Vignette for the `rtf` Package

Michael E. Schaffer

June 2012

Contents

1 Introduction

`rtf` is a package for outputting Rich Text Format (RTF) files with high resolution tables and graphics that may be edited with standard word processors. This package is not meant as a substitute for `Sweave`, but as a simpler alternative that produces

reports compatible with Microsoft Word and other popular word processors. This vignette demonstrates some of the functions in a larger context than the help file examples.

2 Usage

First, load the package.

```
> library(rtf)
```

Now, we can use the `RTF` function to initialize an RTF object and return a reference for all subsequent methods. You may notice that the parameters for this method are similar to those used in `base` plots. We can define the RTF page width and height (in inches), the default font size (in points), and the outer page margins (in inches).

```
> output<-"rtf_vignette.doc"   # although this is RTF, we can use the
>                              # .doc extension so it opens in MS Word
> rtf<-RTF(output,width=8.5,height=11,font.size=10,omi=c(1,1,1,1))
> # Other rtf commands here...
>
> done(rtf)                    # writes and closes the file
```

3 Text

3.1 Basic Text

There are three ways to output text to an RTF document: `addHeader`, `addParagraph`, `startParagraph`/`addText`/`endParagraph`. These are very similar, but some differences are outlined below.

First, we can create a new section with a title in bold followed by either a subtitle or the section text in normal text.

```
> addHeader(rtf,title="Section Header",
+               subtitle="This is the subheading or section text.")
```

If a header is not necessary, a self-contained paragraph can be created with the `addParagraph` method.

```
> addParagraph(rtf,"This is a new self-contained paragraph.\n")
```

Alternatively, we may define a paragraph start and end. Text may be inserted between these methods using the `addText` function.

```
> startParagraph(rtf)
> addText(rtf,"This text was added with the addText command.  ")
> addText(rtf,"You can add styled text too.  ",bold=TRUE,italic=TRUE)
> addText(rtf,"You must end the paragraph manually.")
> endParagraph(rtf)
```

3.2 New Lines

There are a couple ways to insert new lines within the text. First, you may use "`\n`" within any string. Alternatively, you may insert a new line using the `addNewLine` function.

```
> addNewLine(rtf)
```

3.3 Unicode and Greek Characters

Uppercase and lowercase Greek characters may be used in any string but must be encoded in a specific manner. The implementation mimics syntax used for encoding Unicode characters in HTML 4.0.

For example, to export uppercase Alpha through Epsilon, we can use the follow code.

```
> addParagraph(rtf,"&Alpha; &Beta; &Gamma; &Delta; &Epsilon;\n\n")
```

For lowercase, we use the following.

```
> addParagraph(rtf,"&alpha; &beta; &gamma; &delta; &epsilon;\n\n")
```

Other Unicode characters are supported through specific HTML equivalents. These are defined at `http://www.w3.org/TR/html4/sgml/entities.html`. Unicode characters may be encoded directly using the Unicode decimal value. For example, to encode three heart shapes, we use the following combination of Unicode and RTF syntax.

```
> addParagraph(rtf,"\\u9829\\3 \\u9829\\3 \\u9829\\3\n\n")
```

3.4 URLs

URLs are not currently supported, but will be in a future version.

3.5 Mathematical Formulae

Unfortunately, support for mathematical formulae is not part of the RTF specification. This package is not recommended for heavy use of formulae and cannot produce output on par with `Sweave`.

3.6 RTF Code

Native RTF may be used directly within any text block buy simply adding a backslash to the RTF commands. For example, the following demonstrates two ways to style bold text using native RTF commands, "\b" and "\b0". The full RTF specificaiton of RTF syntax is available at http://www.microsoft.com/en-us/download/details.aspx?id=10725.

```
> addParagraph(rtf,"Normal, \\b this is bold\\b0, normal.\n")
```

or alternatively, with curly brackets to contain the code,

```
> addParagraph(rtf,"Normal, {\\b\\i bold-italic}, normal.\n")
```

When incorporating native RTF code, it is important to pay attention to spacing in the RTF syntax. Improper spacing can cause MS Word parser errors that are difficult to debug.

4 Formatting

Currently, document formatting with the `rtf` package functions is limited. There are, however, a few useful methods. In addition, all of the native RTF syntax is available for more complex document formatting.

Three functions that allow changes in text placement include: `increaseIndent` to indent text or figures, `decreaseIndent` to move the indentation to the left, and `addPageBreak` that allows changes in page orientation and margins in the middle of a document.

5 Tables

The `addTable` function may be used to add tabular data to an RTF report from a `data.frame` or any data that can be coerced to a `data.frame`. For tables with specific formatting requirements, it is recommended to format the table data before passing to the `addTable` function. This function will guess the best column widths to use, but also allows the user to specify column widths.

For example we can insert a table based on the `iris` data below.

```
> tab<-as.data.frame(head(iris)) # create a data.frame
> colnames(tab)<-gsub("\\."," ",colnames(tab)) # format column names


> addTable(rtf,tab,font.size=9,row.names=FALSE,NA.string="-")
```

We may also want to output a table from the `table` command, such as the following.

	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

```
> tab<-table(iris$Species,floor(iris$Sepal.Length))
> names(dimnames(tab))<-c("Species","Sepal Length")
```

	4	5	6	7
setosa	20	30	0	0
versicolor	1	25	23	1
virginica	1	6	31	12

Here we can specify the columns widths and a string to substitute for `NA` values.

```
> addTable(rtf,tab,font.size=10,row.names=TRUE,NA.string="-",
+               col.widths=c(1,0.5,0.5,0.5,0.5) )
```

6 Figures

Figures and graphics may be inserted into the RTF document as easily as text. One limitation of the RTF specification is that vector-based image formats (e.g. EPS or PDF) are undefined except for Microsoft's proprietary WMF and EMF formats. For this reason the `rtf` package uses raster-based PNG files for inserting figures into RTF documents. The resolution for these images may be set to any value; however, for publication, it is recommended that users independently export figures as PDF.

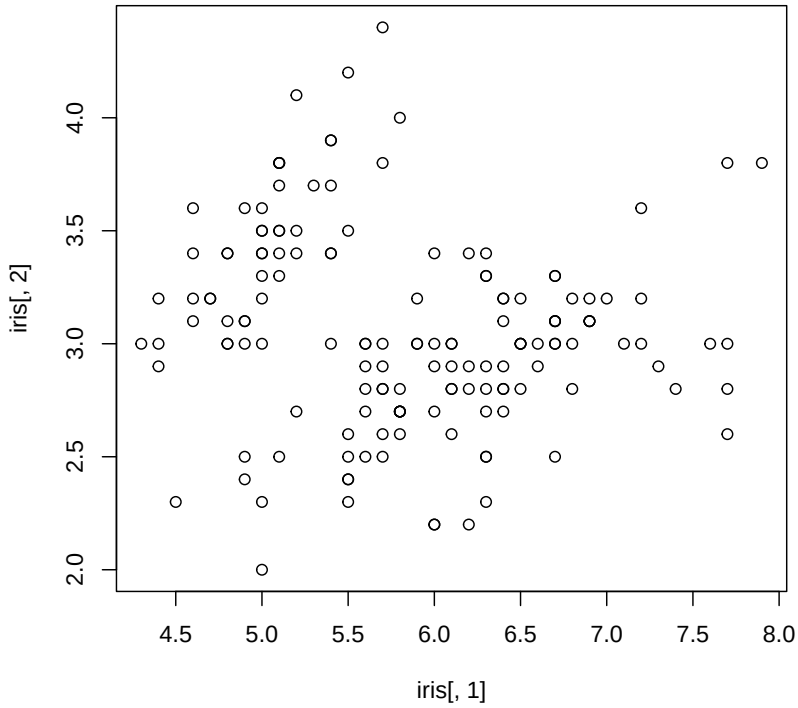
The basic approach for adding plots is the `addPlot` function. This command takes a plot function as a paramater, along with specific RTF parameters, and then any other parameters to be sent to the plot function. The syntax is as follows, where "`...`" represents all parameters to be passed to the `plot.fun` function. Notice how we define the plot width and height (in inches), along with the desired resolution (in dots per inch).

```
> addPlot(RTF.object, plot.fun=plot.fun, width=4, height=5, res=300, ...)
```

6.1 `base` Plots

For `base` graphics plots, we can use the `addPlot` in two ways. First we can directly use the plot function in the `addPlot` function. Below, we simply use `R`'s `plot` function.

```
> plot(iris[,1],iris[,2])
```

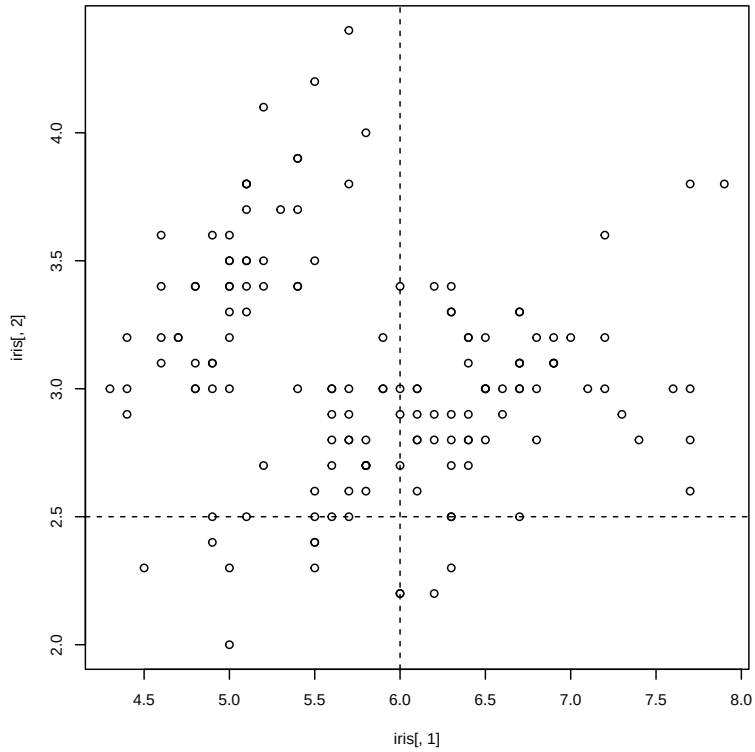


To add this plot to the document, we use the following.

```
> addPlot(rtf,plot.fun=plot,width=6,height=6,res=300, iris[,1],iris[,2])
```

Alternatively, we can wrap a more complex plot that has several steps into a function that takes any number of parameters.

```
> newPlot<-function() {
+         par(pty="s",cex=0.7)      # adjust plot style
+         plot(iris[,1],iris[,2])
+         abline(h=2.5,v=6.0,lty=2) # add some lines
+ }
> newPlot()
```



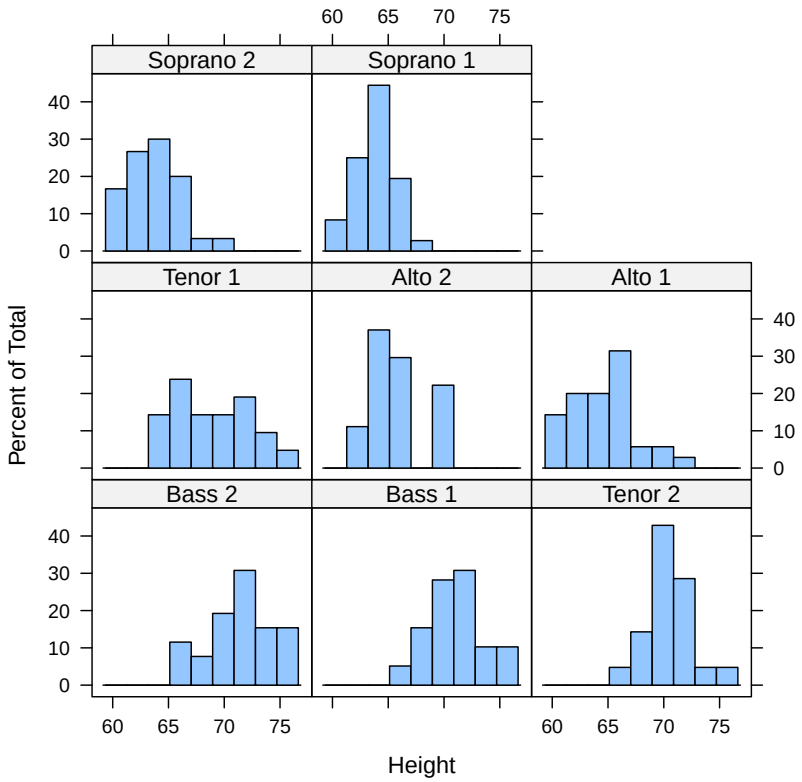
Then we add this plot to the document with the following. Note in this case there are no extra `plot.fun` parameters to pass along, but the function may be rewritten to accept any parameters to promote reuse of plotting code.

```
> addPlot(rtf,plot.fun=newPlot,width=6,height=6,res=300)
```

6.2 `lattice` Plots

The `lattice` package enables trellis plots for visualizing relationships between variables of complex data sets. To create the output for RTF, we just assign the `lattice` plot to a variable. The `print` function may then be used to visualize the plot.

```
> library(lattice)
> p <- histogram( ~ height | voice.part, data = singer, xlab="Height")
> print(p)
```

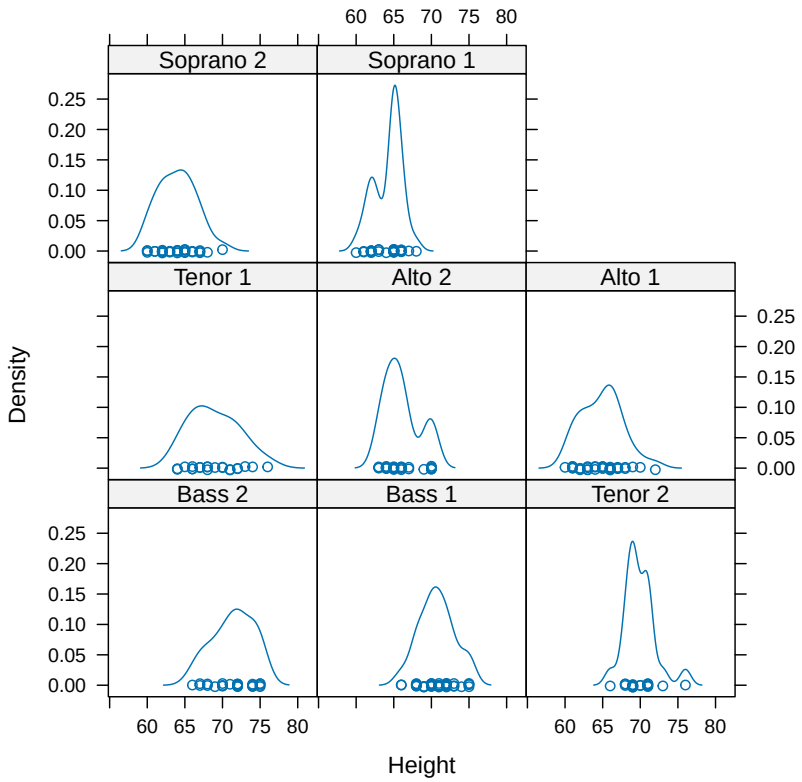


To include this plot in the RTF document we again call the `addPlot` method. However, for `lattice` plots, we pass `print` as the `plot.fun` function and use the `p` variable as a parameter for this function. Below is an example that specifies the plot width and height (in inches), along with the desired resolution (in dots per inch).

```
> addPlot(rtf,plot.fun=print,width=5,height=5,res=300,p)
```

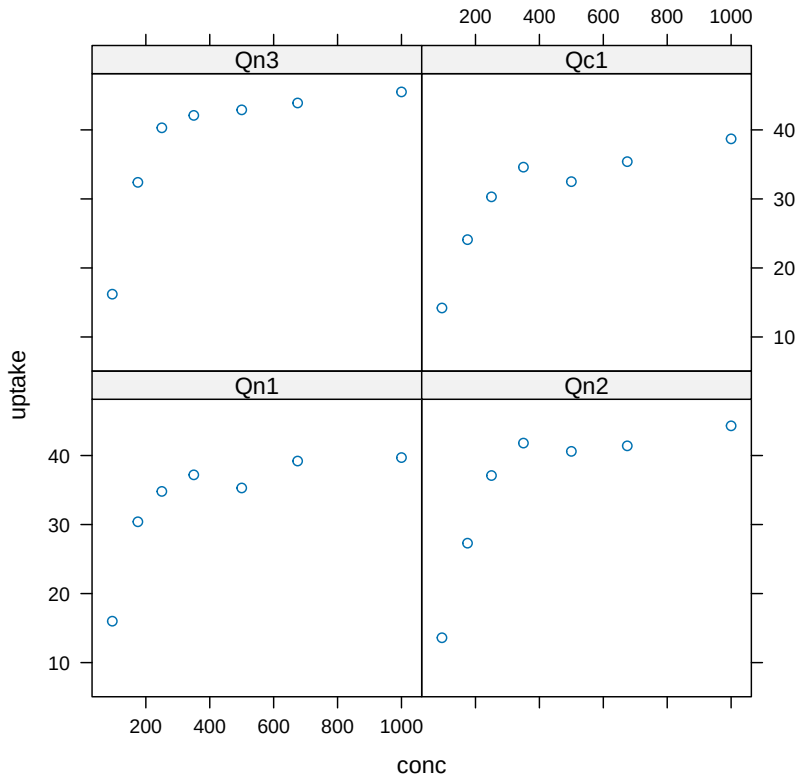
Alternatively, we can use the `addTrellisObject` method with `lattice` plots. The `addTrellisObject` method is suited for multi-page trellis objects and will create an image for each page in the RTF document. It also works for single page images as well.

```
> p2 <- densityplot( ~ height | voice.part, data = singer, xlab = "Height")
> print(p2)
```




```
> addTrellisObject(rtf,trellis.object=p2,width=5,height=5,res=300)

> p3<-xyplot(uptake ~ conc | Plant, CO2, layout = c(2,2))
> print(p3) # note this is a multipage lattice plot
>           # but Sweave only shows the first plot
```

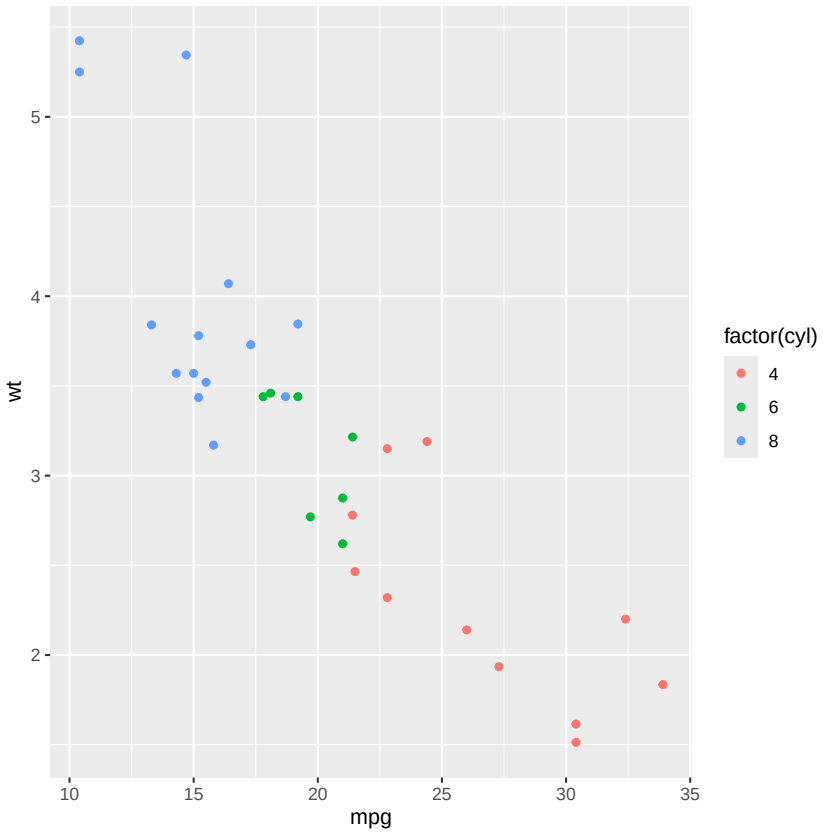



```
> addTrellisObject(rtf,trellis.object=p3,width=6,height=6,res=300)
```

6.3 ggplot2 Plots

The `ggplot2` package is an alternative to `base` and `lattice` graphics and is an implementation of Leland Wilkinson's Grammar of Graphics. Below is code that creates a simple `ggplot2` plot and stores a representation in a variable called, `mt`. The `print` function may then be used to visualize the plot.

```
> # plot
> library(ggplot2)
> mt <- ggplot(mtcars, aes(mpg, wt, colour = factor(cyl))) + geom_point()
> print(mt)
```



To include this plot in the RTF document we again call the `addPlot` method. Just as with `lattice` trellis plots, we pass `print` as the `plot.fun` function and use the `mt` variable as a parameter for this function.

```
> addPlot(rtf,plot.fun=print,width=5,height=4,res=300, mt)
```

6.4 External Figures

The `rtf` package also supports incorporation of existing PNG images generated by `R` or any other software. The code below demonstrates how to do this by pointing to the image file and specifying the desired output size.

```
> addPng(rtf, "foo.png", width=5, height=5)
```

7 Table of Contents

The `rtf` package also supports creation of a Table of Contents. This uses the TOC.level attribute of headers throughout the RTF document. Note that some

word processors, including Microsoft Word, will not automatically render the table of contents unless the field is updated. When opening the file for the first time in Microsoft Word, it may be necessary to force the field to update by right clicking on it and choosing 'Update Field'.

```
> addHeader(rtf,"Table of Contents")
> addTOC(rtf)
> addHeader(rtf,"Section 3",TOC.level=1)
> addHeader(rtf,"Section 3A",TOC.level=2)
> addHeader(rtf,"Section 3B",TOC.level=2)
```

8 Output

When an RTF document is complete, you close and write the document with the `done` command. It may take some time to run depending on the document complexity and number of figures. It is also customary to include information about the `R` session in reports. For this we use the `addSessionInfo` method. For example, the following code creates a page break, adds nicely formatted session information, and writes the RTF document.

```
> addPageBreak(rtf, width=8.5, height=11, omi=c(1,1,1,1))
> addSessionInfo(rtf)
> done(rtf)
```