs. This is especially true if they use SPSS. Data reformatting in SPSS is very complex and confusing and SPSS does not have powerful generic routines for handling very complex operations.

There are many tricks for handling data in R using `tapply`, `lapply`, `stack`, `aggregate`, `by` and a range of other functions. However R now has an efficient and powerful syntax for data handling provided by Hadley Wickham’s elegant reshape package. I have only just discovered reshape and I am very impressed. Reshaping operations were always far easier in R than in a spreadsheet, but that wasn’t to say that they were ever that easy. With reshape they are a lot simpler. I now think I can safely trust my students who use R to be able to handle most of the cases they come accross with a little thought and guidance. As an example here is some data extracted from tables of goals scored in the 2006 -2007 and 2007-2008 football season.

http://www.soccerstats.com/adv_scoring_times.asp?league=england

The number of goals scored by each team are identified by the fifteen minute period of the match (0-15 = m15, 15-30 =m30 etc) and the year when the season started. This is not raw data. That would consist of the exact times the goals are scored identified by team. The data tables available on the web site are in a wide format. There was an issue caused by the fact the original tables hold for and against goals together. This had to be resolved using `strsplit`. However this is a different matter that I don’t go into here. The reshaped data looks like this. Note how `xtable` can provide nicely formatted tables for reports.

```
d <- read.csv(url("http://duncanjg.files.wordpress.com/2008/12/footlong.doc"))
d$Year <- as.factor(d$Year)
str(d)

'data.frame':      240 obs. of  4 variables:
 $ Year : Factor w/ 2 levels "2006","2007": 1 1 1 1 1 1 1 1 1 1 ...
 $ Team : Factor w/ 24 levels " Arsenal"," Aston Villa",...: 1 2 3 4 5 6 7 8 9 10 ...
 $ Time : Factor w/ 6 levels "m15","m30","m45",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Goals: int   8 6 5 6 8 3 6 9 9 5 ...
```

The head of the table.

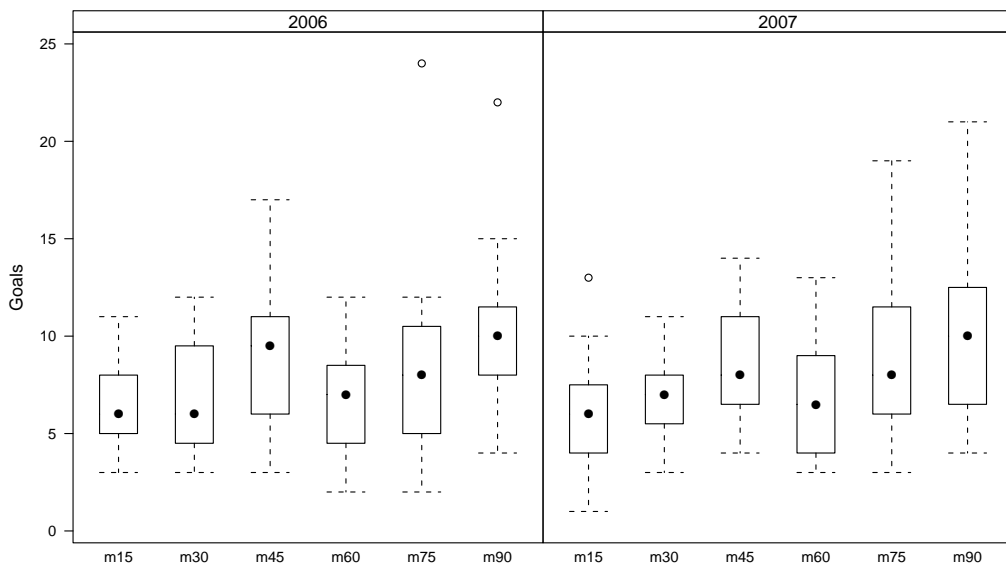
```
library(xtable)
library(reshape)
xtable(head(d, 25))
```

	Year	Team	Time	Goals
1	2006	Arsenal	m15	8
2	2006	Aston Villa	m15	6
3	2006	Birmingham City	m15	5
4	2006	Blackburn	m15	6
5	2006	Bolton	m15	8
6	2006	Charlton	m15	3
7	2006	Chelsea	m15	6
8	2006	Everton	m15	9
9	2006	Fulham	m15	9
10	2006	Liverpool	m15	5
11	2006	Manchester City	m15	8
12	2006	Manchester Utd	m15	11
13	2006	Middlesbrough	m15	6
14	2006	Newcastle	m15	3
15	2006	Portsmouth	m15	4
16	2006	Sunderland	m15	5
17	2006	Tottenham	m15	4
18	2006	West Bromwich	m15	5
19	2006	West Ham Utd	m15	6
20	2006	Wigan Athletic	m15	8
21	2006	Arsenal	m30	12
22	2006	Aston Villa	m30	6
23	2006	Birmingham City	m30	4
24	2006	Blackburn	m30	5
25	2006	Bolton	m30	4

Visualizing data in long format using R

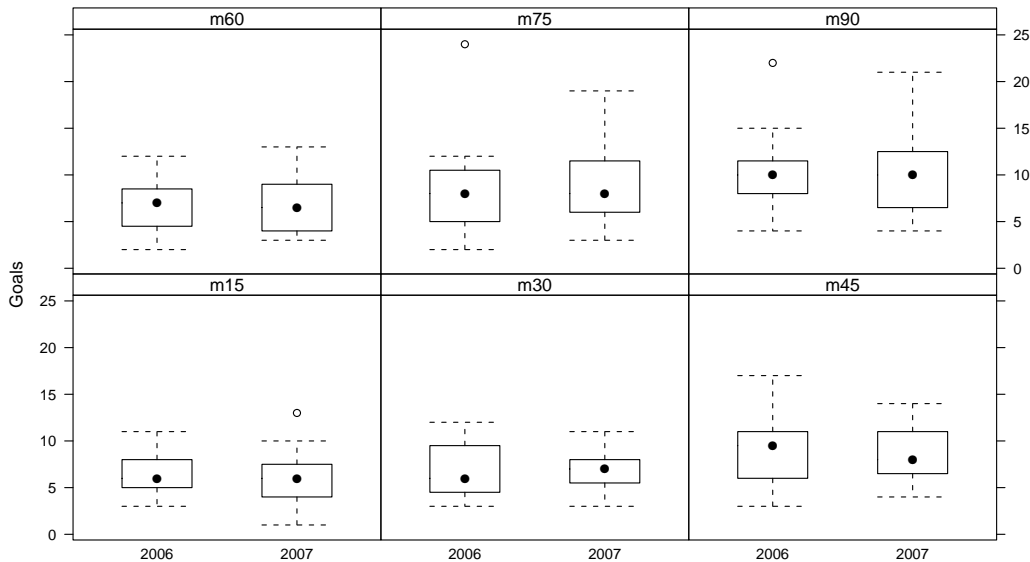
If data is in this standard “long format” everything is just so easy. For example after loading lattice a box plot is produced simply by using the standard model formula.

```
library(lattice)
print(bwplot(Goals ~ Time | Year, data = d))
```



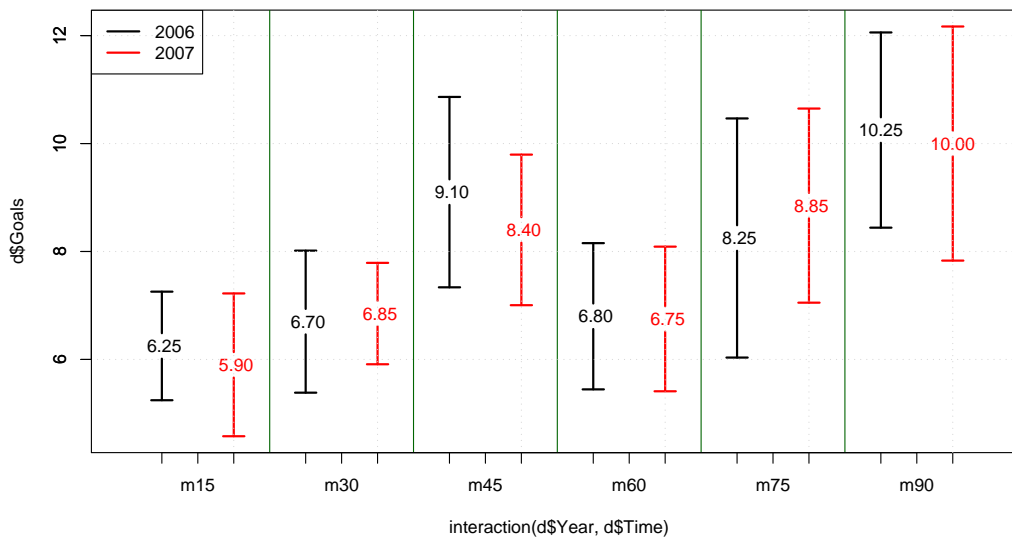
Or the boxplots can be placed the other way around.

```
print(bwplot(Goals ~ Year | Time, data = d))
```



The `plotmeans` function in `gplots` is very useful for plotting confidence intervals. Notice this is not Hadley Wickham's `ggplot` package, it is Greg Warnes `gplots`. This has a very convenient function for plotting confidence intervals for means, one of the more difficult common operations in R for students to achieve.

```
library(gplots)
plotmeans(d$Goals ~ interaction(d$Year, d$Time),
+ n.lab = FALSE, barwidth = 2, connect = F,
+ col = rep(1:2, 6), barcol = rep(1:2, 6), legends = F,
+ mean = T, pch = " ")
abline(v = c(2.5, 4.5, 6.5, 8.5, 10.5), lty = 1,
+ col = "darkgreen", lwd = 1, axes = F, xlab = "Time period",
+ ylab = "Mean number of goals with 95% confidence intervals")
grid()
axis(2)
axis(1, unique(d$Time), at = seq(1.5, 11.5, 2))
legend("topleft", lwd = 2, col = c(1:2), legend = c("2006",
+ "2007"))
```



Modelling data in long format

It is particularly nice to have data in long format because the way we visualize the data using formulae corresponds directly to a simple linear model for the data

```
mod <- lm(Goals ~ Time * Year, data = d)
xtable(anova(mod))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Time	5	477.48	95.50	8.32	0.0000
Year	1	0.60	0.60	0.05	0.8194
Time:Year	5	10.00	2.00	0.17	0.9720
Residuals	228	2617.90	11.48		

```
xtable(summary(mod))
```

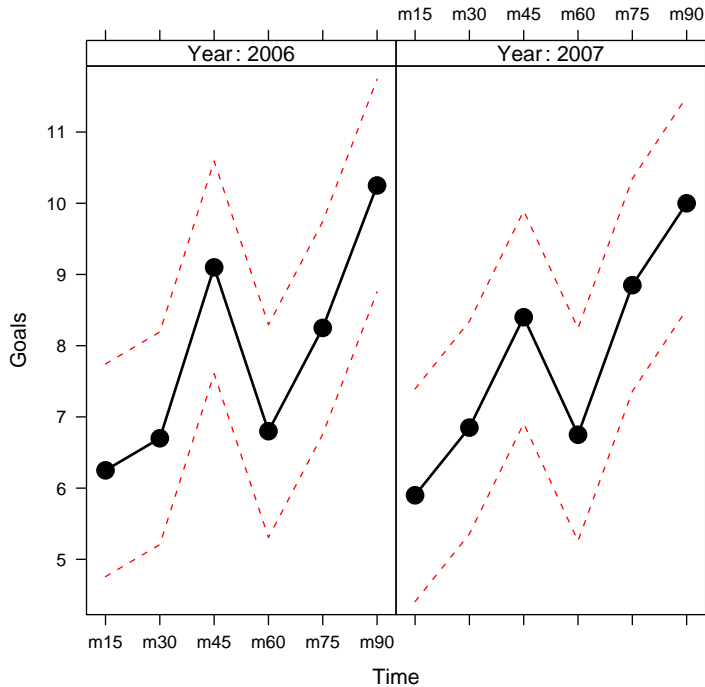
	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	6.2500	0.7577	8.25	0.0000
Timem30	0.4500	1.0715	0.42	0.6749
Timem45	2.8500	1.0715	2.66	0.0084
Timem60	0.5500	1.0715	0.51	0.6083
Timem75	2.0000	1.0715	1.87	0.0633
Timem90	4.0000	1.0715	3.73	0.0002
Year2007	-0.3500	1.0715	-0.33	0.7442
Timem30:Year2007	0.5000	1.5154	0.33	0.7417
Timem45:Year2007	-0.3500	1.5154	-0.23	0.8176
Timem60:Year2007	0.3000	1.5154	0.20	0.8432
Timem75:Year2007	0.9500	1.5154	0.63	0.5314
Timem90:Year2007	0.1000	1.5154	0.07	0.9474

There is clearly one significant main effect. This is the time in the match. Neither season nor interaction is significant.

The effects package can be used to form a very quick graphical picture of the model that is also an effective alternative to using plotmeans.

```
library(effects)
plot(effect("Time:Year", mod))
```

Time*Year effect plot



Notice that the main message is that more goals are scored before half time and at the end of the match. Some of this may be attributable to a few minutes injury time being added on, but there is an upward trend anyway in both halves.

Reshaping data

The standard “long” format unifies data analysis and visualization in R. Unfortunately we do often need to switch between this convenient long format and wider formats for presentation of data as tables and some forms of analysis.

How does Hadley Wickham’s package reshape help us to do this?

Full technical details of the package are available here

<http://www.jstatsoft.org/v21/i12/paper>

To kick start the process of using reshape I will provide practical examples as templates, although to understand its full power I recommend the original formal documentation.

The logic of the package is first to “melt” the data into a form that is consistent for any type of data. This is similar to the long form of the data except that one column is used to store the name of the variable and one for the value. You thus always have a number of columns that correspond to the identifying (grouping) variables + 2.

To melt the data you need to specify which are the variables used for identifying cases and which are the measurements. These can be specified as the names of the columns or their numbers. It is often quicker to use numbers.

```
dd <- melt(d, id = 1:3, measurement = 4)
str(dd)
```

```
'data.frame':      240 obs. of  5 variables:
 $ Year   : Factor w/ 2 levels "2006","2007": 1 1 1 1 1 1 1 1 1 1 ...
 $ Team   : Factor w/ 24 levels " Arsenal"," Aston Villa",...: 1 2 3 4 5 6 7 8 9 10 ...
 $ Time   : Factor w/ 6 levels "m15","m30","m45",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ variable: Factor w/ 1 level "Goals": 1 1 1 1 1 1 1 1 1 1 ...
 $ value  : int  8 6 5 6 8 3 6 9 9 5 ...
```

As Hadley notes, getting data into this form is not always easy. However any properly designed “long” data set will melt like butter. Wide data sets are sometimes more challenging. Once you have the data in this standard “molten” form Hadley Wickham’s formula based approach takes over and is used to reshape it to almost any form you could want. There is something quite magical about seeing data turn itself into exactly what you need with just two lines of code, especially if you have had sufficient previous experience with Excel to understand just how

time consuming it can be without reshape to help. Even with `tidy`, `gather` and `spread` in R things can often be tricky. Under Windows data can be moved from and to Excel through the clipboard, so an R window can easily do the work that pivot tables previously managed in a less elegant way.

The logic of reshape is rather challenging at first. There is no free lunch. It does take a little bit of getting used to, but once mastered it is just so powerful that you will never look back. Not only can you turn long tables into wide tables, but you can easily apply aggregations using the same logic. The trick is that the variables you want as the column names are placed in the second part of a formula. Understanding this and the how to use two special variables `..` and `...` is the key to using reshape. The `...` syntax means “all the variables” and `..` means none. This if all the variables are used to form column names a “wide” format results. Look at the example on the next pages.

Moving to a wide format

Reshape the whole table to a wide format using all the variables. Time will form the columns holding the measurements (number of goals). Notice the way ... is being used in the formula.

```
a <- cast(dd, ... ~ Time)
xtable(a)
```

	Year	Team	variable	m15	m30	m45	m60	m75	m90
1	2006	Arsenal	Goals	8	12	16	6	4	22
2	2006	Aston Villa	Goals	6	6	4	8	7	11
3	2006	Birmingham City	Goals	5	4	5	2	4	8
4	2006	Blackburn	Goals	6	5	12	9	8	11
5	2006	Bolton	Goals	8	4	11	5	9	13
6	2006	Charlton	Goals	3	6	9	5	8	10
7	2006	Chelsea	Goals	6	11	10	12	24	10
8	2006	Everton	Goals	9	5	9	4	2	5
9	2006	Fulham	Goals	9	11	7	7	5	9
10	2006	Liverpool	Goals	5	10	11	7	11	13
11	2006	Manchester City	Goals	8	6	7	4	9	9
12	2006	Manchester Utd	Goals	11	10	17	8	11	15
13	2006	Middlesbrough	Goals	6	6	10	8	9	9
14	2006	Newcastle	Goals	3	8	12	2	11	11
15	2006	Portsmouth	Goals	4	5	5	8	8	7
16	2006	Sunderland	Goals	5	3	7	3	3	4
17	2006	Tottenham	Goals	4	5	11	10	12	11
18	2006	West Bromwich	Goals	5	4	3	7	5	8
19	2006	West Ham Utd	Goals	6	9	5	11	10	12
20	2006	Wigan Athletic	Goals	8	4	11	10	5	7
21	2007	Arsenal	Goals	5	5	12	7	13	21
22	2007	Aston Villa	Goals	3	5	14	6	7	8
23	2007	Blackburn	Goals	7	10	7	9	9	10
24	2007	Bolton	Goals	6	9	7	9	9	7
25	2007	Charlton	Goals	3	6	4	5	5	11
26	2007	Chelsea	Goals	8	9	12	10	13	12
27	2007	Everton	Goals	8	4	9	10	7	14
28	2007	Fulham	Goals	4	6	5	3	8	12
29	2007	Liverpool	Goals	7	6	11	9	14	10
30	2007	Manchester City	Goals	1	3	9	4	8	4
31	2007	Manchester Utd	Goals	13	11	9	13	19	18
32	2007	Middlesbrough	Goals	6	7	7	3	13	8
33	2007	Newcastle Utd	Goals	6	7	7	5	8	6
34	2007	Portsmouth	Goals	7	8	5	9	6	10
35	2007	Reading	Goals	6	7	13	8	3	15
36	2007	Sheffield Utd	Goals	2	5	8	3	10	5
37	2007	Tottenham	Goals	10	8	11	8	8	13
38	2007	Watford	Goals	4	7	4	6	5	4
39	2007	West Ham Utd	Goals	4	8	6	4	6	7
40	2007	Wigan Athletic	Goals	8	6	8	4	6	5

Aggregating

Sum the goals for each team over the two seasons with Time as column header. In this case note that some teams only played in the premier league for one season. Reshape can quite easily produce a simple count for this by further aggregation and this can be used to subset the data.

```
a <- cast(dd, Team ~ Time, sum)
xtable(a)
```

	Team	m15	m30	m45	m60	m75	m90
1	Arsenal	13	17	28	13	17	43
2	Aston Villa	9	11	18	14	14	19
3	Birmingham City	5	4	5	2	4	8
4	Blackburn	13	15	19	18	17	21
5	Bolton	14	13	18	14	18	20
6	Charlton	6	12	13	10	13	21
7	Chelsea	14	20	22	22	37	22
8	Everton	17	9	18	14	9	19
9	Fulham	13	17	12	10	13	21
10	Liverpool	12	16	22	16	25	23
11	Manchester City	9	9	16	8	17	13
12	Manchester Utd	24	21	26	21	30	33
13	Middlesbrough	12	13	17	11	22	17
14	Newcastle	3	8	12	2	11	11
15	Newcastle Utd	6	7	7	5	8	6
16	Portsmouth	11	13	10	17	14	17
17	Reading	6	7	13	8	3	15
18	Sheffield Utd	2	5	8	3	10	5
19	Sunderland	5	3	7	3	3	4
20	Tottenham	14	13	22	18	20	24
21	Watford	4	7	4	6	5	4
22	West Bromwich	5	4	3	7	5	8
23	West Ham Utd	10	17	11	15	16	19
24	Wigan Athletic	16	10	19	14	11	12

```
b <- cast(dd, Team ~ ., length)
xtable(a[b[, 2] > 6, ])
```

	Team	m15	m30	m45	m60	m75	m90
1	Arsenal	13	17	28	13	17	43
2	Aston Villa	9	11	18	14	14	19
4	Blackburn	13	15	19	18	17	21
5	Bolton	14	13	18	14	18	20
6	Charlton	6	12	13	10	13	21
7	Chelsea	14	20	22	22	37	22
8	Everton	17	9	18	14	9	19
9	Fulham	13	17	12	10	13	21
10	Liverpool	12	16	22	16	25	23
11	Manchester City	9	9	16	8	17	13
12	Manchester Utd	24	21	26	21	30	33
13	Middlesbrough	12	13	17	11	22	17
16	Portsmouth	11	13	10	17	14	17
20	Tottenham	14	13	22	18	20	24
23	West Ham Utd	10	17	11	15	16	19
24	Wigan Athletic	16	10	19	14	11	12

Including margins

Margins can also be calculated using reshape. This is a nice feature in some cases when you need to display the data. A bad part of student's use of Excel is their frequent inclusion of marginal totals within the columns in which the data is held. This breaks a fundamental rule. Everything in a column of data should be the product of the same measurement protocol. However marginal totals are often useful when the data is presented in a final form. This is the right time to calculate them and it is easy with reshape.

```
a <- cast(dd, Team ~ Year, sum, margins = T)
xtable(a)
```

	Team	2006	2007	(all)
1	Arsenal	68	63	131
2	Aston Villa	42	43	85
3	Birmingham City	28	0	28
4	Blackburn	51	52	103
5	Bolton	50	47	97
6	Charlton	41	34	75
7	Chelsea	73	64	137
8	Everton	34	52	86
9	Fulham	48	38	86
10	Liverpool	57	57	114
11	Manchester City	43	29	72
12	Manchester Utd	72	83	155
13	Middlesbrough	48	44	92
14	Newcastle	47	0	47
15	Newcastle Utd	0	39	39
16	Portsmouth	37	45	82
17	Reading	0	52	52
18	Sheffield Utd	0	33	33
19	Sunderland	25	0	25
20	Tottenham	53	58	111
21	Watford	0	30	30
22	West Bromwich	32	0	32
23	West Ham Utd	53	35	88
24	Wigan Athletic	45	37	82
25	(all)	947	935	1882

Moving from wide to long format

This is one of the most complex operations to get right using the traditional stack. There is often a need for renaming column headings. However it works fine with reshape.

```
a <- cast(dd, ... ~ Time)
a <- data.frame(a)
head(a)
```

	Year	Team	variable	m15	m30	m45	m60	m75	m90
1	2006	Arsenal	Goals	8	12	16	6	4	22
2	2006	Aston Villa	Goals	6	6	4	8	7	11
3	2006	Birmingham City	Goals	5	4	5	2	4	8
4	2006	Blackburn	Goals	6	5	12	9	8	11
5	2006	Bolton	Goals	8	4	11	5	9	13
6	2006	Charlton	Goals	3	6	9	5	8	10

```
a <- melt(a, ids = 1:2, measurements = 4:9)
a <- cast(a)
head(a)
```

	Year	Team	variable.1	Goals
1	2006	Arsenal	m15	8
2	2006	Arsenal	m30	12
3	2006	Arsenal	m45	16
4	2006	Arsenal	m60	6
5	2006	Arsenal	m75	4
6	2006	Arsenal	m90	22

A simple summary of forest inventory data using reshape

A common task is calculating summed basal areas and counts of individuals from forest inventory data. First lets make up a very short simulated data set with just four species in ten plots. The procedure for reshaping data on 400 species in 10,000 plots is exactly the same and takes no more time, although the resulting table would make a very long appendix! Notice that the species names are first kept as a separate table then merged to form the data. It is very important always to do this to avoid mistakes. I often seen students typing species names multiple times, which is clearly a recipe for disaster.

```
sp.id <- 1:4
name <- c("Quercus segoviensis", "Pinus oocarpa",
+ "Quercus crispipilis", "Pinus maximinoi")
species <- data.frame(sp.id, name)
plot.id <- sort(sample(1:10, 200, replace = T))
sp.id <- sample(1:4, 200, replace = T)
plot.id <- sort(sample(1:10, 200, replace = T))
diam <- round(rlnorm(200, log(20), 1), 1)
plots <- data.frame(sp.id, plot.id, diam)
d <- merge(plots, species)
d <- d[order(d$plot.id), ]
head(d)
```

```
  sp.id plot.id diam      name
7     1     1 15.7 Quercus segoviensis
9     1     1 35.7 Quercus segoviensis
19    1     1  4.1 Quercus segoviensis
23    1     1 10.7 Quercus segoviensis
29    1     1  4.6 Quercus segoviensis
56    2     1 15.1      Pinus oocarpa
```

Counts of individuals are simple as the function “length” counts the number of observations in each cell. Notice that you must not include margins=T if you are going to work with the data as a matrix for multivariate analysis as this produces the row and column totals.

```
dd <- melt(d, id = c("name", "plot.id"), measure = "diam")
counts <- cast(dd, plot.id ~ name, length, margins = T)
xtable(counts)
```

	plot.id	Pinus maximinoi	Pinus oocarpa	Quercus crispipilis	Quercus segoviensis	(all)
1	1	10	6	6	5	27
2	2	4	2	2	8	16
3	3	3	7	4	6	20
4	4	9	7	2	7	25
5	5	9	6	6	2	23
6	6	1	9	5	2	17
7	7	5	2	5	7	19
8	8	5	3	3	7	18
9	9	5	6	4	5	20
10	10	4	3	3	5	15
11	(all)	55	51	40	54	200

More interestingly we can apply a function that calculates and sums basal areas directly to the melted data in the same way.

```
ba <- cast(dd, plot.id ~ name, function(x) sum(pi *
+ (x/200)^2), margins = T)
xtable(ba)
```

	plot.id	Pinus maximinoi	Pinus oocarpa	Quercus crispipilis	Quercus segoviensis	(all)
1	1	2.09	0.86	0.16	0.13	3.24
2	2	0.03	0.16	0.04	0.27	0.50
3	3	2.25	0.29	0.92	1.00	4.46
4	4	2.43	1.91	0.03	2.52	6.89
5	5	8.26	0.39	1.27	0.05	9.96
6	6	0.01	2.11	0.05	0.01	2.18
7	7	0.19	0.08	0.39	0.63	1.30
8	8	0.37	0.04	0.52	0.80	1.74
9	9	2.64	5.96	0.26	0.63	9.49
10	10	0.40	0.13	0.07	0.60	1.21
11	(all)	18.68	11.94	3.72	6.64	40.97