The **tables** Package.*

Duncan Murdoch

May 21, 2018

## Contents

1 Introduction 2

2 Reference 4

2.1 Function syntax 5

2.1.1 `tabular()` 5

2.1.2 `format()`, `print()`, `latex()` 6

2.1.3 `as.matrix()`, `write.csv.tabular()`, `write.table.tabular()` 6

2.1.4 `as.tabular()` 7

2.1.5 `table_options()`, `booktabs()` 7

2.1.6 `latexNumeric()` 12

2.2 Operators 12

2.2.1 $e_1 + e_2$ 12

2.2.2 $e_1 * e_2$ 13

2.2.3 $e_1 \sim e_2$ 13

2.2.4 $e_1 = e_2$ 14

2.3 Terms in Formulas 14

2.3.1 Closures or other functions 14

2.3.2 Factors 15

2.3.3 Logical vectors 15

2.3.4 Language Expressions 16

2.3.5 Other vectors 16

2.4 "Pseudo-functions" 17

---

*This vignette was built using `tables` version 0.8.4
This is a short introduction to the \texttt{tables} package. Inspired by my 20 year old memories of SAS PROC TABULATE, I decided to write a simple utility to create nice looking tables in Sweave documents. For example, we might display summaries of some of Fisher’s iris data using the code

\begin{verbatim}
> tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
+ (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
\end{verbatim}

\begin{verbatim}
Sepal.Length  Sepal.Width
Species    n  mean    sd  mean   sd
setosa   50  5.01  0.35 3.43  0.38
versicolor 50  5.94  0.52 2.77  0.31
virginica 50  6.59  0.64 2.97  0.32
All      150  5.84  0.83 3.06  0.44
\end{verbatim}
You can also pass the output through the `Hmisc::latex()` function (Harrell, 2011; Harrell et al., 2011) to produce \LaTeX output, which when processed by `pdflatex` will produce the following table:

\begin{verbatim}
<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>mean</td>
</tr>
<tr>
<td>setosa</td>
<td>50</td>
<td>5.01</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
<td>5.94</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
<td>6.59</td>
</tr>
<tr>
<td>All</td>
<td>150</td>
<td>5.84</td>
</tr>
</tbody>
</table>
\end{verbatim}

If you prefer the style of table that the \LaTeX \texttt{booktabs} package (Fear, 2005) produces, you can choose that style instead. I mostly like it, so I have used

\begin{verbatim}
> booktabs()
\end{verbatim}

for the rest of this document. This gives

\begin{verbatim}
<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>mean</td>
</tr>
<tr>
<td>setosa</td>
<td>50</td>
<td>5.01</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
<td>5.94</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
<td>6.59</td>
</tr>
<tr>
<td>All</td>
<td>150</td>
<td>5.84</td>
</tr>
</tbody>
</table>
\end{verbatim}

Details on \texttt{booktabs()} are given in section 2.1.5 below.

There is also the \texttt{html.tabular} method for the `Hmisc::html()` generic; it produces output in HTML format. Finally, see section 2.1.3 for other output formats.

The idea of a table in the \texttt{tables} package is a rectangular array of values, with each row and column labelled, and possibly with groups of rows and groups of columns also labelled. These arrays are specified by “table formulas”.

Table formulas are R formula objects, with the rows of the table described before the tilde (“\~{}”), and the columns after. Each of those is an expression containing “\*”, “+”, “=”; as well as functions, function calls and variables, and parentheses for grouping. There are also various directives included in...
the formula, entered as “pseudo-functions”, i.e. expressions that look like function calls but which are interpreted by the \texttt{tabular()} function.

For example, in the formula

$$(\text{Species} + 1) \sim (n=1) + \text{Format(digits=2)}* (\text{Sepal.Length} + \text{Sepal.Width})*(\text{mean} + \text{sd})$$

the rows are given by \(\text{Species} + 1\). The summation here is interpreted as concatenation, i.e. this says rows for \text{Species} should be followed by rows for 1.

In the \texttt{iris} dataframe, \texttt{Species} is a factor, so the rows for it correspond to its levels.

The 1 is a place-holder, which in this context will mean “all groups”.

The columns in the table are defined by

$$(n=1) + \text{Format(digits=2)}* (\text{Sepal.Length} + \text{Sepal.Width})*(\text{mean} + \text{sd})$$

Again, summation corresponds to concatenation, so the first column corresponds to \(n=1\). This is another use of the placeholder, but this time it is labelled as \texttt{n}. Since we haven’t specified any other statistic to use, the first column contains the counts of values in the dataframe in each category.

The second term in the column formula is a product of three factors. The first, \texttt{Format(digits=2)}, is a pseudo-function to set the format for all of the entries to come. (For more on formats, see section \texttt{2.4.1} below.) The second factor, \(\text{Sepal.Length} + \text{Sepal.Width}\), is a concatenation of two variables. Both of these variables are numeric vectors in \texttt{iris}, and they each become the variable to be analyzed, in turn. The last factor, \(\text{mean} + \text{sd}\) names two R functions. These are assumed to be functions that operate on a vector and produce a single value, as \texttt{mean} and \texttt{sd} do. The values in the table will be the results of applying those functions to the two different variables and the subsets of the dataset.

2 Reference

For the examples below we use the following definitions:

\begin{verbatim}
> set.seed(100)
> X <- rnorm(10)
> X
\end{verbatim}
A <- sample(letters[1:2], 10, rep=TRUE)
A
[1] "b" "b" "b" "b" "a" "a" "b" "b" "b" "a"
F <- factor(A)
F
[1] b b b b a a b b b a
Levels: a b

2.1 Function syntax

2.1.1 tabular()

The `tabular` function is a generic function. The default method uses `as.formula()` to try to convert the `table` argument to a formula, then passes it and all the other arguments to `tabular.formula()` method, which does most of the work. That method has 4 arguments plus `...`, but usually only the first two are used, and a warning is issued if anything is passed in the `...` arguments.

table The `table` argument is the table formula, described in detail below.

data The `data` argument is a dataframe or environment in which to look for the data referenced by the table.

n The `tabular` function needs to know the length of vectors on which it operates, because some formulas (e.g. `1 ~ 1`) contain no data. Normally `n` is taken as the number of rows in `data`, or the length of the first referenced object in the formula, but sometimes the user will need to specify it. Once specified, it can’t be modified: all data in the table should be the same length.
**suppressLabels** By default, `tabular` adds a row or column label for each term, but this does sometimes make the table messy. Setting `suppressLabels` to a positive integer will cause that many labels to be suppressed at the start of each term. The pseudo-function `Heading()` can achieve the same effect, one term at a time.

The value returned is a list-mode matrix corresponding to the entries in the table, with a number of attributes to help with formatting. See the `?tabular` help page for more details.

### 2.1.2 format(), print(), latex()

```r
format(x, digits=4, justification="n", ...)  
print(x, ...)  
latex(x, file="", options=NULL, ...)
```

The `tables` package provides methods for the `format()`, `print()` and `Hmisc::latex()` generics. The arguments are:

- `x`: The tabular object returned from `tabular()`.
- `digits`: The default number of digits to use when formatting.
- `justification`: The default text justification to use when printing. For text display, the recognized values are "n", "l", "c", "r", standing for none, left, center and right justification respectively. For LATEX the justification is specified via the `table_options()` function (section 2.1.5).

- `file`: The default method for the `Hmisc::latex()` generic writes the LATEX code to a file; `latex.tabular()` can optionally do the same, but it defaults to writing to screen, for use in Sweave documents like this one.

- `options`: A list of options to pass to `table_options()`. These will be set only for the duration of the call to `latex()`.

### 2.1.3 as.matrix(), write.csv.tabular(), write.table.tabular()

```r
as.matrix(x, format = TRUE,  
           rowLabels = TRUE, colLabels = TRUE, justification = "n", ...)  
write.csv.tabular(x, file = "", justification = "n", row.names=FALSE,  
```

6
write.options=list(), ...)  
write.table.tabular(x, file="",  
    justification = "n", row.names=FALSE, col.names=FALSE,  
    write.options=list(), ...)  

These functions export tables for further computations. The arguments are:

- **x** The tabular object.
- **format** Whether to format the entries. See the help page for alternatives.
- **rowLabels, colLabels** If formatting, whether to include the labels or not.
- **justification** The default text justification to use when formatting.
- **file** Where to write the output.
- **row.names, col.names, write.options** Additional parameters to pass to write.csv() or write.table().

### 2.1.4 as.tabular()

- **as.tabular(x, ...)**
- **as.tabular.default(x, like=NULL, ...)**
- **as.tabular.data.frame(x, ...)**

These functions create tables from existing matrices or dataframes of values. The dimnames of the input are used to construct default row and column names. If more elaborate labelling is wanted, use a tabular object as the like argument. The labelling for like will be used on the newly constructed result.

### 2.1.5 table_options(), booktabs()

The table_options() function sets a number of formatting defaults for the latex() method:

- **justification** This is the default justification for data columns and their headers. Any justification string will be accepted; it should be one that the \LaTeX \texttt{tabular} environment (or substitute) accepts. If a vector of strings is specified they will be recycled across the columns of the table.
rowlabeljustification This is the default justification for row labels. A vector of strings will be recycled across the row label columns.

tabular The environment to use in \LaTeX. Alternatives to "tabular" such as "longtable" can be used here. Those often also need modifications within the table; the \texttt{Literal() (section 2.5.4)} function may be helpful.

toprule, midrule, bottomrule The \LaTeX macros to draw the top, middle and bottom lines in the table. By default these are all "\hline".

titlerule An optional \LaTeX macro to draw a line under multicolumn titles.

doBegin, doHeader, doBody, doFooter, doEnd These logical values control the inclusion of specific parts of the output table.

The defaults are

$justification
[1] "c"

$rowlabeljustification
[1] "l"

$tabular
[1] "tabular"

$toprule
[1] "\hline"

$midrule
[1] "\hline"

$bottomrule
[1] "\hline"

$titlerule
NULL

$doBegin
[1] TRUE
$doHeader
[1] TRUE

$doBody
[1] TRUE

$doFooter
[1] TRUE

$doEnd
[1] TRUE

$knit_print
[1] TRUE

$latexleftpad
[1] TRUE

$latexrightpad
[1] TRUE

$latexminus
[1] TRUE

$doHTMLheader
[1] FALSE

$doCSS
[1] FALSE

$doHTMLbody
[1] FALSE

$CSS
[1] "<style>
ID .center { 
  text-align: center;
  background-color: aliceblue;
}</style>"

$HTMLhead
Some options only apply to HTML output; see the help page `?table_options` for details.

If you are using the \LaTeX `booktabs` package, the `booktabs()` function will set different options. Currently those are:

- `$\texttt{\textbackslash \textbackslash \textbackslash \textbackslash \textbackslash }$toprule
  [1] "\\toprule"

- `$\texttt{\textbackslash \textbackslash }$midrule
  [1] "\\midrule"

- `$\texttt{\textbackslash \textbackslash }$bottomrule
  [1] "\\bottomrule"

- `$\texttt{\textbackslash \textbackslash }$titlerule
  [1] "\\cmidrule(lr)"
The earlier table of iris data was produced using

```r
> latex(
+ tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
+ (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
+ )
```

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>n</td>
<td>mean</td>
</tr>
<tr>
<td>setosa</td>
<td>50</td>
<td>5.01</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
<td>5.94</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
<td>6.59</td>
</tr>
<tr>
<td>All</td>
<td>150</td>
<td>5.84</td>
</tr>
</tbody>
</table>

We can use the doXXXX options to insert raw \TeX into a table:

```r
> latex(tabular(Species ~ (n=1) + Format(digits=2)*
+ (Sepal.Length + Sepal.Width)*(mean + sd), data=iris),
+ options=list(doFooter=FALSE, doEnd=FALSE))
> cat("\\ \multicolumn{6}{l}{\textit{Overall, we see the following: }}\\")
> latex(tabular(1 ~ (n=1) + Format(digits=2)*
+ (Sepal.Length + Sepal.Width)*(mean + sd), data=iris),
+ options=list(doBegin=FALSE, doHeader=FALSE))
```

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>n</td>
<td>mean</td>
</tr>
<tr>
<td>setosa</td>
<td>50</td>
<td>5.01</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
<td>5.94</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
<td>6.59</td>
</tr>
<tr>
<td>All</td>
<td>150</td>
<td>5.84</td>
</tr>
</tbody>
</table>

*Overall, we see the following:*
2.1.6 \texttt{latexNumeric()}

\begin{verbatim}
latexNumeric(chars, minus = TRUE, leftpad = TRUE, rightpad=TRUE,
            mathmode = TRUE)
\end{verbatim}

The \texttt{latexNumeric()} function converts character representations of numbers into a format suitable for display in \LaTeX\ documents. There are two goals:

- If \texttt{chars} is a vector with constant width, then the output will also be constant width. This means the default centering used in \texttt{tabular()} will not misalign decimal points (if they were aligned in \texttt{chars}).
- Minus signs will be displayed with the proper symbol rather than a hyphen.

The arguments are:

- \texttt{chars} A character vector of formatted numeric values.
- \texttt{minus} Whether to pad positive cases with spacing of the same width as a minus sign. If \texttt{TRUE} and some entries are negative, then all positive entries will be padded.
- \texttt{leftpad}, \texttt{rightpad} Whether to pad cases that have leading or trailing blanks with spacing matching a digit width per space. If \texttt{leftpad=TRUE}, leading blanks will be converted to spaces the same width as a digit 0. (If \texttt{minus=TRUE}, one leading blank may have been consumed in the sign padding.) The \texttt{rightpad} argument handles trailing blanks similarly.
- \texttt{mathmode} Whether to wrap the result in dollar signs, so \LaTeX\ will render minus signs properly.

2.2 Operators

2.2.1 $e_1 + e_2$

Summing two expressions indicates that they should be displayed in sequence. For rows, this means $e_1$ will be displayed just above $e_2$; for columns, $e_1$ will be just to the left of $e_2$.

Example:
2.2.2 \( e_1 \times e_2 \)

Multiplying two expressions means that each element of \( e_1 \) will be applied to each element of \( e_2 \). If \( e_1 \) is a factor, then \( e_2 \) will be displayed for each element of it. NB: \( \times \) has higher precedence than \( + \) and evaluation proceeds from left to right. The expression \((e_1 + e_2) \times (e_3 + e_4)\) is equivalent to \(e_1 \times e_3 + e_1 \times e_4 + e_2 \times e_3 + e_2 \times e_4\).

Example:

\[
> \text{latex( tabular( F*F*\{mean + sd\} ~ 1 ) )}
\]

<table>
<thead>
<tr>
<th>F</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>All</td>
<td>10</td>
</tr>
</tbody>
</table>

2.2.3 \( e_1 \sim e_2 \)

The tilde separates row specifications from column specifications, but otherwise acts the same as \( \times \), i.e. each row value applies to each column.

Example:

\[
> \text{latex( tabular( X*F ~ mean + sd ) )}
\]

<table>
<thead>
<tr>
<th>F</th>
<th>mean</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>X a</td>
<td>0.02525</td>
<td>0.34842</td>
</tr>
<tr>
<td>b</td>
<td>-0.03647</td>
<td>0.65611</td>
</tr>
</tbody>
</table>
2.2.4 $e_1 = e_2$

The operator $\ = \ $ is used to set the name of $e_2$ to a displayed version of $e_1$. It is an abbreviation for $\text{Heading}(e_1) * e_2$. NB: because $\ =$ has lower operator precedence than any other operator, we usually put parentheses around these expressions, i.e. $(e_1 = e_2)$.

Example: F is renamed to “Newname”.

\[
> \text{latex( tabular( X*(Newname=F) ~ mean + sd ) )}
\]

<table>
<thead>
<tr>
<th>Newname</th>
<th>mean</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0.02525</td>
<td>0.3484</td>
</tr>
<tr>
<td>b</td>
<td>-0.03647</td>
<td>0.6561</td>
</tr>
</tbody>
</table>

2.3 Terms in Formulas

R parses table formulas into sums, products, and bindings separated by the tilde formula operator. What comes between the operators are other expressions. Other than the pseudo-functions described in section 2.4, these are evaluated and the actions depend on the type of the resulting value.

2.3.1 Closures or other functions

If the expression evaluates to a function (e.g. it is the name of a function), then that function becomes the summary statistic to be displayed. The summary statistic should take a vector of values as input, and return a single value (either numeric, character, or some other simple printable value). If no summary function is specified, the default is $\text{length}$, to count the length of the vector being passed.

Note that only one summary function can be specified for any cell in the table or an error will be reported.

Example: $\text{mean}$ and $\text{sd}$ are specified functions; $\text{n}$ is the renamed default statistic.

\[
> \text{latex( tabular( (F+1) ~ (n=1) + X*(mean + sd) ) )}
\]
### Factors

If the expression evaluates to a factor, the dataset is broken up into subgroups according to the levels of the factor. Most of the examples above have shown this for the factor F, but this can also be used to display complete datasets:

Example: creating a factor to show all data. Use the `identity` function to display the values in each cell.

```latex
\begin{verbatim}
> latex( tabular( (i = factor(seq_along(X))) ~ + Heading()*identity*(X+A + + (F = as.character(F) ) ) ) )
\end{verbatim}
```

<table>
<thead>
<tr>
<th>i</th>
<th>X</th>
<th>A</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.50219</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>0.13153</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>-0.07892</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>0.88678</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>0.11697</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>0.31863</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>7</td>
<td>-0.58179</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>0.71453</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>9</td>
<td>-0.82526</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>-0.35986</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

### Logical vectors

If the expression evaluates to a logical vector, it is used to subset the data.

Example: creating subsets on the fly.

```latex
\begin{verbatim}
> latex( tabular( (X > 0) + (X < 0) + 1 + + ~ ((n = 1) + X*(mean + sd)) ) )
\end{verbatim}
```
<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>mean</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &gt; 0</td>
<td>5</td>
<td>0.43369</td>
<td>0.3496</td>
</tr>
<tr>
<td>X &lt; 0</td>
<td>5</td>
<td>−0.46960</td>
<td>0.2761</td>
</tr>
<tr>
<td>All</td>
<td>10</td>
<td>−0.01796</td>
<td>0.5611</td>
</tr>
</tbody>
</table>

### 2.3.4 Language Expressions

If the expression evaluates to a language object, e.g. the result of `quote()` or `substitute()`, then it will be replaced in the table formula by its result. This allows complicated table formulas to be saved and re-used. For examples, see section [2.5](#).

### 2.3.5 Other vectors

If the expression evaluates to something other than the above, then it is assumed to be a vector of values to be summarized in the table. If you would like to summarize a factor or logical vector, wrap it in `I()` to prevent special handling.

Note that the following must all be true, or an error will be reported:

- only one value vector can be specified for any cell in the table,
- all value vectors must be the same length,
- `is.atomic()` must evaluate to `TRUE` for the vector.

Example: treating a logical vector as values.

```r
> latex( tabular( I(X > 0) + I(X < 0) + ~ ((n=1) + mean + sd) ) )
```

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>mean</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>I(X &gt; 0)</td>
<td>10</td>
<td>0.5</td>
<td>0.527</td>
</tr>
<tr>
<td>I(X &lt; 0)</td>
<td>10</td>
<td>0.5</td>
<td>0.527</td>
</tr>
</tbody>
</table>
2.4  “Pseudo-functions”

Several directives to tables may be embedded in the table formula. This is done using “pseudo-functions”. Syntactically they look like function calls, but reserved names are used. In most cases, their action applies to later factors in the term in which they appear. For example,

\[ X \cdot \text{Justify}(r) \cdot (Y + \text{Format}(\text{digits}=2) \cdot Z) + A \]

will apply the \text{Justify}(r) directive to both \( Y \) and \( Z \), but the \text{Format}(\text{digits}=2) directive will only apply to \( Z \), and neither will apply to \( A \).

2.4.1 Format()

By default tables formats each column using the standard \texttt{format()} function, with arguments taken from the \texttt{format.tabular()} call (see section \ref{section:format}).

The \texttt{Format()} pseudo-function does two things: it changes the formatting, and it specifies that all values it applies to will be formatted together. The “call” to \texttt{Format} looks like a call to \texttt{format}, but without specifying the argument \( x \). When \texttt{tabular()} formats the output it will construct \( x \) from the entries in the table governed by the \texttt{Format()} specification.

Example: The mean and standard deviation are both governed by the same format, so they are displayed with the same number of decimal places, chosen so that the smallest values (the means) show two significant digits.

\begin{verbatim}
> latex( tabular( (F+1) ~ (n=1) + + Format(digits=2)*X*(mean + sd) ) )
\end{verbatim}

\begin{table}[h]
\centering
\begin{tabular}{llll}
\hline
  & F & n & mean & sd \\
\hline
  a & 3 & 0.025 & 0.348 \\
  b & 7 & −0.036 & 0.656 \\
  All & 10 & −0.018 & 0.561 \\
\hline
\end{tabular}
\end{table}

For customized formatting, an alternate syntax is to pass a function call to \texttt{Format()}, rather than a list of arguments. The function should accept an argument named \( x \) (but as with the regular formatting, \( x \) should not be included in the formula), to contain the data. It should return a character vector of the same length as \( x \).
Example: Use a custom function and `sprintf()` to display a standard error in parentheses.

```r
> stderr <- function(x) sd(x)/sqrt(length(x))
> fmt <- function(x, digits, ...) {
+   s <- format(x, digits=digits, ...)
+   is_stderr <- (1:length(s)) > length(s) %/% 2
+   s[is_stderr] <- sprintf("$(%s)$", s[is_stderr])
+   s[!is_stderr] <- latexNumeric(s[!is_stderr])
+   s
+ }
> latex( tabular( Format(fmt(digits=1))*(F+1) ~ X*(mean + stderr) ) )
```

<table>
<thead>
<tr>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td>a</td>
<td>0.03</td>
</tr>
<tr>
<td>b</td>
<td>−0.04</td>
</tr>
<tr>
<td>All</td>
<td>−0.02</td>
</tr>
</tbody>
</table>

Character values in cells in the table are handled specially; see section 3.1 below.

2.4.2 `.Format()`

The pseudo-function `.Format()` is mainly intended for internal use. It takes a single integer argument, saying that data governed by this call uses the same formatting as the format specification indicated by the integer. In this way entries can be commonly formatted even when they are not contiguous. The integers are assigned sequentially as the format specification is parsed; users will likely need trial and error to find the right value in a complicated table with multiple formats.

Example: Format two separated columns with the same format.

```r
> latex( tabular( (F+1) ~ X*(Format(digits=2)*mean
+ + (n=1) + .Format(1)*sd) ) )
```
\begin{verbatim}
F mean n sd
a 0.025 3 0.348
b -0.036 7 0.656
All -0.018 10 0.561
\end{verbatim}

\textbf{2.4.3 Heading()}

Normally \texttt{tabular()} generates row and column labels by deparsing the expression being tabulated. These can be changed by using the \texttt{Heading()} pseudo-function, which replaces the heading on the next object found. The heading can either be a name or a string in quotes. If the \texttt{character.only} argument is \texttt{TRUE}, the expression will be evaluated to a string which will be used as a heading. \LaTeX{} codes which are not syntactically valid R can be used either in quoted strings or with \texttt{character.only = TRUE}.

If no argument is passed, the next label is suppressed.

There's an optional argument \texttt{override}, which must be either \texttt{TRUE} or \texttt{FALSE} if present. If it is \texttt{TRUE} (or not present), then the heading will override a previously specified heading. If \texttt{FALSE}, it will not. The latter seems likely only to be of use in automatically generated code, and is used in the automatically generated labels for factors.

Another optional argument is \texttt{nearData}. This is used only when two terms in a table are concatenated using +, and they don't have the same number of rows or columns. Under the default \texttt{TRUE} value, the smaller one is moved closer to the data in the table (i.e. to the right for row labels, down for column labels); if \texttt{FALSE}, it is moved in the opposite direction.

Example: Replace F with a Greek Φ, and suppress the label for X.

\begin{verbatim}
> latex( tabular( (Heading("$\Phi$")*F+1) ~ (n=1) + Format(digits=2)*Heading()*X*(mean + sd) ) )
\end{verbatim}

\begin{verbatim}
Φ n mean sd
a 3 0.025 0.348
b 7 -0.036 0.656
All 10 -0.018 0.561
\end{verbatim}

Example: Use \texttt{nearData = FALSE} to push a label away from the data:
\begin{verbatim}
> latex( tabular( X*F + Heading("near")*X +
+        Heading("far", nearData = FALSE)*X ~ mean + sd ) )
\end{verbatim}

\begin{tabular}{cccc}
  \hline
  F & mean & sd  \\
  \hline
  X & a & 0.02525 & 0.3484 \\
  & b & -0.03647 & 0.6561 \\
  & near & -0.01796 & 0.5611 \\
  & far & -0.01796 & 0.5611 \\
  \hline
\end{tabular}

2.4.4 \textbf{Justify()}

The \texttt{Justify()} pseudo-function is used to specify the text justification of the headers and data values in the table. If called with one argument, that value is used for both labels and data; if called with two arguments, the first is used for the labels, the second for the data. If no \texttt{Justify()} specification is given, the default passed to \texttt{format()}, \texttt{print()} or \texttt{latex()} will be used. Values may be specified without quotes if they are legal R names; quoted strings may also be used. (The latter is useful for \LaTeX{} output, for example \texttt{Justify("r@{}")}, to suppress column spacing on the right.)

Example:

\begin{verbatim}
> latex( tabular( Justify(r)*(F+1) ~ Justify(c)*(n=1) +
+        Justify(c,r)*Format(digits=2)*X*(mean + sd ) ) )
\end{verbatim}

\begin{tabular}{cccc}
  \hline
  X & F & n & mean & sd  \\
  \hline
  & a & 3 & 0.025 & 0.348 \\
  & b & 7 & -0.036 & 0.656 \\
  & All & 10 & -0.018 & 0.561 \\
  \hline
\end{tabular}

2.4.5 \textbf{Percent()}

The \texttt{Percent()} pseudo-function is used to specify a statistic that depends on other values in the table. It has two optional arguments:

\texttt{denom="all"} This specifies how the denominator (argument \texttt{y} to \texttt{fn} below) is set. The most commonly used values are \texttt{"all"}, meaning all values
are used, "row", meaning only the values in the current row are used, "col", meaning only the values in the current column are used.

The special syntax `Equal(...)` will record the expressions in ..., and ignore any factor based subsetting if the factor does not appear among the expressions. Similarly `Unequal(...)` will use values which differ in any of the expressions in ... from the values in the current cell.

If a logical vector is given, it is used to select which values form the denominator. Anything else is just passed to `fn` as given.

`fn=percent` This is the function which actually does the computation. The default definition is `function(x, y) 100*length(x) /length(y)`, giving the percentage count, but any other two argument function could be used.

These two examples are different ways of producing the same table:

```r
> latex( tabular( (Factor(gear, "Gears") + 1)
+    *(n=1) + Percent()
+    + (RowPct=Percent("row"))
+    + (ColPct=Percent("col"))
+    ~ (Factor(carb, "Carburetors") + 1)
+    *Format(digits=1), data=mtcars ) )

> latex( tabular( (Factor(gear, "Gears") + 1)
+    *(n=1) + Percent()
+    + (RowPct=Percent(Equal(gear))) # Equal, not "row"
+    + (ColPct=Percent(Equal(carb)))) # Equal, not "col"
+    ~ (Factor(carb, "Carburetors") + 1)
+    *Format(digits=1), data=mtcars ) )
```

1 In fact, the mechanism is more general. The expressions in `Equal(...)` or `Unequal(...)` are deparsed and treated as strings. Any logical vector elsewhere in the table may be labelled with a string using the `labelSubset` function and those labels will be respected. Unlabelled logical vectors in the table formula will always be used for subsetting.
### 2.4.6 Arguments()

The Arguments() pseudo-function is an exception to the rule that pseudo-functions apply to later factors in the table. What it does is to specify (additional) arguments to the summary function (see section [2.3.1](#)). For example, the weighted.mean() function takes two arguments: x and w. To use it in a table, you would specify the values to use as x via the usual mechanism for the analysis variable (section [2.3.5](#)), and include a term Arguments(w=weights) either before or after it. The function will be called as weighted.mean(x[subset], w=weights[subset]), where subset is a logical vector indicating which rows of data belong in the current cell.

It is actually a little more complicated than as described above. The arguments to Arguments are evaluated in full, then only those which are length n are subsetted. And if no analysis variable has been specified, but Arguments() has been, then the function will be called without the x[subset] argument. Finally, the Arguments() entry will not create a heading.

For example:
> # This is the example from the weighted.mean help page
> wt <- c(5, 5, 4, 1)/15
> x <- c(3.7, 3.3, 3.5, 2.8)
> gp <- c(1,1,2,2)
> latex( tabular( (Factor(gp) + 1)
+ ~ weighted.mean*x*Arguments(w = wt) ) )

<table>
<thead>
<tr>
<th>weighted.mean</th>
<th>gp</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3.500</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.360</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>3.453</td>
</tr>
</tbody>
</table>

The same table (without the x heading) can be produced using

> latex( tabular( (Factor(gp) + 1)
+ ~ Arguments(x, w = wt)*weighted.mean ) )

The order of the weighted.mean and Arguments() factors makes no difference.

2.4.7 DropEmpty()

DropEmpty() indicates that cells (or whole rows or columns of the table) should be dropped if they contain no observations. This will prevent ugly results like NA or NaN from showing up in the table.

This pseudo-function takes two optional arguments, which (with default value c("row", "col", "cell")) and empty (with default value ").

If the which argument contains "row", then any row in the table in which all cells are empty will be dropped. Similarly, if it contains "col", empty columns will be dropped. If it contains "cell", then cells in rows and columns that are not dropped will be set to the empty string.

For example, without using DropEmpty(), this table is ugly:

> set.seed(730)
> df <- data.frame(Label = LETTERS[1:9],
+ Group = rep(letters[1:3], each=3),
+ Value = rnorm(9),
+ stringsAsFactors = TRUE)
This looks much better:

```r
> latex( tabular( Label ~ Group*Value*mean,*
+                DropEmpty(empty="."),
+                data = df[1:6,]))
```

<table>
<thead>
<tr>
<th>Label</th>
<th>Group</th>
<th>Value mean</th>
<th>Value mean</th>
<th>Value mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>-0.92605</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>-0.69747</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>0.05293</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>NaN</td>
<td>0.7782</td>
<td>NaN</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>NaN</td>
<td>0.9822</td>
<td>NaN</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>NaN</td>
<td>-1.0628</td>
<td>NaN</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

2.5 Formula Functions

Currently several examples of formula functions are provided. Not all are particularly robust; e.g. `\texttt{Hline()}` only works for \LaTeX{} output and must be
in a particular position in the formula. Users can provide their own as well. Such functions should return a language object, which will be substituted into the formula in place of the formula function call.

2.5.1 All()

This function expands all the columns from a dataframe into separate variables in the table. It has syntax

\[ \text{All}(df, \text{numeric}=\text{TRUE}, \text{character}=\text{FALSE}, \text{logical}=\text{FALSE}, \]
\[ \text{factor}=\text{FALSE}, \text{complex}=\text{FALSE}, \text{raw}=\text{FALSE}, \text{other}=\text{FALSE}, \]
\[ \text{texify}=\text{TRUE}) \]

The arguments are

df A dataframe or matrix whose columns are to be displayed

numeric, character, logical, factor, complex and raw Whether to include columns of the corresponding types in the table.

other Whether to include columns that match none of the previous types.

texify Whether to escape \LaTeX\ special characters. See section 3.1.

If functions are given for any of the selection arguments, the columns will be transformed according to the specified function before inclusion. For example, using factor=as.character will convert factors into character vectors in the table.

Example: Show the means of the numeric columns in the iris data.

\[
\begin{align*}
> \text{latex( tabular( Species ~ \text{Heading()}*mean*All(iris), data=iris) )}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>5.006</td>
<td>3.428</td>
<td>1.462</td>
<td>0.246</td>
</tr>
<tr>
<td>versicolor</td>
<td>5.936</td>
<td>2.770</td>
<td>4.260</td>
<td>1.326</td>
</tr>
<tr>
<td>virginica</td>
<td>6.588</td>
<td>2.974</td>
<td>5.552</td>
<td>2.026</td>
</tr>
</tbody>
</table>
2.5.2 AllObs(), RowNum()

The AllObs() function displays all of the observations in a dataset. It does this by creating a factor with a different level for each observation, and a summary statistic function which just displays the observation. It works with DropEmpty() to drop rows (or columns) from the table if they correspond to non-existent observations. For example,

```r
> df <- mtcars[1:10,]
> latex( tabular(Factor(cyl)*Factor(gear)*AllObs(df) ~
+ rownames(df) + mpg, data=df) )
```

<table>
<thead>
<tr>
<th>cyl</th>
<th>gear</th>
<th>rownames(df)</th>
<th>mpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>Datsun 710</td>
<td>22.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Merc 240D</td>
<td>24.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Merc 230</td>
<td>22.8</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>Hornet 4 Drive</td>
<td>21.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valiant</td>
<td>18.1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Mazda RX4</td>
<td>21.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mazda RX4 Wag</td>
<td>21.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Merc 280</td>
<td>19.2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>Hornet Sportabout</td>
<td>18.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Duster 360</td>
<td>14.3</td>
</tr>
</tbody>
</table>

Often (as with the mtcars dataset) the full dataset takes a lot of space to display. In that case, it can be displayed in multiple columns using a combination of the AllObs() and RowNum() functions. Because this affects both rows and columns in the resulting table, the code is a little unusual. You would normally compute the RowNum() formula function outside the call to tabular(), and include it in the row specification wrapped in I() and in the column specification in the within argument to AllObs(). For example,

```r
> rownum <- with(mtcars, RowNum(list(cyl, gear)))
> latex( tabular(Factor(cyl)*Factor(gear)*I(rownum) ~
+ mpg * AllObs(mtcars, within = list(cyl, gear, rownum)),
+ data=mtcars) )
```
Despite its name, `RowNum` can be used to specify columns instead of rows, for a column-major display. In this case, its `perrow` argument should be interpreted as “per column”. For example,

```r
> rownum <- with(mtcars, RowNum(list(cyl, gear), perrow = 2))
> latex( tabular(Factor(cyl)*Factor(gear)*
+       AllObs(mtcars, within = list(cyl, gear, rownum)) ~
+       mpg * I(rownum),
+       data=mtcars) )
```

<table>
<thead>
<tr>
<th>cyl</th>
<th>gear</th>
<th>mpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>21.5</td>
</tr>
<tr>
<td>4</td>
<td>22.8</td>
<td>24.4</td>
</tr>
<tr>
<td>4</td>
<td>22.8</td>
<td>30.4</td>
</tr>
<tr>
<td>3</td>
<td>24.4</td>
<td>32.4</td>
</tr>
<tr>
<td>4</td>
<td>33.9</td>
<td>21.4</td>
</tr>
<tr>
<td>5</td>
<td>26.0</td>
<td>30.4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>21.4</td>
</tr>
<tr>
<td>4</td>
<td>21.0</td>
<td>19.2</td>
</tr>
<tr>
<td>4</td>
<td>21.0</td>
<td>17.8</td>
</tr>
<tr>
<td>5</td>
<td>19.7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>18.7</td>
</tr>
<tr>
<td>8</td>
<td>14.3</td>
<td>16.4</td>
</tr>
<tr>
<td>8</td>
<td>16.4</td>
<td>17.3</td>
</tr>
<tr>
<td>8</td>
<td>15.2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10.4</td>
<td>14.7</td>
</tr>
<tr>
<td>5</td>
<td>19.2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13.3</td>
<td>15.2</td>
</tr>
<tr>
<td>5</td>
<td>15.8</td>
<td>15.0</td>
</tr>
</tbody>
</table>

27
2.5.3 Hline()

This function produces horizontal lines in the table. It only works for LaTeX output, and must be the first factor in a term in the table formula. It has syntax

\texttt{Hline(columns)}

The argument is

columns An optional vector listing which columns should get the line.

Example:

\begin{verbatim}
> latex( tabular( Species + Hline(2:5) + 1
+ ~ Heading() * mean * All(iris), data=iris) )
\end{verbatim}

\begin{verbatim}
<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>5.006</td>
<td>3.428</td>
<td>1.462</td>
<td>0.246</td>
</tr>
<tr>
<td>versicolor</td>
<td>5.936</td>
<td>2.770</td>
<td>4.260</td>
<td>1.326</td>
</tr>
<tr>
<td>virginica</td>
<td>6.588</td>
<td>2.974</td>
<td>5.552</td>
<td>2.026</td>
</tr>
<tr>
<td>All</td>
<td>5.843</td>
<td>3.057</td>
<td>3.758</td>
<td>1.199</td>
</tr>
</tbody>
</table>
\end{verbatim}

2.5.4 Literal()

This function inserts literal text as a label. It has syntax

\texttt{Literal(x)}

The single argument is the text to insert. It is used by the Hline() function to insert the text.

2.5.5 PlusMinus()

This function produces table entries like $x \pm y$ with an optional header. It has syntax

\texttt{PlusMinus(x, y, head, xhead, yhead, digits=2, ...)}

The arguments are
x, y These are expressions which should each generate a single column in the table. The x value will be flush right, the y value will be flush left, with the ± symbol between.

head If not missing, this header will be put over the pair of columns.

xhead, yhead If not missing, these will be put over the individual columns.

digits, ... These arguments will be passed to the standard format() function.

Example: Display mean ± standard error.

> stderr <- function(x) sd(x)/sqrt(length(x))
> latex( tabular( (Species+1) ~ All(iris)*
+       PlusMinus(mean, stderr, digits=1), data=iris ) )

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>5.01 ± 0.05</td>
<td>3.43 ± 0.05</td>
<td>1.46 ± 0.02</td>
<td>0.25 ± 0.01</td>
</tr>
<tr>
<td>versicolor</td>
<td>5.94 ± 0.07</td>
<td>2.77 ± 0.04</td>
<td>4.26 ± 0.07</td>
<td>1.33 ± 0.03</td>
</tr>
<tr>
<td>virginica</td>
<td>6.59 ± 0.09</td>
<td>2.97 ± 0.05</td>
<td>5.55 ± 0.08</td>
<td>2.03 ± 0.04</td>
</tr>
<tr>
<td>All</td>
<td>5.84 ± 0.07</td>
<td>3.06 ± 0.04</td>
<td>3.76 ± 0.14</td>
<td>1.20 ± 0.06</td>
</tr>
</tbody>
</table>

2.5.6 Paste()

This function produces table entries made up of multiple values. It has syntax

\[ \text{Paste}(..., \text{head}, \text{digits}=2, \text{justify}="c", \text{prefix}="", \text{sep}="", \text{postfix}="") \]

The arguments are

... Expressions to be displayed in the columns of the table.

head If not missing, this will be used as a column heading for the combined columns.

digits Digits used in formatting. If a single value is given, all columns will be formatted in common. If multiple values are given, each is formatted separately.
justifies One or more justifications to use on the individual columns.

prefix, sep, postfix Text to use before, between, and after the columns.

Example: Display a confidence interval.

```r
> lcl <- function(x) mean(x) - qt(0.975, df=length(x)-1)*stderr(x)
> ucl <- function(x) mean(x) + qt(0.975, df=length(x)-1)*stderr(x)
> latex( tabular( (Species+1) ~ All(iris)*
+   Paste(lcl, ucl, digits=2,
+   head="95\% CI", sep="", prefix="[",
+   postfix="]"),
+   data=iris ) )
```

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>4.91, 5.11</td>
<td>3.32, 3.54</td>
<td>1.41, 1.51</td>
<td>0.22, 0.28</td>
</tr>
<tr>
<td>versicolor</td>
<td>5.79, 6.08</td>
<td>2.68, 2.86</td>
<td>4.13, 4.39</td>
<td>1.27, 1.38</td>
</tr>
<tr>
<td>virginica</td>
<td>6.41, 6.77</td>
<td>2.88, 3.07</td>
<td>5.40, 5.71</td>
<td>1.95, 2.10</td>
</tr>
<tr>
<td>All</td>
<td>5.71, 5.98</td>
<td>2.99, 3.13</td>
<td>3.47, 4.04</td>
<td>1.08, 1.32</td>
</tr>
</tbody>
</table>

2.5.7 Factor(), RowFactor() and Multicolumn()

The Factor() function converts its argument into a factor, but keeps the original name for a column heading. RowFactor() is designed to be used only for \LaTeX output: it produces multiple rows the way a factor does, but with more flexibility in the formatting. The Multicolumn() function is also designed for \LaTeX output: it displays factor levels in the style where the level is displayed across multiple columns on its own line.

They have syntax

Factor(x, name, levelnames, texify=TRUE)
RowFactor(x, name, levelnames, spacing=3, space=1, nopagebreak="\nopagebreak", texify=TRUE)
Multicolumn(x, name, levelnames, width=2, first=1, justify="l", texify=TRUE)

The arguments are
x  A variable to be treated as a factor.

name  The name to be used for the factor; by default, the name passed as x.

levelnames  An optional argument to allow customization of the displayed level names.

texify  Whether to escape \LaTeX{} special characters. See section \ref{sec:escape}.

spacing  Extra spacing is added before every group of spacing lines.

space  How much extra space to add (in “ex” units).

nopagebreak  Macro to insert to suppress page breaks except between groups.

width  How many columns for the label?

first  What is the first column?

justify  What justification to use.

Example: Show the first 15 lines of the iris dataset, in groups of 5 lines.

```r
> subset <- 1:15
> latex( tabular( RowFactor(subset, "$i$", spacing=5) ~ + All(iris[subset,], factor=as.character)*Heading()*identity ) )
```
To add extra space after each high level group in a multi-way classification, use `spacing = 1`. For example:

```r
> set.seed(1000)
> dat <- expand.grid(Block=1:3, Treatment=LETTERS[1:2],
+                     Subset=letters[1:2])
> dat$Response <- rnorm(12)
> latex( tabular( RowFactor(Block, spacing=1)
+                   * RowFactor(Treatment, spacing=1, space=0.5)
+                   * Factor(Subset)
+                   ~ Response*Heading()*identity, data=dat),
+       options=list(rowlabeljustification="c"))
```

<table>
<thead>
<tr>
<th>i</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>6</td>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>7</td>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>8</td>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>9</td>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>10</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>11</td>
<td>5.4</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>12</td>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>13</td>
<td>4.8</td>
<td>3.0</td>
<td>1.4</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>14</td>
<td>4.3</td>
<td>3.0</td>
<td>1.1</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>15</td>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>
Block | Treatment | Subset | Response  
--- | --- | --- | ---  
1 | A | a | −0.44578  
| | | b | −0.47587  
| | B | a | 0.63939  
| | | b | −1.37312  
2 | A | a | −1.20586  
| | | b | 0.71975  
| | B | a | −0.78655  
| | | b | −0.98243  
3 | A | a | 0.04113  
| | | b | −0.01851  
| | B | a | −0.38549  
| | | b | −0.55449  

For longer tables, the "longtable" environment allows the table to cross page boundaries. Using this is more complicated, as in the example below. The \texttt{toprule} setting inserts the caption as well as the top rule, because the \texttt{longtable} package requires it to be \texttt{within} the table. The \texttt{midrule} setting gets the headings to repeat on subsequent pages. \footnote{T\'e done all of this in a way that is compatible with the \texttt{booktabs} style; if you want the default style, use \texttt{\hline} in place of the \texttt{booktabs} \texttt{\toprule} and \texttt{\midrule} macros in the \texttt{options} settings instead.} To avoid extra spacing at the top of those pages, we need to undo the automatic addition of a \texttt{\normalbaselineskip} there, and use \texttt{suppressfirst=FALSE} so that the first page doesn’t get messed up. Whew!

```r
> subset <- 1:50
> latex( tabular( RowFactor(subset, "$i$", spacing=5,
+ suppressfirst=FALSE) ~
+ All(iris[subset,], factor=as.character)*Heading()*identity ),
+ options = list(tabular="longtable",
+ toprule="\caption{This table crosses page boundaries.}\\
+ \toprule",
+ midrule="\midrule\\[-2\normalbaselineskip]\endhead\hline\endfoot") )
```
Table 1: This table crosses page boundaries.

<table>
<thead>
<tr>
<th>i</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>6</td>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>7</td>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>8</td>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>9</td>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>10</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>11</td>
<td>5.4</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>12</td>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>13</td>
<td>4.8</td>
<td>3.0</td>
<td>1.4</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>14</td>
<td>4.3</td>
<td>3.0</td>
<td>1.1</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>15</td>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>16</td>
<td>5.7</td>
<td>4.4</td>
<td>1.5</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>17</td>
<td>5.4</td>
<td>3.9</td>
<td>1.3</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>18</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>19</td>
<td>5.7</td>
<td>3.8</td>
<td>1.7</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>20</td>
<td>5.1</td>
<td>3.8</td>
<td>1.5</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>21</td>
<td>5.4</td>
<td>3.4</td>
<td>1.7</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>22</td>
<td>5.1</td>
<td>3.7</td>
<td>1.5</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>23</td>
<td>4.6</td>
<td>3.6</td>
<td>1.0</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>24</td>
<td>5.1</td>
<td>3.3</td>
<td>1.7</td>
<td>0.5</td>
<td>setosa</td>
</tr>
<tr>
<td>25</td>
<td>4.8</td>
<td>3.4</td>
<td>1.9</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>
Table 1: This table crosses page boundaries.

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>5.0</td>
<td>3.0</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>27</td>
<td>5.0</td>
<td>3.4</td>
<td>1.6</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>28</td>
<td>5.2</td>
<td>3.5</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>29</td>
<td>5.2</td>
<td>3.4</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>30</td>
<td>4.7</td>
<td>3.2</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>31</td>
<td>4.8</td>
<td>3.1</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>32</td>
<td>5.4</td>
<td>3.4</td>
<td>1.5</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>33</td>
<td>5.2</td>
<td>4.1</td>
<td>1.5</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>34</td>
<td>5.5</td>
<td>4.2</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>35</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>36</td>
<td>5.0</td>
<td>3.2</td>
<td>1.2</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>37</td>
<td>5.5</td>
<td>3.5</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>38</td>
<td>4.9</td>
<td>3.6</td>
<td>1.4</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>39</td>
<td>4.4</td>
<td>3.0</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>40</td>
<td>5.1</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>41</td>
<td>5.0</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>42</td>
<td>4.5</td>
<td>2.3</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>43</td>
<td>4.4</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>44</td>
<td>5.0</td>
<td>3.5</td>
<td>1.6</td>
<td>0.6</td>
<td>setosa</td>
</tr>
<tr>
<td>45</td>
<td>5.1</td>
<td>3.8</td>
<td>1.9</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>46</td>
<td>4.8</td>
<td>3.0</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>47</td>
<td>5.1</td>
<td>3.8</td>
<td>1.6</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>48</td>
<td>4.6</td>
<td>3.2</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>49</td>
<td>5.3</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>50</td>
<td>5.0</td>
<td>3.3</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

To suppress the row numbering, use `suppress=3` in the call to `tabular`. (It is 3 because we need to suppress the column heading, the rewritten labels for the rows, and the original labels. Trial and error is the best way to determine this!) Unfortunately, the spacing features of `RowFactor()` won’t
work without the row labels.

```r
> subset <- 1:10
> latex( tabular( Factor(subset) ~
+     All(iris[subset,], factor=as.character)*Heading()*identity,
+     suppress=3 ) )
```

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>setosa</td>
</tr>
</tbody>
</table>

(It is actually possible to get this to work with `RowFactor()`, but it is ugly: set the name and level names to "", and set the justification to "l@{}" to suppress the intercolumn spacing. Then the column of row labels will be there, but it will be zero width and invisible.)

`RowFactor` with `spacing > 1` will add the `nopagebreak` macro at the beginning of each label except the first in the group. This can produce \LaTeX errors in any column except the first one. One workaround for this is to post-process the table to move the macro. For example, if `tab` contains the result of `tabular()` and \LaTeX complains about misplaced `\nopagebreak` macros, this will allow it to be displayed properly:

```r
> code <- capture.output( latex( tab ) )
> code <- sub("\-(.*)\(\\\\nopagebreak \)", "\\\1", code)
> cat(code, sep = \"\n\")
```

To get group labels to span multiple columns, the `levelnames` argument can be used with embedded \LaTeX code. For example,

```r
> latex( tabular( Multicolumn(Species, width=3,
+     levelnames=paste("\textit{Iris", levels(Species),"}"))
+     * (mean + sd) ~ All(iris), data=iris, suppress=1))
```
### Iris setosa
\[
\begin{array}{cccc}
\text{mean} & 5.0060 & 3.4280 & 1.4620 \\
\text{sd} & 0.3525 & 0.3791 & 0.1737 \\
\end{array}
\]

### Iris versicolor
\[
\begin{array}{cccc}
\text{mean} & 5.9360 & 2.7700 & 4.2600 \\
\text{sd} & 0.5162 & 0.3138 & 0.4699 \\
\end{array}
\]

### Iris virginica
\[
\begin{array}{cccc}
\text{mean} & 6.5880 & 2.9740 & 5.5520 \\
\text{sd} & 0.6359 & 0.3225 & 0.5519 \\
\end{array}
\]

## 3 Further Details

### 3.1 Formatting

As mentioned in Section 2.4.1, formatting in tables depends on the standard `format()` function or other user-selected functions. Here are the details of how it is done.

The `format.tabular()` method does the first part of the work. First, it constructs the calls to the appropriate formatting functions, and uses them to format all of the non-character entries in the table. The character entries are left as-is, except as described below. This converts the `tabular` object to a character array.

The procedure goes as follows:

1. Entries in the table without specified formatting are formatted first, separately by column using the `format()` function. This is so that entries in a given column will end up with the same character width and (with the default settings) with the same number of decimal places.

2. Entries in the table with specified formatting are grouped according to the format specification. For example, if two columns both share the same `Format()`, they will be formatted in a single call. This results in such entries ending up with the same character width and (with the default settings) with the same number of decimal places.

3. If the `latex` argument is `TRUE`, any numeric entries are passed to the `latexNumeric()` function (see Section 2.1.6), which replaces blanks and mi-
nus signs with fixed width spaces and \TeX minus signs so that all entries will display in the same width. This means that numeric values will normally have decimal points aligned, unless the formatting function explicitly removes leading spaces. Non-numeric entries are passed through the \texttt{Hmisc::latexTranslate} function so that special characters are displayed properly.

4. If the \texttt{latex} argument is \texttt{FALSE}, an attempt is made to justify the results using simple ASCII spacing, according to the \texttt{Justify()} specification with the \texttt{justification} argument used as a default.

Note that \TeX special characters will be escaped in data when \texttt{latex()} is called, but row and column headings generated by \texttt{All()}, \texttt{Factor()}, etc. will by default have the escapes done in all cases. Those functions have a \texttt{texify} argument that can be set to \texttt{FALSE} to disable this behaviour (e.g. if the label is meant to be processed by \TeX). For example, with the definition

```r
> df <- data.frame(A = factor(c( "\$", "\" " ) ), B_label=1:2)
```

the code

```r
> latex( tabular( mean ~ A*B_label, data=df ) )
```

would fail, as the labels would include the special characters. But this will work:

```r
> latex( tabular( mean ~ Factor(A)*All(df), data=df ) )
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>B_label</td>
<td>B_label</td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

As mentioned above, character values in cells in the table are handled specially. If the default \texttt{format} function (or a custom function named \texttt{format}) is used, then those character values are not formatted, they are just copied into the result. (This is so that a column can have mixed numeric and character values, and the numerics are not converted to character before formatting.) If you want to use \texttt{format} on character values, you will need to use a custom formatting function with a different name.
3.2 Missing Values

By default, most summary statistics in R return NA if any of the input values are NA, but have ways to treat NA differently. For example, the mean() function has the na.rm argument:

```r
> dat <- data.frame( a = c(1, 2, 3, NA), b = 1:4 )
> mean(dat$a)
[1] NA

> mean(dat$a, na.rm=TRUE)
[1] 2
```

The tabular() function itself has no way to specify special NA handling, but there are several ways to do this yourself, depending on how you want them handled. To ignore NA values within the column, define a new function which sets the different behaviour. For example,

```r
> Mean <- function(x) base::mean(x, na.rm=TRUE)
> latex( tabular( Mean ~ a + b, data=dat ) )

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2</td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
</tbody>
</table>
```

An alternative approach is to use na.omit() to work on a subset of your data which has rows with any missing values removed, e.g.

```r
> latex( tabular( mean ~ a + b, data = na.omit(dat) ) )

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

A third possibility is to use the complete.cases() function to remove missings only from some columns, e.g.

```r
> latex( tabular( Mean ~ (1 + Heading(Complete)*complete.cases(dat)) * (a + b),
+     data=dat ) )

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2</td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
</tbody>
</table>
```

39
Missing values in factors are normally ignored, i.e. observations whose value is missing won’t match any category. If you would like NA to be used as an additional category, use `exclude = NULL` in a call to `factor()` when you create the variable, e.g. compare the following two tables:

```r
> A <- factor(dat$a)
> latex( tabular( A + 1 ~ (n=1)) )

<table>
<thead>
<tr>
<th>A</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>4</td>
</tr>
</tbody>
</table>

> A <- factor(dat$a, exclude = NULL)
> latex( tabular( A + 1 ~ (n=1) ) )

<table>
<thead>
<tr>
<th>A</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>4</td>
</tr>
</tbody>
</table>
```

### 3.3 Subsetting and Joining Tables

It is possible to select a subset of a table using the usual R matrix indexing on the table object. For example, this table contains rows with no data in them, and those yield ugly NA and NaN statistics:

```r
> set.seed(1206)
> q <- data.frame(p = rep(c("A","B"),each=10,len=30),
+                 a = rep(c(1,2,3),each=10),id=seq(30),
+                 b = rep(c(1,2,3),each=10)
```

```r
> latex( tabular( q + 1 ~ (n=1) ) )

<table>
<thead>
<tr>
<th>q</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>4</td>
</tr>
</tbody>
</table>
```
b = round(runif(30,10,20)),
c = round(runif(30,40,70)),
stringsAsFactors = FALSE)

> tab <- tabular((Factor(p)*Factor(a)+1) ~ (N = 1) + (b + c)*(mean+sd),data=q)
> latex(tab)

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>N</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>All</td>
<td>30</td>
<td>14.43</td>
</tr>
</tbody>
</table>

To omit those rows, use matrix-like subsetting to select the rows where the first column of data (i.e. \( N \)) is greater than zero:

> latex(tab[ tab[,1] > 0, ])

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>All</td>
<td>30</td>
<td>14.43</td>
</tr>
</tbody>
</table>

Similarly, `cbind()` can be used to join tables that have identical row labels, and `rbind()` can be used to join tables with identical column labels. Thus the top part of the table above could be produced in another way:

> formula <- Factor(p)*Factor(a) ~
+ (N = 1) + (b + c)*(mean+sd)
> tab <- NULL
> for (sub in c("A", "B"))
+ tab <- rbind(tab, tabular( formula,  
+               data = subset(q, p == sub) ) )  
> latex(tab)

\begin{tabular}{lcccc}
  & b & & c \\
  a & N & mean & sd & mean & sd \\
  \hline
  A & 1 & 10 & 14.4 & 3.026 & 55.7 & 6.447 \\
  & 3 & 10 & 14.5 & 2.877 & 52.8 & 8.954 \\
  B & 2 & 10 & 14.4 & 2.836 & 56.3 & 7.889 \\
\end{tabular}

It is also possible to edit the row or column labels after constructing the table. For example,

> colLabels(tab)

\[,1\] [,2\] [,3\] [,4\] [,5\]
[1,] b <NA> c <NA>
[2,] N mean sd mean sd
Attributes: dim, justification, colnamejust, justify, suppress, nearData, class

> labs <- colLabels(tab)
> labs[1, 2] <- "New label"
> colLabels(tab) <- labs

> latex(tab)

\begin{tabular}{lcccc}
  & New label & & c \\
  a & N & mean & sd & mean & sd \\
  \hline
  A & 1 & 10 & 14.4 & 3.026 & 55.7 & 6.447 \\
  & 3 & 10 & 14.5 & 2.877 & 52.8 & 8.954 \\
  B & 2 & 10 & 14.4 & 2.836 & 56.3 & 7.889 \\
\end{tabular}

Note that <NA> in the column labels means “same as the label to the left”, and in the row labels it means “same as the label above”. This is used in constructing multi-column or multi-row labels.
3.4 **knitr, rmarkdown and kableExtra support**

This vignette was originally written many years ago using `Sweave`, and is still in that format. Nowadays I would recommend most users to use `knitr` instead: it is easier and more flexible. The input may be in Noweb syntax very similar to this file, or Markdown syntax using the `rmarkdown` package.

One specific advantage of using `knitr` or `rmarkdown` is that explicit calls to `latex()` are not needed: by default, tabular objects will print in the appropriate formatting for LaTeX or HTML output.

The `kableExtra` package may be used to customize displays. For example, the code below causes the table to be full width, and the colour of the 4th column is changed. These features require additional LaTeX packages; see the `kableExtra` documentation for details.

```r
library(magrittr)
library(kableExtra)
toKable(tab) %>%
  kable_styling(full_width = TRUE) %>%
  column_spec(4, color = "red")
```

<table>
<thead>
<tr>
<th></th>
<th>New label</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>N</td>
<td>mean</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>14.5</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>10</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td></td>
<td>mean</td>
</tr>
<tr>
<td>A</td>
<td>55.7</td>
<td>6.447</td>
<td></td>
</tr>
<tr>
<td></td>
<td>52.8</td>
<td>8.954</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>56.3</td>
<td>7.889</td>
<td></td>
</tr>
</tbody>
</table>

See the HTML vignette (which is written in rmarkdown) for more discussion and examples.

**Acknowledgments**

I gratefully acknowledge helpful suggestions and hints from Rich Heiberger, Frank Harrell, Dieter Menne, Marius Hofert, Jeff Newmiller and Jeffrey Miller. Hao Zhu was extremely helpful in adding the `kableExtra` support.
References


Frank E. Harrell, Jr., Richard M. Heiberger, and David R. Whiting. *latex: Convert an S Object to \LaTeX{}, and Related Utilities*, 2011. URL \url{http://CRAN.R-project.org/package=Hmisc}. Help page in Hmisc 3.9-0.