Get Your Sparkle On

A Guide to Using the Magic of `pixiedust`

Benjamin Nutter

2016-08-18

Contents

Chapter 1

Introduction

Tables are a powerful tool for communicating information. They are pervasive in statistical literature in many forms. We see them in text books as references for standard normal probabilities. Some tables organize analysis methods for differing strata in a population. We may even find tables displaying conference schedules.

For statisticians, the table is an inevitable tool for conveying the results of any variety of statistical tables. In some cases, we begin to rely on a specific style of table we like for a specific kind of analysis, such as one we might use in exploratory data analysis. Other times, we may need to custom fit a table to a rare or novel analysis worthy of special attention.

Creating tables in R is not a new endeavor. Many have attempted it, and many have performed it well. Perhaps most notably, the `xtable` package has streamlined the process of turning table - like objects into presentation - ready formats. Likewise, `knitr`'s `kable` function provides a rapid and reliable method to convert two dimensional grids into aesthetically pleasing tables.

Beyond table - like objects, `xtable` and `stargazer` packages make conversion of non-table-like objects into tabular representations simple and dependable.

In addition to these packages, a multitude of packages have popped up on CRAN to facilitate the creation of tables. These include `tableHTML`, `htmlTable`, `condformat`, and `ReporteRs`. Each of these has strengths and weaknesses - which we will not enumerate here-and each of these are good packages.

So why, then, do we concern ourselves with `pixiedust`, yet *another* R package for creating tables?

In short, it is because `pixiedust` offers a simple, consistent interface for fine tuned customization of each and every cell of the table.

1.1 `pixiedust` Design Constraints

The `pixiedust` package was envisioned with a unique set of constraints. These constraints can be separated into categories involving the data, the user interface, and the output. These constraints guide the development of `pixiedust` and the philosophy of table customization.

1.1.1 Data Constraints

Within any cell of the table

1. The underlying data may be formatted, but not changed.
2. The user must be able to obtain the original object upon request.

This means that, no matter what customizations you make to a cell, the data will never be altered until the table is actually printed. In other words, you can eliminate a vast amount of any pre-processing to the data you may have needed to do in the past. Furthermore, if you inadvertently make a `dust` object from your data, you can restore the data without any loss of information.

Consider the following simple example. With `pixiedust`, we are able to round the first, second, and third rows to two decimal places, and the fourth and fifth rows to four decimal places.

```
library(pixiedust)
options(pixiedust_print_method = "html")
set.seed(1)
Example <- data.frame(sample_id = 1:5,
                      random = rnorm(5)) %>%
  dust() %>%
  sprinkle(rows = 1:3,
           round = 2) %>%
  sprinkle(rows = 4:5,
           round = 4) %>%
  sprinkle_print_method("html")

Example %>% print(asis = FALSE) %>% cat()
```

sample_id

random

1

-0.63

2

0.18

3

-0.84

4

1.5953

5

0.3295

In most other packages, this type of formatting can't be done without some preprocessing. Additionally, even though we've saved `Example` as a `dust` object, the core data is easily retrieved.

```
as.data.frame(Example, sprinkled = FALSE)
```

```
##   sample_id     random
## 1         1 -0.6264538
## 2         2  0.1836433
## 3         3 -0.8356286
## 4         4  1.5952808
## 5         5  0.3295078
```

1.1.2 User Interface Constraints

1. The interface must have as few functions as possible.

2. The user should be able to apply as many customizations to a cell as is desired in one call.
3. The user should be able to designate rows and columns by positional reference and named reference.

`pixiedust` comes with two core functions. The `dust` function converts an object to a tabular structure associated with its formatting choices. The `sprinkle` function acts on the `dust` object to change those associations as directed by the user. `dust` is described in more detail in Section 2.2. Chapter **??** is dedicated to the details of sprinkling.

Additionally, there are some variants on `sprinkle`, namely `sprinkle_colnames` for changing how column names are printed in the table; `sprinkle_table` for applying a sprinkle to multiple parts of a table (header, body, and footer); and `sprinkle_print_method` for changing the print method for a table.

Some less commonly used functions that are included with the package are `pixieply`, a variant of `mapply` that permits `sprinkle` to function over a list of `dust` objects; an `as.data.frame` method for converting `dust` objects to data frames; and a `print` method.

Lastly, there are a handful of utility functions for managing table numbering. These can be seen by calling `?pixie_count`.

The `sprinkle` function has only four formal arguments. The massive flexibility of `pixiedust` on such a simple interface is made possible by a flagrant abuse of the `...` argument (really, I don't think `...` should ever be treated like it is in `pixiedust`). This approach allows the user to call as many defined sprinkles as desired without any action being taken on sprinkles that aren't requested.

1.1.3 Output Constraints

1. The user may choose to output to the console, markdown, HTML, or LaTeX.
2. The interface must be identical for all outputs.
3. When customizations do not apply to an output format, the request will be quietly ignored.
4. When working in Rmarkdown scripts, the output format should be automatically detected, but may be altered by the user.
5. Table numbering should be automatic and handled internally.

`pixiedust` currently supports output to the console, markdown, HTML and LaTeX (PDF), and an engine to output `ReporteRs` formatted tables is in the works. All of these outputs are controlled using the same interface. This has advantages over packages such as `xtable`, where some customizations must be written to a specific output, meaning they can't be transferred from one output format to another.

Not all of the customizations are relevant to every format. For example, backgrounds and borders are not supported by console or markdown output. Font families are supported in HTML, but not in LaTex. When a sprinkle is applied that isn't supported by the format, the request is stored in the `dust` object, but is quietly ignored at the `print` request. The advantage to this arrangement is that there are no errors for applying a sprinkle to an unsupported format. The disadvantage is that you may not always realize that your sprinkle won't show up in your output. Generally, I find that being able to have near-identical tables across multiple formats to outweigh the disadvantage.

Since `pixiedust` is intended to work with Rmarkdown scripts, it is unnecessary to explicitly declare the output format. `pixiedust` will make use of `knitr::opts_knit$get("rmarkdown.pandoc.to")` to determine the output format for you and make the appropriate adjustments.

1.1.4 `pixiedust` Is Not ...

While `pixiedust` is partially motivated by `ggplot` graphics, it does *not* purport to be a grammar of any sort, and does not pretend to have a structured philosophy about how to build tables. It simply does what it is told.

Additionally, `sprinkles` should not be considered layers. Any changes made to a cell will simply overwrite previous changes, and will not allow any interaction.

A last note of caution before you move on to the documentation: be careful not to over-format your tables. It can be tempting, but the content of the table should speak for itself. Customizations to the tables should merely guide the reader's eye to the important information. The customizations you make should never scream at the reader.

With that, I hope you enjoy using `pixiedust` as much as I have. If you have questions, problems, frustrations, or suggestions, please submit an issue to the GitHub repository or send me an e-mail: benjamin.nutter@gmail.com

Chapter 2

`dusting` Objects

2.1 Table-wide options

2.2 Anatomy of the `dust` Object

Chapter 3

Sprinkling

3.1 Table-valued Sprinkles

3.2 Cell-valued Sprinkles

Chapter 4

Colors

Colors may be used for cell borders and backgrounds. `pixiedust` works hard to make it easy to use colors while also giving a great deal of flexibility.

4.1 Named Colors

Designating a color can be a simple as providing a name, so long as you know the valid color names. Fortunately, `pixiedust` recognizes all of the DVIPS color names recognized by R itself. The full list of colors can be seen by running `colors()` at the command line. We show the first 30 colors here:

```
head(colors(), 30)
```

```
##  [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"
##  [5] "antiquewhite2"  "antiquewhite3"  "antiquewhite4"  "aquamarine"
##  [9] "aquamarine1"    "aquamarine2"    "aquamarine3"    "aquamarine4"
## [13] "azure"          "azure1"         "azure2"         "azure3"
## [17] "azure4"         "beige"          "bisque"         "bisque1"
## [21] "bisque2"        "bisque3"        "bisque4"        "black"
## [25] "blanchedalmond" "blue"           "blue1"          "blue2"
## [29] "blue3"          "blue4"
```

A good resource for seeing what these colors looks like is http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

`pixiedust` also recognizes an additional name, `"transparent"`, for when you wish to remove a color that may have already been applied.

In the example below, we'll add a background color to the non-intercept rows in the model summary that are statistically significant.

```
fit <- lm(mpg ~ qsec + factor(gear) + wt, data = mtcars)

dust(fit) %>%
  sprinkle(rows = c(2, 5),
           bg = "lightgreen") %>%
  print(asis = FALSE) %>% cat
```

term	estimate	std.error	statistic	p.value
(Intercept)	16.9629385	7.1503505	2.3723227	0.025061
qsec	0.9531255	0.3481294	2.7378486	0.0108106
factor(gear)4	1.4053374	1.3340693	1.0534216	0.3014842
factor(gear)5	1.4936744	1.8064949	0.8268357	0.4155779
wt	-4.5519623	0.6539171	-6.9610696	2e-07

term	estimate	std.error	statistic	p.value
(Intercept)	16.9629385	7.1503505	2.3723227	0.025061
qsec	0.9531255	0.3481294	2.7378486	0.0108106
factor(gear)4	1.4053374	1.3340693	1.0534216	0.3014842
factor(gear)5	1.4936744	1.8064949	0.8268357	0.4155779
wt	-4.5519623	0.6539171	-6.9610696	2e-07

4.2 RGB Colors

The `colors()` named in R offer you 657 unique colors. But colors may also be specified as an RGB character string of the format `"rgb(RRR, GGG, BBB)"` where `RRR`, `GGG`, and `BBB` are integers between 0 and 255. With 256 values of each of those three spots, that provides you with a total of 16,777,216 distinct colors available for use.

We can replicate the table from the previous example using the `"rgb()"` color specification as follows:

```
fit <- lm(mpg ~ qsec + factor(gear) + wt, data = mtcars)

dust(fit) %>%
  sprinkle(rows = c(2, 5),
           bg = "rgb(144, 238, 144)") %>%
  print(asis = FALSE) %>% cat
```

4.3 HEX Colors

Hexadecimal color codes are another common way to define colors, especially in HTML formats. `RColorBrewer` is a package that returns color codes in hexadecimal format.

To specify a hexidecimal color in `pixiedust`, use a character string in `"#RRGGBB"` format where `RR`, `GG`, and `BB` are hexidecimal values between `00` and `FF`. Our previous example is replicated again using hex codes as

```
fit <- lm(mpg ~ qsec + factor(gear) + wt, data = mtcars)

dust(fit) %>%
  sprinkle(rows = c(2, 5),
           bg = "#90EE90") %>%
  print(asis = FALSE) %>% cat
```

term	estimate	std.error	statistic	p.value
(Intercept)	16.9629385	7.1503505	2.3723227	0.025061
qsec	0.9531255	0.3481294	2.7378486	0.0108106
factor(gear)4	1.4053374	1.3340693	1.0534216	0.3014842
factor(gear)5	1.4936744	1.8064949	0.8268357	0.4155779
wt	-4.5519623	0.6539171	-6.9610696	2e-07

4.4 Transparency

`pixiedust` supports color transparency for HTML tables only. To use transparency, define your colors as `"rgba(RRR, GGG, BBB, AA)"` where `AA` is a value between 0 (fully transparent) and 1 (fully opaque). Hexidecimal colors are specified by `"#RRGGBBAA"` where `AA` is a hexidecimal number from 00 (fully transparent) and FF (fully opaque).

We will do a variation of the previous example where we shade the statistically significant rows in black, and use transparency to ensure the text is still visible (it's a silly thing to do, since we could just use gray, but it's a good way to illustrate the point)

```
fit <- lm(mpg ~ qsec + factor(gear) + wt, data = mtcars)

dust(fit) %>%
  sprinkle(rows = 2,
           bg = "#0000003F") %>%
  sprinkle(rows = 4,
           bg = "rgba(00,00,00,.25)") %>%
  sprinkle_print_method("html") %>%
  print(asis = FALSE) %>% cat
```

term

estimate

std.error

statistic

p.value

(Intercept)

16.9629385

7.1503505

2.3723227

0.025061

qsec

0.9531255

0.3481294

2.7378486

0.0108106

factor(gear)4

1.4053374

1.3340693

1.0534216

0.3014842

factor(gear)5

1.4936744

1.8064949

0.8268357

0.4155779

wt

-4.5519623

0.6539171

-6.9610696

2e-07

Chapter 5

LaTeX Configuration

In order to produce LaTeX output, you will need to ensure the following packages are installed and in use in your document.

- `amssymb`
- `arydshln`
- `caption`
- `graphicx`
- `hhline`
- `longtable`
- `multirow`
- `xcolor` (with the `dvipsnames`, and `table` options.)

If you are using an Rmarkdown document, I recommend including the following text in your YAML front matter. The new command defined in this block is necessary if you wish to use dashed borders. If you don't intend to use dashed borders, it is usually harmless to leave the new command definition in your front matter. There are, however, certain circumstances where this definition cause LaTeX rendering to fail. Unfortunately, I haven't quite narrowed down what the boundaries of those circumstances are. If you aren't using dashed borders and are getting odd messages, you may try removing the last three lines from this block to make it work.

```
header-includes:
- \usepackage{amssymb}
- \usepackage{arydshln}
- \usepackage{caption}
- \usepackage{graphicx}
- \usepackage{hhline}
- \usepackage{longtable}
- \usepackage{multirow}
- \usepackage[dvipsnames,table]{xcolor}
- \makeatletter
- \newcommand*\vdashline{\rotatebox[origin=c]{90}{\$\dabar@\dabar@\dabar@\$}}
- \makeatother
```

If you are using Sweave, or just need the straight LaTeX code without the YAML format, here's a convenient place from which to copy it.

```
\usepackage{amssymb}
\usepackage{arydshln}
\usepackage{caption}
\usepackage{graphicx}
```

```
\usepackage{hhline}
\usepackage{longtable}
\usepackage{multirow}
\usepackage[dvipsnames,table]{xcolor}
\makeatletter
\newcommand*\vdashline{\rotatebox[origin=c]{90}{\$\dabar@\dabar@\dabar@\$}}
\makeatother
```

Chapter 6

Medleys

Chapter 7

Table Numbering and Referencing

Chapter 8

Working with Lists of Tables