The **tablesgg** Package

Richard Raubertas

27 May 2021

Contents

1 Introduction **2**
1.1 Package features . . . 3
1.2 Some limitations . . . 4
1.3 Logical structure of a data summary table . . . 4
1.4 Remainder of this vignette . . . 4

2 Getting started **5**
2.1 `textTable`s and `pltdTable`s . . . 5
2.2 Table annotation . . . 6
2.3 Table size and scaling . . . 6
2.4 Positioning the table on the graphics device . . . 7
2.5 Grouping rows: `rowheadInside` and `rowgroupSize` . . . 7
2.6 Mathematical notation in table entries . . . 8
2.7 Markdown/HTML in table entries . . . 9
2.8 Reference marks . . . 9
2.9 Setting minimum and maximum widths for table entries . . . 10
2.10 Use with table objects from other packages . . . 10
2.11 Next steps . . . 10

3 Terminology and concepts **10**
3.1 Table parts . . . 10
3.2 The augmented row-column grid; table cells . . . 11
3.3 The hierarchical structure of headers . . . 11
3.4 Table elements: entries, blocks, hvrules . . . 12
3.4.1 Entries . . . 13
3.4.2 Blocks of cells . . . 13
3.4.3 Horizontal and vertical rules (hvrules) . . . 14
3.5 Styles . . . 14
3.6 Confusing conventions . . . 15

4 Customizing `textTables` and `pltdTables` **16**
4.1 Modifying a `textTable` . . . 16
4.2 Modifying a `pltdTable` . . . 17
4.2.1 Changing style and scale . . . 17
4.2.2 Viewing the elements of a plotted table . . . 17
4.2.3 Fine-tuning graphical properties of table elements: `props` functions . . . 18
4.2.4 Adding blocks or hvrules . . . 21
4.2.5 Adding reference marks . . . 22
4.2.6 Modifications at the `ggplot2` level . . . 22
4.3 Setting default styles: `tablesggSetOpt` . . . 23

5 More about styles **23**
5.1 Style specification and matching: Entry and block styles 23
5.2 Style specification and matching: hvrule styles 24
5.3 Editing or creating styles 25

Appendix A: `textTable` objects **25**

Appendix B: Blocks associated with row and column headers **26**
B.1 Header blocks with subtypes A, B, and C 26
B.2 Row header blocks when `plot` argument `rowheadInside` is `TRUE` 28
B.3 Blocks representing groups of rows (`rowgroupSize > 0`) 28

Appendix C: Setting minimum and maximum widths for table entries **29**
`minwidth` 29
`maxwidth` and automatic text wrapping 29

Appendix D: Tables as graphs **30**

References **30**

1 Introduction

The `tablesgg` package displays presentation-quality tables as plots on an R graphics device. There are many packages that will format tables for display. (See the *design-principles* vignette of the `huxtable` package [Hugh-Jones, 2020] for a list and comparisons). `tablesgg` is, to my knowledge, unique in combining two features:

- It is aware of the logical structure of the table being presented, and makes use of that for automatic layout and styling of the table. This avoids the need for most manual adjustments to individual rows, columns, or cells to achieve an attractive result.
- It displays tables using `ggplot2` graphics [Wickham, 2016], on any of R's graphics devices. Therefore a table can be presented anywhere a graph could be, with no more effort. External software such as LaTeX or HTML or their viewers is not required.

`tablesgg` does not *create* tables from raw data, it *displays* tables created by other means. It has methods to display matrices; data frames; contingency tables created by R's built-in `table` and `xtabs` functions; tables created by R's `ftable` function; and tables created by the packages `tables` [Murdoch, 2020] and `xtable` [Dahl, et al, 2019]. Methods can be added to display other table-like objects as well.

Two quick examples illustrate these points. First, a simple listing of a data frame. The package provides the data set `iris2`, which is the same as R's built-in `iris` data frame but with the four measurements per flower reshaped to long format rather than wide:

```
library(tablesgg)
```

```
str(iris2)
```

```
## 'data.frame':    600 obs. of  5 variables:
##  $ plant      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Species    : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ flower_part: Factor w/ 2 levels "Sepal","Petal": 1 1 1 1 1 1 1 1 1 1 ...
##  $ direction  : Factor w/ 2 levels "Length","Width": 1 1 1 1 1 1 1 1 1 1 ...
##  $ value      : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

The common starting point for table display in this package is a `textTable` object, created by the generic function of the same name. There is a `plot` method for `textTable` objects, so displaying the first few rows of the iris data requires just

```
plot(textTable(head(iris2)))
```

	plant	Species	flower_part	direction	value
1	1	setosa	Sepal	Length	5.1
2	2	setosa	Sepal	Length	4.9
3	3	setosa	Sepal	Length	4.7
4	4	setosa	Sepal	Length	4.6
5	5	setosa	Sepal	Length	5.0
6	6	setosa	Sepal	Length	5.4

(Throughout this vignette, the `ggplot` theme is set to use a transparent background for all plots, as follows:

```
library(ggplot2)
theme_update(plot.background=element_rect(fill=NA))
```

)

To create more sophisticated data summary tables I recommend the `tables` package [Murdoch, 2020]. `tablesgg` defines a `textTable` method for the `tabular` objects created by that package, so displaying them can be done in one line. For example, to display means and standard deviations for each measurement and species in the iris data, first create the table,

```
library(tables)
iris2_tab <- tabular(Species*Heading()*value*Format(digits=2)*(mean + sd) ~
                     Heading("Flower part")*flower_part*Heading()*direction,
                     data=iris2)
```

and then plot it:

```
plot(textTable(iris2_tab))
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

1.1 Package features

- A full set of tools is provided to control the appearance of tables, including titles, footnotes and reference marks, horizontal and vertical rules, and spacing of rows and columns. Many properties can be set automatically by specifying *styles*, such as the default styles used above. The user can also define custom styles.
- There are tools for low-level manipulation of the appearance of individual table elements if desired.
- All sizes and dimensions in displayed tables are specified in physical units (points for font size, millimeters for everything else). Therefore a plotted table has a well-defined physical size, independent of the size of the graphics device on which it is displayed. The user can easily increase or decrease the displayed size by a scale factor, maintaining the relative proportions of table elements.

- Table entries can use markdown/HTML tags to mix different fonts, font faces, colors, and text sizes within a single entry. (Requires the `ggtext` package [Wilke, 2020].)
- Automatic wrapping of entry text to a user-specified width is available. (Requires the `ggtext` [Wilke, 2020] and `quadprog` [Turlach, 2019] packages.)
- Since the plotted tables are ordinary `ggplot` objects, the facilities of `ggplot2` and its various extension packages are available to modify or manipulate the table. For example, the table can be inserted as an image within another plot.
- Any table-like object can be displayed just by writing a `textTable` method for the object class. See Appendix A.

1.2 Some limitations

- Splitting a very long or wide table into multiple, smaller subtables is currently not supported.
- Mathematical symbols and notation in table entries are implemented using R's `plotmath` facility. This is more limited than what is available through LaTeX or HTML. A particular limitation is that `plotmath` ignores line breaks in text strings, so math notation cannot be used in multi-line entries. Also `plotmath` and markdown/HTML cannot mixed within the same entry.

1.3 Logical structure of a data summary table

The conceptual model for tables used by this package is similar to that used in the `tables` package [Murdoch, 2020]. The table of summary statistics for the iris data, shown above, will be used as an example.

The rows of a table are defined by combinations of one or more discrete variables (here, species and type of summary statistic—mean or standard deviation). The columns are defined by combinations of one or more other discrete variables (here, flower part and measurement direction). The table body contains the values associated with each combination of row and column variables (e.g., each unique combination of species, flower part, direction, and summary statistic), formatted as character strings.

The idea that a table consists of row variables, column variables, and a text string associated with each combination of values of those variables is quite general. For example, consider the first table above, a simple listing of the first few observations in a data frame. The row variable of this table is just the row or observation number. The column variable is more subtle: it is a categorical variable that takes values in the set `c("plant", "Species", "flower_part", "direction", "value")`, corresponding to the column *names* of the data frame. And the table body entries are the formatted values associated with each combination of row number and column name. Note that table entries are always treated as character strings, so the fact that the original values in the data frame had different types (numeric or factor) does not matter.

When there is more than one row variable, or more than one column variable, they are treated as nested from outermost to innermost. So in the second table above, summary statistic types are nested within species, and measurement directions are nested within flower part. Nesting implies a hierarchical or tree structure for the table rows and columns, and this structure is used in styling the table. Note for example the extra space inserted between levels of the outermost row and column variables (between the different species and between the two flower parts), and how horizontal rules (lines) are used to group the columns associated with each flower part. This can be done automatically by the `tablesgg` package because it is aware of this hierarchical structure.

1.4 Remainder of this vignette

Section 2 describes how to get started with `tablesgg` and illustrates some of the main features. Section 3 discusses the model and terminology for tables used by `tablesgg` in more detail. This material is important for users who wish to customize the appearance of their tables. Customization of table display for individual tables is discussed in section 4, and section 5 describes how to define custom styles that can be applied to any table.

2 Getting started

2.1 textTables and pltdTables

The starting point for all displays generated by this package is a `textTable` object. These objects are created by the generic function of the same name. The package includes methods to create `textTable`s from a variety of table-like objects. For example the `data.frame` method creates a `textTable` that represents a simple listing of a data frame. The `tabular` method creates a `textTable` from the data summary tables produced by the `tables` package. To see all the methods currently available, enter

```
methods(textTable)
```

```
## [1] textTable.data.frame  textTable.default*    textTable.ftable
## [4] textTable.matrix      textTable.table      textTable.tabular
## [7] textTable.tblEntries* textTable.xtable     textTable.xtableList
## see '?methods' for accessing help and source code
```

Appendix A describes how to write methods for other types of objects.

As the name suggests, all parts of a table in a `textTable` object—the table body, row and column headers, and any annotation such as titles or footnotes—are text strings. That is, the process of converting any object to a `textTable` includes formatting numbers or other non-text into the character strings that are to be displayed in the final table.

The package defines a `plot` method for `textTable`s. Plotting a `textTable` creates a `pltdTable` object, which is also a `ggplot`. As with any other graph created by `ggplot2`, printing the object causes it to be displayed on the currently active graphics device.

To illustrate, let's return to the tables shown in the Introduction. The listing of the first few rows of the `iris2` data frame was produced by

```
plot(textTable(head(iris2)))
```

The `row.names` argument to the `data.frame` method of `textTable` controls whether row names are displayed (`FALSE` suppresses them), and if so, what label is used for the column containing them:

```
plot(textTable(head(iris2), row.names="Obs. #"))
```

Obs. #	plant	Species	flower_part	direction	value
1	1	setosa	Sepal	Length	5.1
2	2	setosa	Sepal	Length	4.9
3	3	setosa	Sepal	Length	4.7
4	4	setosa	Sepal	Length	4.6
5	5	setosa	Sepal	Length	5.0
6	6	setosa	Sepal	Length	5.4

The second table in the Introduction was created using the `tables` package:

```
iris2_tab <- tabular(Species*Heading()*value*Format(digits=2)*(mean + sd) ~
                     Heading("Flower part")*flower_part*Heading()*direction,
                     data=iris2)
```

`iris2_tab` is an object of class `tabular`, which is converted to a `textTable` and displayed as follows.

```
plot(textTable(iris2_tab))
```

(In fact, `tablesgg` includes a `plot` method for `tabular` objects, which does the conversion to `textTable` automatically. So just `plot(iris2_tab)` would also work.)

We now consider some options and enhancements to these basic tables. See the help pages for the functions mentioned for more details and additional capabilities.

2.2 Table annotation

Annotation can be added to a table in the form of title, subtitle, and foot lines, via the `title`, `subtitle`, and `foot` arguments to `textTable`. Each of these is a character vector, with each element in a vector generating a new entry that spans the full width of the table. Title lines appear at the top, followed by subtitle lines. Footlines appear at the bottom of the table. For example,

```
ttbl <- textTable(iris2_tab, title="The iris data",
                  subtitle=c("Summary statistics by species",
                             "A second subtitle line"),
                  foot="sd = standard deviation")
```

```
plot(ttbl)
```

The iris data
Summary statistics by species
A second subtitle line

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

The same arguments can be used in the call to `plot` to add or replace existing annotation in a `textTable`.

```
# Change the main title, remove the subtitles.
plot(ttbl, title="A new title", subtitle=character(0))
```

A new title

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

2.3 Table size and scaling

Plotted tables (`pltdTable` objects) have a well-defined physical size, which can be extracted using the `pltdSize` function. By default the size is reported in millimeters, but inches or centimeters can be requested using the `units` argument. The first value is the width and the second is the height.

```
plt <- plot(iris2_tab, title="The iris data")
pltdSize(plt)
```

```
## [1] 77.04089 52.58743
## attr(,"device")
## [1] "pdf"
## attr(,"units")
## [1] "mm"
```

If desired, this can be used to open a graphics device or `grid` viewport of exactly the right size to hold the table. For example

```
sz <- pltdSize(plt, units="in")   # R expects device dimensions in inches
dev.new(width=sz[1], height=sz[2])
plt
```

The size is determined by the fonts used for table entries, the amount of space allocated for horizontal and vertical rules, and other graphical parameters. These are set by *styles*, which are discussed in sections 3 and 5 below. The result is called the *natural size* of the table. However the physical size can be modified by two arguments to `plot`. `scale` is a multiplier that increases or decreases the size of all table elements proportionally.

```
plt2 <- plot(iris2_tab, scale=0.8, title="The iris data (scale=0.8)")
plt2
```

The iris data (scale=0.8)

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

Argument `plot.margin` is a numeric vector of length 4 that specifies how much extra empty space should be added around the sides of the table, in millimeters (as always). This is equivalent to the theme element of the same name in `ggplot2`, and sides follow same order: top, right, bottom, left. `plot.margin` is added *after* any scaling by `scale`, and is included in the table size reported by `pltdSize`.

Note that if the active graphics device or viewport is smaller than the physical size of the plotted table, then parts of the table will be clipped off and not visible.

2.4 Positioning the table on the graphics device

As mentioned previously, a `pltdTable` object must be "printed" in order for it to be displayed on the currently active graphics device. There is a special `print` method for these objects which ensures that the table is displayed at the correct size. In addition, it allows specifying where on the device surface the table should appear.

The default is that the table is drawn centered in the current graphics viewport (usually the whole graphics device surface). This can be changed using either the `position` or the `just`, `vpx` and `vpy` arguments to `print`. See the documentation for `print.pltdTable` for details.

2.5 Grouping rows: `rowheadInside` and `rowgroupSize`

Two arguments to the `plot` method for `textTable`s allow visual grouping of table rows. Setting `rowheadInside` to `TRUE` moves the outermost row header column inside the table, making the table narrower and longer. Setting `rowgroupSize` to a positive integer causes extra space to be inserted after every `rowgroupSize` rows.

```
plt1 <- plot(iris2_tab, title="The iris data", subtitle="With rowheadInside = TRUE",
             rowheadInside=TRUE)
plt2 <- plot(textTable(iris2[1:9, ]), title="The first 9 rows of 'iris2'",
             subtitle="In groups of 4 (rowgroupSize=4)", rowgroupSize=4)
```

```
print(plt1, position=c("left", "center"))
print(plt2, position=c("right", "center"), newpage=FALSE)
```

The iris data
With rowheadInside = TRUE

	Flower part			
	Sepal		Petal	
	Length	Width	Length	Width
Species: setosa				
mean	5.01	3.43	1.46	0.25
sd	0.35	0.38	0.17	0.11
Species: versicolor				
mean	5.94	2.77	4.26	1.33
sd	0.52	0.31	0.47	0.20
Species: virginica				
mean	6.59	2.97	5.55	2.03
sd	0.64	0.32	0.55	0.27

The first 9 rows of 'iris2'
In groups of 4 (rowgroupSize=4)

	plant	Species	flower_part	direction	value
1	1	setosa	Sepal	Length	5.1
2	2	setosa	Sepal	Length	4.9
3	3	setosa	Sepal	Length	4.7
4	4	setosa	Sepal	Length	4.6
5	5	setosa	Sepal	Length	5.0
6	6	setosa	Sepal	Length	5.4
7	7	setosa	Sepal	Length	4.6
8	8	setosa	Sepal	Length	5.0
9	9	setosa	Sepal	Length	4.4

2.6 Mathematical notation in table entries

Mathematical notation can be included in table entries, including annotation. This is done by setting the entry text in a `textTable` to a string representing a `plotmath` expression (see `?plotmath`), and prefixing it with the characters '`MATH_`'. For example, to include math in the title,

```
ttbl <- textTable(iris2_tab, title=paste0("MATH_plain('The length of vector')~",
                                          "group('(', list(a, b), ')')~plain('is ')~",
                                          "sqrt(a^2 + b^2)"))
```

```
plot(ttbl)
```

The length of vector (a, b) is $\sqrt{a^2 + b^2}$

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

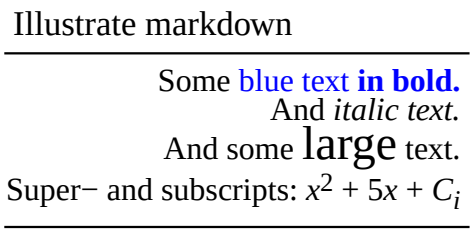
See `?plotmath` for a full description of the available symbols and notation. `plotmath` expressions can be included in a `textTable` when it is created, as above, or can be added or edited afterward using the `props` functions discussed in section 4.2.3. Note that `plotmath` ignores control characters such as newline (`\n`) in expressions.

2.7 Markdown/HTML in table entries

If the `ggtext` package [Wilke, 2020] has been installed, entry text can include markdown/HTML tags. These allow one to mix different fonts, font faces and sizes, and colors within a single entry. To indicate that entry text is to be interpreted as markdown/HTML, prefix it with `MKDN_`. For example,

```
txt1 <- paste0(
          "MKDN_Some <span style='color:blue'>blue text **in bold.**</span><br>",
          "And *italic text.*<br>",
          "And some <span style='font-size:18pt; color:black'>large</span> text.")
txt2 <- "MKDN_Super- and subscripts: *x*<sup>2</sup> + 5*x* + *C*<sub>*i*</sub>"
plt <- plot(textTable(matrix(c(txt1, txt2), ncol=1)),
            title="Illustrate markdown", scale=1.2)
```

```
print(plt)
```



- This feature can be turned on and off using the package option `allowMarkdown` (see `?tablesggSetOpt`). By default the option is set to `TRUE` if `ggtext` is installed and `FALSE` if not.
- Not all HTML tags are supported. See `?ggtext::geom_richtext` for examples of what is available. Note that line breaks are indicated by `\n` in plain text, but by the `<br>` tag in HTML.
- Markdown/HTML and `plotmath` cannot be mixed in the same entry.

2.8 Reference marks

A reference mark is a symbol placed before or after entry text to indicate a cross-reference; e.g. for footnotes. Reference marks can be added to either a `textTable` or `pltdTable` using the `addRefmark` function. The following adds a footnote to explain the abbreviation "sd", and cross-references entries containing the abbreviation to the footnote:

```
ttbl <- textTable(iris2_tab, foot="sd = standard deviation")
ttbl <- addRefmark(ttbl, mark="a", before="sd =", after="sd$", raise=TRUE)
```

```
plot(ttbl)
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd[a]	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd[a]	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd[a]	0.64	0.32	0.55	0.27

[a]sd = standard deviation

Argument `mark` is the character or symbol to be used as the reference mark. `before` and `after` are *regular expressions* (see `?regex`) that identify which table entries are to have the mark placed at their beginning or end, respectively. `raise` indicates whether the reference mark is to be displayed as a superscript (using `plotmath` or markdown).

In the example the `after` regular expression matches all three row headers corresponding to standard deviations, since they all have the same entry text. For finer control, such as to mark only the first appearance, the `props` functions can be used with a `pltdTable` object, as described in section 4.2.3.

2.9 Setting minimum and maximum widths for table entries

The graphical properties `minwidth` and `maxwidth`, set either by a style or with one of the `props` functions, can be used to control the width of individual table entries (and thereby the widths of the columns they span). For example, setting the minimum or maximum width of a table's title will control the width of the whole table, since the title spans all columns. See Appendix C for details.

2.10 Use with table objects from other packages

Packages like `tables` and `xtable` do two things: they create table-like objects (with classes `tabular` and `xtable`, respectively), and they generate LaTeX or HTML code to style those objects for rendering to a PDF viewer or browser. This package provides methods to convert the table-like objects from the first step into `textTable`s. Then the styling and rendering (to a graphics device) are done using the facilities of this package, not those of the original package.

2.11 Next steps

The next section describes the model and terminology that the `tablesgg` package uses for tables. It is important to understand these concepts in order to fine-tune or customize the appearance of your tables.

3 Terminology and concepts

3.1 Table parts

A table has seven parts, illustrated by the shaded regions here.

The iris data
Summary statistics by species
A second subtitle line

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

Part	Part ID
Title	`title`
Subtitle	`subtitle`
Row header	`rowhead`
Row header labels	`rowheadLabels`
Column header	`colhead`
Body	`body`
Foot lines	`foot`

- Parts `title`, `subtitle`, and `foot` are collectively referred to as the table *annotation*. Each is optional. They can each contain multiple entries, which will be displayed on separate lines. They span the full width of the table.

- Parts `rowhead` and `colhead` contain the row headers and column headers of the table, respectively. Visually they simply provide labels for rows and columns of the table body. Conceptually they represent combinations of values of *row and column variables*, which are discussed further below.
- Part `body` contains the body of the table: the values associated with each combination of row and column variables, formatted as text strings.
- Part `rowheadLabels` is optional and contains labels for the row header variables; that is, labels for each column of `rowhead`.

`body`, `rowhead`, and `colhead` can be thought of as matrices that fit together into a larger matrix, with `body` in the lower right quadrant, `rowhead` in the lower left quadrant, and `colhead` in the upper right quadrant. The upper left quadrant is called the *stub* of the table; part `rowheadLabels` occupies the bottom row of the stub.

The number of rows and columns in each table part can be obtained with the `summary` function, applied to either a `textTable` or `pltdTable`:

```
summary(ttbl)
```

```
## 'textTable' with augmented row-column grid dimensions: (13, 6)
## Table parts:
##                nr nc
## title           1 NA
## subtitle        2 NA
## rowhead         6  2
## rowheadLabels   1  2
## colhead         3  4
## body            6  4
## foot            1 NA
```

The number of columns for annotation parts is reported as `NA` because they don't have a fixed number of columns; they span however many columns are required by the other parts. Not every part needs to be present in a given table, and `nr` and/or `nc` will be 0 for empty parts.

3.2 The augmented row-column grid; table cells

When we speak of the rows and columns of a table, we are typically referring to the rows and columns of the table *body*. However it is clear from the figure in section 3.1 that the arrangement of the other table parts can be thought of as adding additional rows and columns, creating an *augmented row-column grid* for the table:

- Each row header variable (column of `rowhead`) adds a column.
- Each column header variable (row of `colhead`) adds a row.
- Each line of annotation (in `title`, `subtitle`, or `foot`) adds a row.

The `adim` function reports the number of rows and columns in this augmented grid:

```
adim(ttbl)
```

```
## [1] 13  6
```

In the `tablesgg` package all references to positions within a plotted table are with respect to the augmented row-column grid. Each location in the grid is called a *cell* of the table.

3.3 The hierarchical structure of headers

As mentioned in the introduction, the model for a data summary table used by this package is one in which:

- There are one or more discrete variables in a data set. These are arbitrarily partitioned into row variables and column variables. (In the iris data table above, the column variables are flower part (Sepal or Petal) and measurment type (Length or Width). The row variables are Species and summary statistic type (mean or sd).)
- Each combination of values of row variables corresponds to one row of the table body, and each combination of values of column variables corresponds to one column of the body.
- For each combination of values of the row and column variables, a single text string is generated, representing the value associated with that combination. These text strings form the entries in the body of the table.

The combinations of values of the row variables are displayed in the table as the *row header*, and the combinations of values of the column variables as the *column header*. Each column of `rowhead` corresponds to one row variable and is called a *layer* of the header. Layers are numbered from innermost (closest to the table body) to outermost. Thus in the iris table, statistic type is layer 1 of the row header and species is layer 2. Similarly each row of `colhead` corresponds to one column variable, with a layer number that increases from innermost (measurement type) to outermost (flower part).

Within a header, the variables are treated as *nested*, inner (lower numbered) layers within outer layers. (This is independent of whether the variables would be considered nested or crossed for statistical modeling purposes.) Thus in the iris data table, measurement type is considered nested within flower part, and statistic type is considered nested within species.

Nesting implies a hierarchical or tree structure for the rows of `rowhead` and the columns of `colhead`. Layer number indicates how close to the bottom of the hierarchy a row or column variable is. The *level* then numbers the nodes within a layer. In the example, species, at layer 2, has three nodes or levels: 1 (setosa), 2 (versicolor), and 3 (virginica). Statistic type, at layer 1, has six nodes or levels: 1 (mean), 2 (sd), 3 (mean), 4 (sd), etc. Note that because of the nesting structure, "mean" for species versicolor is not assumed to have any relation to "mean" for setosa; for the purposes of displaying the table, they are entirely different levels of statistic type and have different node/level numbers.

This structure can be used to help style the appearance of the table. For example, by default additional space is inserted between different levels of the row or column header hierarchy at layers 2 or higher (e.g. between different species in the example). Similarly, horizontal rules are used to group columns at different levels of the column header hierarchy in layers 2 or higher (e.g., spanning the two measurement types for each flower part). The way this styling is specified is discussed in section 5.

3.4 Table elements: entries, blocks, hvrules

As implemented in this package, plotted tables have three types of *elements*, namely *entries*, *blocks*, and *hvrules*. Elements are the smallest pieces of a table whose display can be individually controlled. In a sense they are the "atoms" of the table display. Each type of element is described in more detail below, but elements of all types share the following characteristics:

- Each element has an ID, a character string that is unique within the element type. To see the ID's of all the elements of a given type in a table, use the `ids` function.
- Each element has a set of *graphical properties* that specify how it is to be displayed. For example, the font, color, and border for a table entry; or the line type and thickness for an hvrule. `?elements` documents the available graphical properties for each element type.
- Each element has a special property called `enabled`. This is either `TRUE` or `FALSE`, and controls *whether* the element is displayed at all.
- Elements have additional descriptors related to their role in the logical structure of the table, such as the table part they are associated with, and their position within that part. These additional descriptors vary depending on the element type, and are described in `?elements`.

3.4.1 Entries

Recall that a cell is a single position within the augmented row-column grid of the table. A table *entry* is the text string (and associated properties) assigned to a cell, or to a rectangular set of contiguous cells. In the latter case we say the entry *spans* multiple cells. For example, in the table in section 3.1 above

- There are two subtitle entries. The first spans all the cells in row 2 of the augmented grid (columns 1-6), and the second all the cells in row 3.
- The column header entry "Petal" spans two columns in the second row of `colhead`, corresponding to columns 5-6 in row 5 of the augmented grid.
- The body entry "6.59" occupies a single cell, at row 11, column 3 of the augmented grid.

The key points are that (a) the term *entry* includes text appearing in any part of the table, not just the body; and (b) entries can span multiple cells.

The standard ID's for entries have the form '*part,row number,column number*' for table parts that are matrices (the body, row and column headers, and row header labels), and '*part,element number*' for table parts that are vectors (table annotation). Note that in entry ID's only, '*row number*' and '*column number*' refer to rows and columns within the table part, not to row and column numbers of the augmented row-column grid. When an entry spans more than one row or column, the smallest row or column number is used. Thus in the above table, the ID of the column header entry "Petal" is the string "`colhead,2,3`".

3.4.2 Blocks of cells

A *block* is simply a rectangular set of contiguous table cells. Any number of blocks may be defined for a given table, and blocks may overlap. A block may be empty, having 0 rows or 0 columns. Blocks serve three purposes:

- They provide a convenient way to refer to a collection of cells, or to the entries occupying those cells. The display properties of the whole collection can be set in a single operation.
- A block can be assigned certain graphical properties of its own, independent of the entries it contains. This can be used to highlight a region of the table by adding background shading or a border. For example, in the figure in section 3.1, different background colors were used to highlight blocks corresponding to the seven table parts.
- Blocks provide the framework for adding horizontal and vertical rules to a table. This is discussed in the next subsection.

By default a standard set of blocks is defined for all tables. These include:

- `table`: The whole table (all cells).
- `title`, `subtitle`, `colhead`, `rowhead`, `rowheadLabels`, `body`, `foot`: The standard table parts. (If there are interior row header entries, `rowhead` and `body` are omitted because the interleaving of headers and body means neither are valid blocks.)
- `titles`: The union of the `title` and `subtitle` parts.
- `stub`: The cells above the row headers and to the left of the column headers.
- `colhead_and_stub`, `rowhead_and_stub`: The unions of `stub` with `colhead` and `rowhead`, respectively.
- `colhead_and_body`, `rowhead_and_body`: The unions of `body` with `colhead` and `rowhead`, respectively.

Additional blocks are defined to represent the hierarchical structure of row and column headers. They have ID's that begin with strings "`rowblock`" or "`colblock`". See Appendix B for details.

A user can define arbitrary additional blocks for a table using the `addBlock` function; see section 4.2.4 for examples.

3.4.3 Horizontal and vertical rules (hvrules)

In the context of tables, *rules* refer to horizontal or vertical lines that are used to separate or group table parts or sections. The table in section 3.1 includes five such horizontal lines (and no vertical lines). The `tablesgg` package generalizes this idea and uses the term *hvrule* to refer to something more flexible: a thin rectangle that is inserted between rows or columns, *which may or may not* contain a visible line. The effect of an hvrule with no visible line is simply to add extra space between rows or columns. This can be seen in the example: hvrules were used to add space between the subtitles and the column header, between the row header and body, between levels of species, and between the two flower parts.

By default, this package creates hvrules that run along each of the four sides of each standard block. They are given ID's of the form '*block id_side*', where '*side*' is one of "top", "bottom", "left", or "right". However the `enabled` property is set to `FALSE` for most of them so that they are not displayed, and thus add no space to the table. As for any element, the user can enable or disable selected hvrules and/or modify their graphical properties; see section 4. In addition, the user can define arbitrary additional hvrules with the `addHvrule` function; see section 4.2.4 for examples.

An important point to note is that hvrules are entirely distinct from *borders*. Borders are a graphical property of entries or blocks, while hvrules are their own type of element. One should not try to use borders to create table rules, nor use hvrules to create borders around other elements. To illustrate the difference, the following figure shows the same table twice, highlighting entry borders on the left, and shading in hvrule rectangles on the right. As can be seen, the amount of space inserted between rows or columns by an hvrule can vary; `space` is one of the graphical properties of hvrules.

Highlight borders of table entries
Summary statistics by species
A second subtitle line

Species		Flower part: Sepal Length	Sepal Width	Petal Length	Petal Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

Highlight hvrules
Summary statistics by species
A second subtitle line

Species		Flower part: Sepal Length	Sepal Width	Petal Length	Petal Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

Also note that hvrules do not change a table's augmented row-column grid. Instead, horizontal rules are assigned a nominal row number that is the half-integer between the row numbers of the two rows it separates. For example, the horizontal rule running between augmented row numbers 3 and 4 in the figure above (i.e., below the subtitles) has a row number of 3.5. A horizontal rule always spans an integer number of columns. Analogously, a vertical rule has a nominal column number that is a half-integer, and spans an integer number of rows.

3.5 Styles

In processing a `textTable` into a `pltdTable` that is ready for display, graphical properties like font, text justification, color, etc., have to be assigned to each table element. The initial assignment of graphical properties is specified by a set of *styles*, one each for entries, blocks, and hvrules. Thus, as the name suggests,

styles control the visual appearance of elements in the plotted table. Styles are specified via the `entryStyle`, `blockStyle`, and `hvruleStyle` arguments to `plot`.

Styles are implemented as `styleObj` objects. The package includes a few built-in styles that serve as defaults. Users can edit these or add additional styles as they choose. The way styles are defined and applied is described in section 5. As an illustration, the following shows the same table plotted twice. On the left the default entry style is used. Among other things it uses a serif font, a larger font size for the title, and sets horizontal and vertical justification of text according to each entry's structural role in the table. On the right a "base" style is used, which just assigns the same generic graphical parameters to all entries.

```
plt1 <- plot(ttbl, title="Default style for entries")
plt2 <- plot(ttbl, entryStyle=styles_pkg$entryStyle_pkg_base,
             title="The 'base' style for entries")
```

```
print(plt1, position=c("left", "center"))
print(plt2, position=c("right", "center"), newpage=FALSE)
```

Default style for entries
Summary statistics by species
A second subtitle line

Species		Flower part			
		Sepal		Petal	
		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

The 'base' style for entries
Summary statistics by species
A second subtitle line

Species		Flower part			
		Sepal		Petal	
		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

3.6 Confusing conventions

A potential source of confusion in displaying tables as plots is the differing conventions about coordinate systems: origin, axis directions, and axis order.

- Tables follow the matrix convention in which the origin is at the upper left, with row numbers increasing from top to bottom and column numbers increasing from left to right. Dimensions and coordinates are in (row, column) order, that is, vertical coordinate first, then horizontal.
- The plot convention is to have the origin at the lower left, with the vertical coordinate increasing from bottom to top, and the horizontal from left to right. Dimensions and coordinates are in (horizontal, vertical) order.

In this package the matrix convention is followed in almost all cases: for the augmented row-column grid, the dimensions of tables and their parts, horizontal and vertical justification of entry text within cells, and descriptors for table elements. The plot convention is used only for the following two aspects of `pltdTable` objects:

- The `pltdSize` function returns physical dimensions in (horizontal, vertical) order.
- The `position`, `just`, `vpx`, and `vpy` arguments of the `print` method also expect values in (horizontal, vertical) order, with 0 meaning left/bottom and 1 meaning right/top.

The other difference in conventions to be aware of is that `tablesgg` uses millimeters for dimensions, whereas R graphics functions use inches. This should only matter to the user when opening a graphics device based

on `pltdSize`; specify `units="in"` to get table size in inches.

4 Customizing `textTables` and `pltdTables`

Facilities are available to modify existing `textTable` and `pltdTable` objects, without re-creating them from the original source objects. One can also change the default styles used to assign graphical properties to table elements.

4.1 Modifying a `textTable`

There is an `update` method for `textTable` objects. It allows one to change or remove the table's annotation (titles, subtitles, foot lines) and labels for the row header columns (`rowheadLabels`).

A `textTable` can also be subscripted in the usual matrix way, to create a new `textTable` with fewer (or rearranged) rows or columns. The subscripts are applied to the augmented row-column grid. For example, the following will remove the first column header row ("Flower part") from the example table, and reverse the order of the "Sepal" and "Petal" sets of columns:

```
subttbl <- ttbl[-4, c(1,2,5,6,3,4)]
# Also change annotation:
subttbl <- update(subttbl, title="Example of subscripting a 'textTable'")
```

```
plot(subttbl)
```

Example of subscripting a 'textTable'
Summary statistics by species
A second subtitle line

		Petal		Sepal	
Species		Length	Width	Length	Width
setosa	mean	1.46	0.25	5.01	3.43
	sd	0.17	0.11	0.35	0.38
versicolor	mean	4.26	1.33	5.94	2.77
	sd	0.47	0.20	0.52	0.31
virginica	mean	5.55	2.03	6.59	2.97
	sd	0.55	0.27	0.64	0.32

sd = standard deviation

The above subscripting required us to count rows and columns of the augmented grid, and would have to be modified if, for example, we changed the number of title or subtitle lines in the table. The helper functions `arow` and `acol` allow the subscripts to be specified in a less fragile way:

```
i <- arow(ttbl, "colhead")[1]  # row number of first column header row
j1 <- acol(ttbl, "rowhead")  # column numbers for row header
j2 <- acol(ttbl, "colhead")  # column numbers for column header
subttbl2 <- ttbl[-i, c(j1, j2[c(3,4,1,2)])]
subttbl2 <- update(subttbl2, title="Example of subscripting a 'textTable'")
identical(subttbl, subttbl2)
```

```
## [1] TRUE
```

Subscripting cannot be used to move rows or columns between different table parts (e.g., between headers and the table body).

4.2 Modifying a `pltdTable`

It is possible to change the graphical properties of table elements in an existing plotted table, as well as shrink or expand its overall displayed size. One can also add additional blocks or hvrules to the table. However it is not possible to make changes that alter the table's augmented row-column grid, such as adding new entries or annotation. For that one must go back to the starting `textTable`, or to the object from which the `textTable` was generated.

4.2.1 Changing style and scale

There is an `update` method for `pltdTable` objects. It allows one to change the styles used to assign graphical properties to table entries, blocks, and hvrules. It also accepts the `scale` argument, a multiplier to change the displayed size of the table by shrinking or expanding all elements proportionally. Note that scaling is not cumulative; it is always relative to the natural size of the table as determined by its styles. Thus in the following

```
plt1 <- plot(ttbl)
plt2 <- update(plt1, scale=0.8)
plt3 <- update(plt2, scale=1.0)
rbind(pltdSize(plt1), pltdSize(plt2), pltdSize(plt3))
```

```
##          [,1]     [,2]
## [1,] 77.04089 62.69298
## [2,] 61.83271 50.26097
## [3,] 77.04089 62.69298
```

the third plot is the same size as the first, not the second.

The update method also accepts the `plot.margin` argument to change the amount of padding space added around the four sides of the plot.

4.2.2 Viewing the elements of a plotted table

The `elements` function extracts and returns the elements of a plotted table, as data frames. Each data frame has one row per element, and columns that include the element ID, descriptors of the role, position, and characteristics of the element in the table, and the graphical properties assigned to it. See the function documentation for a description of each of these columns.

The `elements` function has an argument `enabledOnly`, with a default of `TRUE`, to extract only the *enabled* elements of the appropriate type.

The following shows the first few entry elements of the table from the previous subsection:

```
head(elements(plt1, type="entry"))
```

```
##                    id     part subpart partrow partcol headlayer level_in_layer
## title,1       title,1    title    <NA>       1      NA         6              1
## subtitle,1 subtitle,1 subtitle    <NA>       1      NA         4              1
## subtitle,2 subtitle,2 subtitle    <NA>       2      NA         5              1
## foot,1         foot,1     foot    <NA>       1      NA         1              1
## body,1,1     body,1,1     body    <NA>       1       1         0              1
## body,2,1     body,2,1     body    <NA>       2       1         0              2
##                                    text      type arow1 arow2 acol1 acol2
## title,1                     The iris data character     1     1     1     6
## subtitle,1 Summary statistics by species character     2     2     1     6
## subtitle,2        A second subtitle line character     3     3     1     6
## foot,1           sd = standard deviation character    13    13     1     6
## body,1,1                            5.01   numeric     7     7     3     3
```

```
## body,2,1                            0.35   numeric     8     8     3     3
##            enabled textspec multicolumn multirow hjust vjust color alpha size
## title,1       TRUE    plain        TRUE    FALSE     0     0 black     1   11
## subtitle,1    TRUE    plain        TRUE    FALSE     0     0 black     1    9
## subtitle,2    TRUE    plain        TRUE    FALSE     0     0 black     1    9
## foot,1        TRUE    plain        TRUE    FALSE     0     0 black     1    9
## body,1,1      TRUE    plain       FALSE    FALSE     1     1 black     1   10
## body,2,1      TRUE    plain       FALSE    FALSE     1     1 black     1   10
##            family fontface lineheight angle hpad vpad fill fill_alpha
## title,1     serif        1        0.9     0    1  0.7 <NA>          1
## subtitle,1  serif        1        0.9     0    1  0.7 <NA>          1
## subtitle,2  serif        1        0.9     0    1  0.7 <NA>          1
## foot,1      serif        1        0.9     0    1  0.5 <NA>          1
## body,1,1    serif        1        0.9     0    1  0.7 <NA>          1
## body,2,1    serif        1        0.9     0    1  0.7 <NA>          1
##            border_size border_color minwidth maxwidth
## title,1            0.5         <NA>     -0.4       NA
## subtitle,1         0.5         <NA>     -0.4       NA
## subtitle,2         0.5         <NA>     -0.4       NA
## foot,1             0.5         <NA>     -0.4       NA
## body,1,1           0.5         <NA>     -1.0      Inf
## body,2,1           0.5         <NA>     -1.0      Inf
```

4.2.3 Fine-tuning graphical properties of table elements: `props` functions

To make changes to the overall appearance of a table in a way that is readily applied to other tables, it is simplest to edit or create a new style object for the corresponding table elements. See section 5. However for one-off changes, or for fine control of individual table elements, there are more direct tools: the `props<-`, `propsa<-`, and `propsd<-` functions.

These are "setter" or replacement functions, designed to appear on the left-hand side of an assignment. Their first argument is the `pltdTable` to be modified. Additional arguments identify the specific elements to be changed. The right-hand side of the assignment is an object that both indicates the type of elements being changed (entries, blocks, or hvrules) and lists the new properties to be given to those elements. For example,

```
plt <- plot(ttbl)
props(plt, id="body") <- element_entry(fontface=3, fill="gray85")
props(plt, id="subtitle,2") <- element_entry(text="Properties changed",
                                             fill="gray85")
props(plt, id="rowhead_right") <- element_hvrule(linetype=1, color="black")
```

```
plt
```

The iris data
Summary statistics by species
Properties changed

Species		Flower part			
		Sepal		Petal	
		Length	Width	Length	Width
setosa	mean	*5.01*	*3.43*	*1.46*	*0.25*
	sd	*0.35*	*0.38*	*0.17*	*0.11*
versicolor	mean	*5.94*	*2.77*	*4.26*	*1.33*
	sd	*0.52*	*0.31*	*0.47*	*0.20*
virginica	mean	*6.59*	*2.97*	*5.55*	*2.03*
	sd	*0.64*	*0.32*	*0.55*	*0.27*

sd = standard deviation

changes the display of all entries in the table body to italics (`fontface=3`), with a gray background; changes the second line of the subtitle; and puts a visible line in the vertical rule separating the row header from the body of the table. Note that one of the available properties for entries is `text`, here used to change the text of the second subtitle line.

The right-hand side of the assignment must be an `element_*` object, where `*` is either `entry`, `block`, `hvrule`, or `refmark`. These objects are modeled on `element_text`, `element_rect`, etc. objects from the `ggplot2` package. They are created by functions of the same names. Arguments to the functions specify values of graphical properties; any property not specified in the `element_*` object is left unchanged. Note particularly that the `enabled` property allows one to turn on and off the display of individual elements. See the documentation of the `elements` function for full lists of the available properties.

On the left-hand side of the assignment, one specifies which table elements are to receive the new properties. The only difference between the three `props` functions is the form of the specification. `props<-` uses element or part ID's. It is convenient for changing properties of table annotation or whole sections of the table, as shown above. (To see the ID's of all the elements in a table, use the `ids` function.)

`props<-` can also select table entries based on their text content: the `regex` argument takes a regular expression, and matching entries are selected. For example, to add reference marks to explain the abbreviation "sd":

```
plt <- plot(textTable(iris2_tab, foot="sd = standard deviation"))
props(plt, regex="^sd$") <- element_refmark(mark="*", side="after")
props(plt, regex="^sd =") <- element_refmark(mark="*", side="before")
```

```
plt
```

Species		Flower part			
		Sepal		Petal	
		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd*	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd*	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd*	0.64	0.32	0.55	0.27

*sd = standard deviation

`propsa<-` selects table elements using explicit row and column numbers within the augmented row-column grid:

```
propsa(plt, arows=c(5, 7, 9), acols=5) <- element_entry(color="red")
```

```
plt
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd*	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd*	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd*	0.64	0.32	0.55	0.27

*sd = standard deviation

The helper functions `arow` and `acol` return row and column numbers associated with table elements or parts. For example, to put all the mean values in bold:

```
propsa(plt, arows=arow(plt, hpath=c(NA, "mean")),
       acols=acol(plt, id="body")) <- element_entry(fontface=2)
```

```
plt
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	**5.01**	**3.43**	**1.46**	**0.25**
	sd*	0.35	0.38	0.17	0.11
versicolor	mean	**5.94**	**2.77**	**4.26**	**1.33**
	sd*	0.52	0.31	0.47	0.20
virginica	mean	**6.59**	**2.97**	**5.55**	**2.03**
	sd*	0.64	0.32	0.55	0.27

*sd = standard deviation

`propsd<-` selects table elements using the values of element descriptors. Internally entries, blocks, and hvrules are represented as data frames, with one row per element, and columns describing their content, position, and structural role in the table. (See `?elements` for the descriptor columns for each type of element.) Argument `subset` of `propsd<-` is an expression involving those columns, that evaluates to a logical vector; the elements for which this vector is `TRUE` will be selected. (`NA` in the logical vector is treated as `FALSE`.) Thus the `subset` argument works in the same way as R's built-in `subset` function to select rows from a data frame. For example

```
plt <- plot(textTable(iris2_tab))
propsd(plt, subset=(enabled)) <- element_hvrule(color="red")
propsd(plt, subset=(part == "colhead" & headlayer == 1)) <-
       element_entry(angle=90, hjust=0.5, vjust=1.0)
```

```
plt
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

As illustrated in the second line above, a useful descriptor on which to base element selection is `enabled`, to change the properties of just the hvrules or blocks that are currently enabled for display.

4.2.4 Adding blocks or hvrules

The `addBlock` and `addHvrule` functions allow one to add arbitrary additional blocks or hvrules to a plotted table. Unlike the blocks and hvrules that are automatically generated and styled when a `textTable` is plotted, addition of elements using these functions is entirely manual: their location and span with respect to the *augmented row-column grid* must be set explicitly, and their graphical properties are unaffected by styles applied to the table.

Location and span are specified by arguments `arows` and `acols`. For a new block, each is a numeric vector. The minimum and maximum values in the vector specify the first and last rows, and first and last columns, contained in the block. Graphical properties are specified by argument `props`, which should be an `element_block` object as described in the previous subsection. As an example, continuing with the previous (modified) table,

```
plt <- addBlock(plt, arows=c(6, 7), acols=c(3, 4),
                props=element_block(border_color="red", border_size=1.0),
                enabled=TRUE)
```

```
plt
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

adds (and makes visible) a block that highlights the cells in rows 6-7 of columns 3 and 4. Leaving `enabled` at its default value of `FALSE` allows one to define a block without making it visible, in which case `props` can be omitted. The `id` argument allows one to set the string used as the block ID, so that the block can be referred to later.

When using `addHvrule` to create a new horizontal rule, `arows` should be a single value: the half-integer bracketed by the table rows between which the rule runs. `acols` should be a numeric vector whose range

specifies the column numbers spanned by the rule. For a vertical rule the roles of `arows` and `acols` are reversed: `arows` is a numeric vector indicating the row numbers spanned by the rule, and `acols` is the half-integer bracketed by the table columns between which it runs. Graphical properties for the rule are specified by setting the `props` argument to an `element_hvrule` object. Thus the following adds a new, dashed vertical rule between columns 4 and 5, spanning just the body of the table:

```
plt <- addHvrule(plt, direction="vrule", acols=4.5, arows=arow(plt, "body"),
                 props=element_hvrule(linetype=2, color="blue"), enabled=TRUE)
```

```
plt
```

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

Again, argument `enabled` controls whether the hvrule is displayed (`TRUE` by default for hvrules), and `id` can be used to assign it an ID string.

4.2.5 Adding reference marks

Reference marks (section 2.8) are modifications of table entries, not table elements in themselves. Nevertheless, for convenience there is an `element_refmark` function that can be used on the right-hand side of `props` function assignments to set reference marks on entries. This was illustrated in section 4.2.3 above.

A second way to add reference marks is the `addRefmark` function. It works for both `textTable`s and `pltdTable`s, and was illustrated in section 2.8.

4.2.6 Modifications at the `ggplot2` level

A `pltdTable` object is also a `ggplot`, and can be used and modified as such. Nevertheless in most cases it is best to do styling and modification using the tools provided by the `tablesgg` package. The main reason is that most `ggplot2` plots do not have a fixed physical size—they adapt to the size of the current graphics device or viewport when they are displayed. `pltdTable` objects do have a specific physical size, and there is a dedicated `print` method to make sure they are displayed that way. Modifying a `pltdTable` with `ggplot2` operations, such as `plt + ggtitle("A title")`, will produce a result that may not display properly using either the `pltdTable` or `ggplot` print methods.

An exception is the `ggplot2` theme element `plot.background`, which sets a background color and optional border around the whole table. For example, all of the displays in this vignette use `plot.background=element_rect(fill=NA)` to make the background transparent. If one wishes to put a border or box around the whole table it is in fact preferable to do it like this:

```
plt + theme(plot.background=element_rect(fill=NA, color="black", size=1))
```

rather than trying to create an outer border using hvrules or entry borders.

4.3 Setting default styles: `tablesggSetOpt`

The default element styles (and default `plot.margin`) can be accessed and changed using functions `tablesggOpt` and `tablesggSetOpt`. See their documentation for the details.

5 More about styles

The role of styles is to automate the assignment of graphical properties to table elements. They make it easy to obtain a consistent and attractive appearance across multiple tables, even if the tables differ in their structure and complexity. This section describes how styles are specified, and how they are then applied to tables.

Styles are implemented as `styleObj` objects, created by the function of the same name. There are three types, corresponding to the three element types (entries, blocks, and hvrules). A plotted table will make use of one style of each type.

A `styleObj` object is a data frame. Each row can be thought of as a *pattern* plus a set of graphical properties. Table elements that are to be styled are compared to the patterns. If the pattern in a style row matches a table element, the graphical properties in that row are assigned to the element. If more than one style row matches an element, the properties from the last matching row override the earlier ones. The matching process is done automatically when a `textTable` is plotted, or when a `pltdTable` is updated with new styles.

Specification of style patterns and how they are matched to elements is similar for table entries and blocks, and is described first. The process for hvrules is more complicated and is described second.

5.1 Style specification and matching: Entry and block styles

First note that table entries and blocks internally are stored in objects that are themselves data frames, with one row per element. (These data frames can be accessed using the `elements` function.) Columns include element descriptors such as the table part associated with the element, its position in the table, whether the element spans multiple rows or columns, and other information. See `?elements` for lists of the standard descriptors.

In styles for table entries and blocks, the pattern part of the `styleObj` object consists of a single column named `condition`. `condition` should contain character strings that can be interpreted as expressions involving element descriptors. Each `condition` expression, when evaluated within an entries or blocks data frame, should produce a logical vector with one value per element. (Vectors of length 1 are recycled to the necessary length.) Elements for which the `condition` expression in a style row evaluates to `TRUE` are considered to match that row of the style, and are assigned the graphical properties in that row.

As an illustration, the following shows the package's default style for table entries:

```
styles_pkg$entryStyle_pkg_1
```

```
##                             condition hjust vjust color alpha size family
## 1                        part == "body"   1.0   1.0 black     1   10  serif
## 2       part == "rowhead" & !multirow   0.0   1.0 black     1   10  serif
## 3        part == "rowhead" & multirow   0.0   0.0 black     1   10  serif
## 4  part == "rowhead" & headlayer == 0   0.5   0.5 black     1   10  serif
## 5    part == "colhead" & !multicolumn   1.0   1.0 black     1   10  serif
## 6     part == "colhead" & multicolumn   0.5   1.0 black     1   10  serif
## 7              part == "rowheadLabels"   0.0   1.0 black     1   10  serif
## 8                      part == "title"   0.0   0.0 black     1   11  serif
## 9                   part == "subtitle"   0.0   0.0 black     1    9  serif
## 10                     part == "foot"   0.0   0.0 black     1    9  serif
##    fontface lineheight angle hpad vpad fill fill_alpha border_size border_color
## 1         1        0.9     0    1  0.7   NA          1         0.5           NA
```

```
## 2         1        0.9     0    1  0.7   NA          1         0.5           NA
## 3         1        0.9     0    1  0.7   NA          1         0.5           NA
## 4         3        0.9     0    1  0.7   NA          1         0.5           NA
## 5         1        0.9     0    1  0.7   NA          1         0.5           NA
## 6         1        0.9     0    1  0.7   NA          1         0.5           NA
## 7         1        0.9     0    1  0.7   NA          1         0.5           NA
## 8         1        0.9     0    1  0.7   NA          1         0.5           NA
## 9         1        0.9     0    1  0.7   NA          1         0.5           NA
## 10        1        0.9     0    1  0.5   NA          1         0.5           NA
##    minwidth maxwidth
## 1      -1.0      Inf
## 2      -1.0      Inf
## 3      -1.0      Inf
## 4      -1.0      Inf
## 5      -1.0      Inf
## 6      -1.0      Inf
## 7      -1.0      Inf
## 8      -0.4       NA
## 9      -0.4       NA
## 10     -0.4       NA
```

The first style row has pattern `part == "body"`, and so the graphical properties in that row will be assigned to every entry in the table body. The second and third rows assign graphical properties to row header entries, with a different vertical justification of text (`vjust`) depending on whether the entry spans more than one row. The fourth row applies when the table is plotted with `rowheadInside=TRUE`: the outermost row header entries are moved inside the table and assigned a header layer number of 0. These entries will be in italics (`fontface` equal to 3). `part`, `multirow`, `headlayer`, and so on are all standard entry descriptors.

The default style for blocks has a single row:

```
styles_pkg$blockStyle_pkg_1[, 1:5]
```

```
##   condition   fill fill_alpha border_size border_color
## 1        NA gray85          1         0.5           NA
```

An NA value (or equivalently an empty string) as a style row's `condition` is treated specially: it matches *any* element. The row's graphical properties will be applied to all elements, unless overridden by a later style row. So by default all blocks are assigned a light gray background (`fill=gray85`) and no border (`border_color=NA`). (However by default all standard blocks also have `enabled=FALSE`, and so this background will not be displayed.)

5.2 Style specification and matching: hvrule styles

The creation and styling of hvrules is closely tied to table blocks: by default, four hvrules are created for each block, one running along each side. (They are initially disabled.) Style specification for hvrules is more complicated than for table blocks because hvrules effectively *separate* blocks. Therefore one may want their appearance to depend on characteristics of the blocks on *both* sides of the hvrule. For example, one might want to insert extra space after a block of columns, but only if it is followed by another block of columns, not if it is the rightmost block in the table.

Similar to entries and blocks, hvrules are represented internally as a data frame with one row per hvrule. Columns include: `block`, the ID of the block that generated the hvrule; `side`, the side of `block` along which the hvrule runs ("top", "right", "bottom", or "left"); and `adjacent_blocks`, a string listing the ID's of all the blocks adjacent to `block` on the same side as the hvrule. That is, the hvrule separates `block` from the blocks in `adjacent_blocks`. Note that `adjacent_blocks` may be empty.

In styles for hvrules, the pattern part of the `styleObj` object consists of three columns: `block_condition`,

side, and adjacent_condition. side is one of "top", "bottom", "left" or "right". block_condition and adjacent_condition are like the condition column for block styles: they should contain character strings that can be interpreted as expressions involving block descriptors. Each expression will be evaluated within the data frame of blocks that generated the hvrules (not the data frame containing the hvrules themselves). It should produce a logical vector with one element per block; if the value is TRUE for a block, the block satisfies that expression. See styles_pkg$hvruleStyle_pkg_1 for examples of such expressions.

An hvrule matches a given style row if (a) its generating block satisfies the style row's block_condition; (b) they have the same value of side; and (c) one or more of the hvrule's adjacent_blocks satisfies the style row's adjacent_condition.

Any of block_condition, side, and adjacent_condition in a style row may also be set to NA (or equivalently, to an empty string). In that case the corresponding criterion (a), (b), or (c) is considered to be satisfied for all hvrules, and so does not limit matches. Note that setting adjacent_condition to NA is the only way to satisfy criterion (c) if an hvrule's adjacent_blocks is empty. In all other cases, an empty adjacent_blocks will never satisfy criterion (c).

5.3 Editing or creating styles

Package users can create new styles by editing an existing one, or creating one from scratch. For the former, see ?styles_pkg for a list of styles provided by the package. For the latter, prepare a data frame or .csv file with the appropriate columns for pattern and for graphical properties, and pass it as the first argument to function styleObj. The graphical property columns that must be present in the data frame are described in ?elements.

Appendix A: textTable objects

In order to plot any table-like object using this package, it is sufficient to create a textTable method for the object's class. Examples of such methods can be seen by running methods(textTable).

The key tasks of a textTable method are to (a) specify the logical structure of the table by defining each of its seven parts (see section 3.1); and (b) formatting the contents of those parts as character strings. The resulting textTable object must be a list with the following components:

- body: Character matrix containing the body of the table.
- rowhead: Character matrix with the same number of rows as the table body, containing row headers for the table. Row headers are displayed as a set of columns to the left of the table body. May be empty (0 columns).
- rowheadLabels: Character matrix with as many columns as 'rowhead' and at most one row, specifying labels for the rowhead columns. May be empty (0 rows).
- colhead: Character matrix with the same number of columns as the table body, containing column headers for the table. Column headers are displayed as a set of rows above the table body. If rowheadLabels are present, colhead must have at least one row, but otherwise it may be empty (0 rows).
- title, subtitle, foot: Character vectors providing annotation for the table. Each may be empty (length 0).
- partdim: Numeric matrix with one row per table part (i.e., the components listed above), and columns:
  - nr, nc: Number of rows, columns in the part (nc equal to NA for annotation parts)).
  - arow1, arow2, acol1, acol2: First and last rows, first and last columns occupied by the part within the table's augmented row-column grid. arow1 and arow2 should be NA if nr is 0, acol1 and acol2 should be NA if nc is 0.
- rowhier, colhier: Lists describing the hierarchical structure of row and column headers, respectively. Each list has one component per header layer (column of rowhead, row of colhead), in order from outermost layer to innermost. In turn, each of these components is a data frame with one row per node in the hierarchy at that layer.

In general a `textTable` method should define only the first seven of these components (those representing table parts). The `partdim`, `rowhier`, and `colhier` components are then generated automatically by making the last line of the method function a call to the default `textTable` method. That is,

```
{
#  ... code to create character vectors/matrices for table parts, then ...
z <- list(title=title, subtitle=subtitle, rowhead=rowhead,
          rowheadLabels=rowheadLabels, colhead=colhead, body=body, foot=foot)
# Invoke 'textTable' on the list to finish up processing and for validity
# checks (uses the default method).
textTable(z)
}
```

Components `body`, `rowhead`, and `colhead` should each have an attribute `type`. For `body` this will be a character matrix with the same dimensions, containing an arbitrary string describing the type of value represented in each cell (e.g., "numeric"), or NA. For `rowhead` and `colhead`, it will be a character vector with length equal to the number of header layers, again containing a string describing the type of values in each layer, or NA. `type` will become one of the descriptors of table entries (see `?elements`). Therefore a style or the `propsd<-` function can use its value to assign graphical properties to entries.

The components representing table parts should each have an attribute `justification`. It should be a character matrix or vector of the same size and shape as the component. Values "l", "c", "r" specify left, centered, and right horizontal justification of text, respectively, for the corresponding table entry. Value NA means that the type of justification is not specified. (It will be assigned by a style when the `textTable` is plotted.)

Both `type` and `justification` attributes will be generated automatically, if not already present, by `textTable.default`. Values will be set to the default, NA.

Components `partdim`, `rowhier`, and `colhier` are automatically re-derived from the other components whenever a `textTable` is updated using `update`.

As an aside, text justification might logically be considered part of table styling and display, rather than part of converting entries to character strings. However the fact that `tabular` and `xtable` objects may include justification information makes it desirable that `textTable` objects provide a way for that information to be retained.

Appendix B: Blocks associated with row and column headers

In addition to the standard blocks mentioned in section 3.4.2, collections of blocks are defined to represent the hierarchical structure of row and column headers. These have types `rowblock` and `colblock`. To describe these blocks, some terminology is needed. For concreteness, the description is in terms of column headers; analogous comments apply to row headers.

When a table is displayed, each *row* of column headers (corresponding to a row of the `colhead` matrix in a `textTable` object), defines one *layer* of the header. Layers are numbered from innermost (closest to the table body) to outermost. Structurally, layers form a hierarchy: header values at a lower numbered (inner) layer are nested within values at higher numbered (outer) layers. This hierarchy implies a tree-structured partitioning of table columns according to values of the header variables. A set of contiguous columns that share the same header value for a layer, and for all layers above it in the hierarchy, belong to a single *level* of that layer. Levels are numbered from 1 to the number of levels in a layer.

B.1 Header blocks with subtypes A, B, and C

For each combination of layer number `i` and level number `j` in a header, three blocks are defined, with subtypes "A", "B", and "C". The following figure illustrates the three subtypes for layer 2, level 1 of a table's column headers (i.e., the columns for the "Sepal" measurements).

Highlight a 'colblock' of subtype 'A'
ID of the highlighted block is 'colblock/A/2/1'

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

Highlight a 'colblock' of subtype 'B'
ID of the highlighted block is 'colblock/B/2/1'

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

Highlight a 'colblock' of subtype 'C'
ID of the highlighted block is 'colblock/C/2/1'

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

sd = standard deviation

A block with subtype “A” consists of just the cells in header layer `i` whose value corresponds to level number `j`. It will thus come from a single row in the column headers of the table. And since all the cells in the block have the same value, those cells will typically be merged into a single entry when displaying the table.

A block with subtype “B” is bigger: it consists of the cells in the subtype “A” block, plus the header cells with *smaller* layer numbers in the same columns. So it extends from layer `i` down through the rest of the header rows. And finally a block with subtype “C” is bigger yet: it consists of the cells in the subtype “B” block plus the cells in the table body in the same columns. That is, it spans the same set of columns as the subtype “A” and “B” blocks but adds rows down through the table body. Block ID’s have the form `colblock/<subtype>/i/j`.

Sets of blocks for the row headers are defined analogously. Each *column* in the row headers of a displayed table (corresponding to a column in the `rowhead` matrix of a `textTable` object) represents one layer. The layer closest to the table body is numbered 1 and layer number increases toward the left edge of the table. A subtype “A” block consists of the cells in layer number `i` whose value corresponds to level number `j` in that layer. It will thus come from a single column in the row headers of the table. A subtype “B” block consists of the cells in the “A” block, plus the header cells with *smaller* layer numbers in the same rows. A subtype “C” block further adds the cells in the table body in the same rows. That is, it spans the same set of rows as “A” and “B” blocks but adds columns across through the table body. Block ID’s have the form `rowblock/<subtype>/i/j`.

B.2 Row header blocks when `plot` argument `rowheadInside` is `TRUE`

Moving the outermost layer of row header entries into the interior of the table, where they separate and label groups of rows, changes the shape of table parts. Specifically, since row headers and body are interleaved, neither forms a valid rectangular block. However their union is a valid block, with ID and type `rowhead_and_body`.

When a row header layer is moved inside, its layer number is set to 0. (Conceptually, since it is interleaved with the table body, it is interior even to header layer 1.) Blocks `rowblock/<subtype>/i/j`, where `i` indicates layer number, are different when `i=0` than for other layers. Subtype "A", `rowblock/A/0/j`, has one row and spans all table columns. It contains the label for the `j`-th level. Subtype "B", `rowblock/B/0/j`, contains all row header entries (if any) nested within level `j`. Subtype "C", `rowblock/C/0/j`, combines `rowblock/B/0/j` with all the body rows associated with level `j`. Thus `rowblock/C/0/j` spans all table columns. Unlike when `i` is greater than 0, neither `rowblock/B/0/j` nor `rowblock/C/0/j` contain `rowblock/A/0/j`.

Highlight a set of layer−0 row header blocks ('rowheadInside' set to TRUE)

	Flower part			
	Sepal		Petal	
	Length	Width	Length	Width
	Species: setosa			
mean	5.01	3.43	1.46	0.25
sd	0.35	0.38	0.17	0.11
	Species: versicolor			
mean	5.94	2.77	4.26	1.33
sd	0.52	0.31	0.47	0.20
	Species: virginica			
mean	6.59	2.97	5.55	2.03
sd	0.64	0.32	0.55	0.27

sd = standard deviation

Block subtype	Block ID
A	`rowblock/A/0/2`
B	`rowblock/B/0/2`
C	`rowblock/C/0/2`

B.3 Blocks representing groups of rows (`rowgroupSize > 0`)

When a table has many rows within a given level of the row header hierarchy, the table may be easier to read if rows are grouped into smaller sets of fixed size (groups of 5, for example), with some extra space inserted between groups. To facilitate this, when the `rowgroupSize` argument to `plot` is positive, blocks are created to represent such groups. The block type is "rowblock" and subtype is "G".

Grouping respects the row header hierarchy: the innermost header layer that has runs of repeated values is identified (layer `i` say), and grouping is done separately within each of its levels. The block representing a row group spans all columns of the table body as well as row header layers out to layer `i-1`. Block ID's have the form `rowblock/G/i/j/k`, where `j` is the level number (within layer `i`) that contains the group, and `k` is the group number within that level. Thus `i`, `j`, and `k` are the values of descriptors `headlayer`, `level_in_layer`, and `group_in_level` for the block.

However if the table has no row headers, or none of the row header layers have runs of repeated values, table rows are simply grouped into sets of size `rowgroupSize`. `headlayer` and `level_in_layer` will be NA for the group blocks, and block ID's will have the form `rowblock/G///k`, where `k` is the group number (and value of `group_in_level`).

Appendix C: Setting minimum and maximum widths for table entries

The graphical properties `minwidth` and `maxwidth`, set either by a style or with one of the `props` functions, can be used to control the width of individual table entries. (Here *width* is with respect to the text itself; i.e., the direction of reading for English text, and therefore measured vertically if the text is rotated by 90 or 270 degrees.)

Constraints may be expressed in two forms. Positive values are interpreted as absolute widths in millimeters, and should include the amount of padding specified by `hpad` (when `angle` is 0 or 180 degrees) or `vpad` (when `angle` is 90 or 270 degrees). Negative values are interpreted as multiples of the natural width of the text itself, *without* including padding. Thus setting `minwidth` for an entry to `-1` will guarantee that the width of the spanned cell(s) will be at least enough to contain the text without wrapping.

For simplicity the remainder of this description will assume unrotated text, so that width contraints on entries affect the widths of columns and the width of the table as a whole. When text is rotated to be vertical, width constraints will instead affect row heights and the height of the table.

`minwidth`

`minwidth` constraints are satisfied by expanding column widths as much as necessary: a column will be at least as wide as the maximum `minwidth` for any entry contained in that column. When an entry spans multiple columns, the additional width is allocated proportionally to each of the spanned columns.

An NA value for `minwidth` means there is no constraint on minimum width for that entry, and is equivalent to 0. The default entry style `styles_pkg$entryStyle_pkg_1` sets `minwidth` to `-1` for all entries in the table body, row and column headers, and row header labels. The default value for table annotation (title, subtitle, and foot lines) is `-0.4`, which means that the table width will always be at least 40% of the natural, unwrapped width of the annotation text. This prevents excessive amounts of text wrapping when `maxwidth` is NA.

`maxwidth` and automatic text wrapping

`maxwidth` constraints are satisfied by *wrapping*: breaking a long line of text into multiple, shorter lines. Wrapping is available thanks to the `geom_textbox` function of the `ggtext` package [Wilke, 2020]. It can be enabled or disabled in this package via the option `tablesggOpt("allowWrap")`.

An `Inf` value for `maxwidth` means there is no constraint on maximum width. (However, in the absence of constraints, the internal algorithm favors widths as close as possible to the natural, unwrapped width of the entry text.) This is the default for entries in the table body, row and column headers, and row header labels.

An NA value for `maxwidth` means the maximum width will be determined passively from the `maxwidth` values of other entries in the same table column(s). (It will never be less than `minwidth` however.) This is the default for table titles and footnotes, where long text should be wrapped to fit widths implied by the other table entries. For example, in the following the width of table columns (and hence the table) is based on the entries in the body and headers, and the title is automatically wrapped to fit:

```
tablesggSetOpt(allowWrap=TRUE)
plt <- plot(iris2_tab, title=paste0("An unnecessarily long title, used to ",
            "illustrate automatic text wrapping"))
```

```
print(plt)
```

An unnecessarily long title, used to illustrate automatic text wrapping

		Flower part			
		Sepal		Petal	
Species		Length	Width	Length	Width
setosa	mean	5.01	3.43	1.46	0.25
	sd	0.35	0.38	0.17	0.11
versicolor	mean	5.94	2.77	4.26	1.33
	sd	0.52	0.31	0.47	0.20
virginica	mean	6.59	2.97	5.55	2.03
	sd	0.64	0.32	0.55	0.27

- Setting `maxwidth` to a finite value greater than `-1` and less than the natural width of an entry's text means the spanned cell(s) will not be wide enough to hold the text without wrapping. Therefore if option `tablesggOpt("allowWrap")` is `FALSE`, a warning will be given and `maxwidth` will be ignored.
- The wrapping algorithm in `ggtext::geom_textbox` only breaks lines at spaces in the text; it does not hyphenate words or break at punctuation characters.
- The general effect of setting `minwidth` to a non-zero value is to reduce or prevent text wrapping, while the general effect of setting `maxwidth` to NA or a finite value is to encourage wrapping. Settings for one entry may affect the width and wrapping of other entries, because column widths for the table as a whole must satisfy the constraints for all their entries.
- Text representing `plotmath` expressions cannot be wrapped, so `maxwidth` should be `Inf` or `<= -1` for such entries.

Appendix D: Tables as graphs

Although it is common to think of tables and graphs as quite different ways of presenting data, a table is in fact a kind of scatterplot. The key idea of a scatterplot is to display observations in a 2-dimensional plane, such that their spatial position reflects the values of two variables associated with the observations, the x-coordinate and y-coordinate. A table does exactly that: it is a planar display of data where the spatial position of an entry reflects values of associated variables. The x-axis is defined by the combinations of values of the column variables, and the y-axis by combinations of values of the row variables. The plotting symbol placed at the appropriate x-y position is simply a text string (a table entry) rather than a more abstract glyph. Viewed in this way, it is natural to display tables in the same way one displays graphs, as plots on a graphics device.

References

Dahl, David B., David Scott, Charles Roosen, Arni Magnusson, and Jonathan Swinton. 2019. *xtable: Export Tables to Latex or Html.* `https://CRAN.R-project.org/package=xtable`.

Hugh-Jones, David. 2020. *huxtable: Easily Create and Style Tables for Latex, Html and Other Formats.* `https://CRAN.R-project.org/package=huxtable`.

Murdoch, Duncan. 2020. *tables: Formula-Driven Table Generation.* `https://CRAN.R-project.org/package=tables`.

Turlach, Berwin A. 2019. *quadprog: Functions to Solve Quadratic Programming Problems.* `https://CRAN.R-project.org/package=quadprog`.

Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wilke, Claus O. 2020. *ggtext: Improved Text Rendering Support for 'ggplot2'*. https://CRAN.R-project.org/package=ggtext.