Creating Markup Tables with the `mutable` Package (Version 0.0-1)

Erik Iverson

September 29, 2010

1 What is `mutable`?

`mutable` is:

- An R package to create statistical tables (μ table)
- Tables are completely user-defined, and easily changed to meet the user's exact needs
- Tables can be exported to several different markup languages

2 How does `mutable` differ from other table packages?

There are already several packages on CRAN that generate tables. Several of these even generate markup so that you can display the resulting table in LaTeX or HTML.

2.1 Hmisc

Perhaps the most well-known of these functions are contained in the `Hmisc` package, through the `summary.formula` and `latex` functions. I have used these functions for a long time to generate LaTeX tables for my statistical reports. They have served me and countless others very well, and will continue to do so.

However, I continually ran into problems when the output wasn't just as I had hoped. There are `latex` methods for the objects created by `summary.formula`. The general approach is to create an object using summary.formula, and then pass it to the latex function.

There are literally dozens of parameters to the latex function, many of which control some small piece of the resulting latex output. For example, there is a first.hline.double parameter to control the placment of a horizontal line, and helvetica to control the default font used in the table.

The main issue with this approach is that LaTeX tables can be configured to a very fine degree, and introducing a new parameter every time you want to change a small bit of output was a painstaking process.

Another issue related to the first was that I could not obtain an R object with a representation of the LaTeX code that would be written out to a file upon calling the latex function. If this were available, I could have finely tuned the LaTeX code 'by hand' as it were, without introducing new parameters to the latex function.

More substantial changes to the output required even more challenging procedures. When dealing with categorical variables, I find it useful to prohibit pagebreaks in the middle of the levels of the variable. Adding this functionality required tracking down the appropriate places in the function to make the change, and then hoping that it didn't break anything else.

In short, these functions take an omnibus approach to generating markup. If you want to alter the resulting markup code, you have to dig into the function or introduce a new parameter.

2.2 How does mutable fix this?

The mutable package takes a very modular, function-based approach to generating markup tables. Instead of defining and setting lots of parameters to control the final output, you pass small, user-defined functions to generate each part of the markup. In general, the user makes one call to mutable for each column of the resulting table.

3 The mutable function

Since we will be constructing tables by making multiple calls to the `mutable` function, it pays to describe it here. The `mutable` function is an S3 generic function, meaning that multiple methods are implemented depending on the first argument. The two that have been implemented so far are for `formula` objects and for `function` objects. Let us first turn our attention to the formula interface.

```
mutable(formula, data, summary.function = esummary,
        format.function = eformat,
        latex.function = elatex,
        html.function = ehtml,
        colname,
        subset = NULL, ...)
```

A summary of the arguments is given in 3

For example, the following simple table was generated with mutable.

```
> pead.bl <- data.frame(hiv = sample(c("Positive",
+     "Negative"), 100, replace = TRUE), age = rnorm(100,
```

Table 1: Arguments to `mutable.formula`

argument	description	default
formula*	an R formula, where the left-hand side is a stratification variable and the right-hand side is list of variables to be included as the rows in the table	NA
data*	the data.frame where variables in `formula` are found	NA
summary.function	a function to summarize each variable. Can be generic to accomodate different types of data	esummary
format.function	function that will generate plain text the object from `summary.function`	eformat
latex.function	function that will generate latex markup from `summary.function`	elatex
html.function	function that will generate HTML markup from `summary.function`	ehtml
colname*	a character string specifying the name of the column	NA
subset	a subset to use of `data`	NULL

Variable	Combined Categories	Positive	Negative	P-value
Age	38.5 / 44.6 / 52.2	36.3 / 44.6 / 51.1	39 / 45 / 54.9	0.20
Gender				0.75
Female	45% 45/100	48% 22/46	43% 23/54	
Male	55% 55/100	52% 24/46	57% 31/54	
V2 GMT - V1 GMT	429 / 495.1 / 564.1	422.6 / 481 / 539.9	454.3 / 502.2 / 577.5	0.15
$\frac{V1GMT}{V2GMT} >= 4$				0.53
No	52% 52/100	57% 26/46	48% 26/54	
Yes	48% 48/100	43% 20/46	52% 28/54	
BMI	28.8 / 30.2 / 31.4	28.5 / 30.5 / 31.5	28.9 / 30.1 / 31.3	0.83

Table 2: Baseline Table

```
+     c(50, 40), sd = 10), gender = sample(c("Male",
+     "Female"), 100, replace = TRUE), diffs = rnorm(100,
+     500, sd = 100), inc4x = sample(c("Yes", "No"),
+     100, replace = TRUE), bmi = rnorm(100, 30,
+     sd = 2))
> label(pead.bl$hiv) <- "HIV Status"
> label(pead.bl$age) <- "Age"
> label(pead.bl$gender) <- "Gender"
> label(pead.bl$diffs) <- "V2 GMT - V1 GMT"
> label(pead.bl$inc4x) <- "$\\frac{V1 GMT}{V2 GMT} >= 4$"
> label(pead.bl$bmi) <- "BMI"
```

We define a table by calling mutable once for each column. We combine columns into single table object with the '+' operator. We need a formula and a data.frame, which will proprogate from the first column to subsequent columns if they are not specified by further columns.

Each column needs a summary.function, and functions to generate the markup. There are defaults for these. Finally, the colname argument must be specified.

```
> form <- hiv ~ age + gender + diffs + inc4x + bmi
> tab1 <- etable(form, data = pead.bl, colname = "Variable",
+     summary.function = erownames) + etable(colname = "Combined Categories") +
+     etable(subset = hiv == "Positive", colname = "Positive") +
+     etable(subset = hiv == "Negative", colname = "Negative") +
+     etable(summary.function = etest, colname = "P-value")
> latex(tab1, caption = "Baseline Table")
```

This may seem like a lot of work to generate the table, but for common types of tables such as this, high-level wrapper functions can be constructed to do all the work for you. Something else has also happened, The tab1 object also contains an HTML version of the table.