Create Awesome HTML Table with knitr::kable and kableExtra

Hao Zhu

2018-05-21

Please see the package documentation site (http://haozhu233.github.io/kableExtra/) for how to use this package in LaTeX.

Overview

The goal of `kableExtra` is to help you build common complex tables and manipulate table styles. It imports the pipe `%>%` symbol from `magrittr` and verbalize all the functions, so basically you can add "layers" to a kable output in a way that is similar with `ggplot2` and `plotly`.

To learn how to generate complex tables in LaTeX, please visit
http://haozhu233.github.io/kableExtra/awesome_table_in_pdf.pdf
(http://haozhu233.github.io/kableExtra/awesome_table_in_pdf.pdf)

There is also a Chinese version of this vignette. You can find it here
(http://haozhu233.github.io/kableExtra/awesome_table_in_html_cn.html)

Installation

```
install.packages("kableExtra")

# For dev version
# install.packages("devtools")
devtools::install_github("haozhu233/kableExtra")
```

Getting Started

Here we are using the first few columns and rows from dataset `mtcars`

```
library(knitr)
library(kableExtra)
dt <- mtcars[1:5, 1:6]
```

When you are using `kable()`, if you don't specify `format`, by default it will generate a markdown table and let pandoc handle the conversion from markdown to HTML/PDF. This is the most favorable approach to render most simple tables as it is format independent. If you switch from HTML to pdf, you basically don't need to change anything in your code. However, markdown doesn't support complex table. For example, if you want to have a double-row header table, markdown just cannot provide you the functionality you need. As a result, when you have such a need, you should **define `format` in `kable()`** as either "html" or "latex". *You can also define a global option at the beginning using `options(knitr.table.format = "html")` so you don't repeat the step everytime.*

Starting from `kableExtra` 0.9.0, when you load this package (`library(kableExtra)`), **it will automatically set up the global option 'knitr.table.format' based on your current environment**. Unless you are rendering a PDF, `kableExtra` will try to render a HTML table for you. **You no longer need to manually set either the global option or the `format` option in each `kable()` function**. I'm still including the explanation above here in this vignette so you can understand what is going on behind the scene. Note that this is only an global option. You can manually set any format in `kable()` whenever you want. I just hope you can enjoy a peace of mind in most of your time.

You can disable this behavior by setting `options(kableExtra.auto_format = FALSE)` before you load `kableExtra`.

```
# If you are using kableExtra < 0.9.0, you are recommended to set a global option f
irst.
# options(knitr.table.format = "html")
## If you don't define format here, you'll need put `format = "html"` in every kabl
e function.
```

Basic HTML table

Basic HTML output of `kable` looks very crude. To the end, it's just a plain HTML table without any love from css.

```
kable(dt)
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Bootstrap theme

When used on a HTML table, `kable_styling()` will automatically apply twitter bootstrap theme to the table. Now it should looks the same as the original pandoc output (the one when you don't specify `format` in `kable()`) but this time, you are controlling it.

```
dt %>%
  kable() %>%
  kable_styling()
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Table Styles

`kable_styling` offers a few other ways to customize the look of a HTML table.

Bootstrap table classes

If you are familiar with twitter bootstrap, you probably have already known its predefined classes, including `striped`, `bordered`, `hover`, `condensed` and `responsive`. If you are not familiar, no worries, you can take a look at their documentation site (http://getbootstrap.com/css/#tables) to get a sense of how they look like. All of these options are available here.

For example, to add striped lines (alternative row colors) to your table and you want to highlight the hovered row, you can simply type:

```
kable(dt) %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

The option `condensed` can also be handy in many cases when you don't want your table to be too large. It has slightly shorter row height.

```
kable(dt) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Tables with option `responsive` looks the same with others on a large screen. However, on a small screen like phone, they are horizontally scrollable. Please resize your window to see the result.

```
kable(dt) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"
))
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620

Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Full width?

By default, a bootstrap table takes 100% of the width. It is supposed to use together with its grid system to scale the table properly. However, when you are writing a rmarkdown document, you probably don't want to write your own css/or grid. For some small tables with only few columns, a page wide table looks awful. To make it easier, you can specify whether you want the table to have `full_width` or not in `kable_styling`. By default, `full_width` is set to be `TRUE` for HTML tables (note that for LaTeX, the default is `FALSE` since I don't want to change the "common" looks unless you specified it.)

```
kable(dt) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Position

Table Position only matters when the table doesn't have `full_width`. You can choose to align the table to `center`, `left` or `right` side of the page

```
kable(dt) %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "left")
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215

Hornet Sportabout	18.7	8	360	175	3.15	3.440

Becides these three common options, you can also wrap text around the table using the `float-left` or `float-right` options.

```
kable(dt) %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "float_ri
ght")
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras sit amet mauris in ex ultricies elementum vel rutrum dolor. Phasellus tempor convallis dui, in hendrerit mauris placerat scelerisque. Maecenas a accumsan enim, a maximus velit. Pellentesque in risus eget est faucibus convallis nec at nulla. Phasellus nec lacinia justo. Morbi fermentum, orci id varius accumsan, nibh neque porttitor ipsum, consectetur luctus risus arcu ac ex. Aenean a luctus augue. Suspendisse et auctor nisl. Suspendisse cursus ultrices quam non vulputate. Phasellus et pharetra neque, vel feugiat erat. Sed feugiat elit at mauris commodo consequat. Sed congue lectus id mattis hendrerit. Mauris turpis nisl, congue eget velit sed, imperdiet convallis magna. Nam accumsan urna risus, non feugiat odio vehicula eget.

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Font size

If one of your tables is huge and you want to use a smaller font size for that specific table, you can use the `font_size` option.

```
kable(dt) %>%
  kable_styling(bootstrap_options = "striped", font_size = 7)
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Column / Row Specification

Column spec

When you have a table with lots of explanatory texts, you may want to specified the column width for different column, since the auto adjust in HTML may not work in its best way while basic LaTeX table is really bad at handling text wrapping. Also, sometimes, you may want to highlight a column (e.g. a "Total" column) by making it bold. In these scenario, you can use `column_spec()`. You can find an example below.

Warning: If you have a super long table, you should be cautious when you use `column_spec` as the xml node modification takes time.

```
text_tbl <- data.frame(
  Items = c("Item 1", "Item 2", "Item 3"),
  Features = c(
    "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vehicula tempor
ex. Morbi malesuada sagittis turpis, at venenatis nisl luctus a. ",
    "In eu urna at magna luctus rhoncus quis in nisl. Fusce in velit varius, posuer
e risus et, cursus augue. Duis eleifend aliquam ante, a aliquet ex tincidunt in. ",
    "Vivamus venenatis egestas eros ut tempus. Vivamus id est nisi. Aliquam molesti
e erat et sollicitudin venenatis. In ac lacus at velit scelerisque mattis. "
  )
)

kable(text_tbl) %>%
  kable_styling(full_width = F) %>%
  column_spec(1, bold = T, border_right = T) %>%
  column_spec(2, width = "30em", background = "yellow")
```

Items	Features
Item 1	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vehicula tempor ex. Morbi malesuada sagittis turpis, at venenatis nisl luctus a.
Item 2	In eu urna at magna luctus rhoncus quis in nisl. Fusce in velit varius, posuere risus et, cursus augue. Duis eleifend aliquam ante, a aliquet ex tincidunt in.
Item 3	Vivamus venenatis egestas eros ut tempus. Vivamus id est nisi. Aliquam molestie erat et sollicitudin venenatis. In ac lacus at velit scelerisque mattis.

Row spec

Similar with `column_spec`, you can define specifications for rows. Currently, you can either bold or italiciz an entire row. Note that, similar with other row-related functions in `kableExtra`, for the position of the target row, you don't need to count in header rows or the group labelling rows.

```
kable(dt) %>%
  kable_styling("striped", full_width = F) %>%
  column_spec(5:7, bold = T) %>%
  row_spec(3:5, bold = T, color = "white", background = "#D7261E")
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	**110**	**3.90**	**2.620**
Mazda RX4 Wag	21.0	6	160	**110**	**3.90**	**2.875**
Datsun 710	**22.8**	**4**	**108**	**93**	**3.85**	**2.320**
Hornet 4 Drive	**21.4**	**6**	**258**	**110**	**3.08**	**3.215**
Hornet Sportabout	**18.7**	**8**	**360**	**175**	**3.15**	**3.440**

Header Rows

One special case of `row_spec` is that you can specify the format of the header row via `row_spec(row = 0, ...)`.

```
kable(dt) %>%
  kable_styling("striped", full_width = F) %>%
  row_spec(0, angle = -45)
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Cell/Text Specification

Function `cell_spec` is introduced in version 0.6.0 of `kableExtra`. Unlike `column_spec` and `row_spec`, **this function is designed to be used before the data.frame gets into the `kable` function**. Comparing with figuring out a list of 2 dimentional index for targeted cells, this design is way easier to learn and use and it fits perfectly well with `dplyr`'s `mutate` and `summarize` functions. With this design, there are two things to be noted: * Since `cell_spec` generates raw `HTML` or `LaTeX` code, make sure you remember to put `escape = FALSE` in `kable`. At the same time, you have to escape special symbols including `%` manually by yourself * `cell_spec` needs a way to know whether

you want `html` or `latex`. You can specify it locally in function or globally via the `options(knitr.table.format = "latex")` method as suggested at the beginning. If you don't provide anything, this function will output as HTML by default.

Currently, `cell_spec` supports features including bold, italic, monospace, text color, background color, align, font size & rotation angle. More features may be added in the future. Please see function documentations as reference.

Conditional logic

It is very easy to use `cell_spec` with conditional logic. Here is an example.

```
library(dplyr)
mtcars[1:10, 1:2] %>%
  mutate(
    car = row.names(.),
    mpg = cell_spec(mpg, color = ifelse(mpg > 20, "red", "blue")),
    cyl = cell_spec(cyl, color = "white", align = "c", angle = 45,
                    background = factor(cyl, c(4, 6, 8),
                                        c("#666666", "#999999", "#BBBBBB")))
  ) %>%
  select(car, mpg, cyl) %>%
  kable(escape = F) %>%
  kable_styling("striped", full_width = F)
```

car	mpg	cyl
Mazda RX4	21	6
Mazda RX4 Wag	21	6
Datsun 710	22.8	4
Hornet 4 Drive	21.4	6
Hornet Sportabout	18.7	8
Valiant	18.1	6
Duster 360	14.3	8
Merc 240D	24.4	4
Merc 230	22.8	4
Merc 280	19.2	6

Visualize data with Viridis Color

This package also comes with a few helper functions, including `spec_color`, `spec_font_size` & `spec_angle`. These functions can rescale continuous variables to certain scales. For example, function `spec_color` would map a continuous variable to any viridis color palettes (https://CRAN.R-project.org/package=viridisLite). It offers a very visually impactful representation in a tabular format.

```
iris[1:10, ] %>%
  mutate_if(is.numeric, function(x) {
    cell_spec(x, bold = T,
              color = spec_color(x, end = 0.9),
              font_size = spec_font_size(x))
  }) %>%
  mutate(Species = cell_spec(
    Species, color = "white", bold = T,
    background = spec_color(1:10, end = 0.9, option = "A", direction = -1)
  )) %>%
  kable(escape = F, align = "c") %>%
  kable_styling(c("striped", "condensed"), full_width = F)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

In the example above, I'm using the `mutate` functions from `dplyr`. You don't have to use it. Base R solutions like `iris$Species <- cell_spec(iris$Species, color = "red")` also works.

Text Specification

If you check the results of `cell_spec`, you will find that this function does nothing more than wrapping the text with appropriate HTML/LaTeX formatting syntax. The result of this function is just a vector of character strings. As a result, when you are writing a `rmarkdown` document or write some text in shiny apps, if you need extra markups other than **bold** or *italic*, you may use this function to color, change font size or rotate your text.

An aliased function `text_spec` is also provided for a more literal writing experience. In HTML, there is no difference between these two functions.

```
sometext <- strsplit(paste0(
  "You can even try to make some crazy things like this paragraph. ",
  "It may seem like a useless feature right now but it's so cool ",
  "and nobody can resist. ;)"
), " ")[[1]]
text_formatted <- paste(
  text_spec(sometext, color = spec_color(1:length(sometext), end = 0.9),
            font_size = spec_font_size(1:length(sometext), begin = 5, end = 20)),
  collapse = " ")

# To display the text, type `r text_formatted` outside of the chunk
```

You can even try to make some crazy things like this paragraph. It may seem like a useless feature right now but it's so cool and nobody can resist. ;)

Tooltip

It's very easy to add a tooltip to text via `cell_spec`. For example, `text_spec("tooltip", color = "red", tooltip = "Hello World")` will give you something like Hover over me (you need to wait for a few seconds before your browser renders it).

Note that the original browser-based tooltip is slow. If you want to have a faster response, you may want to initialize bootstrap's tooltip by putting the following HTML code on the page.

```
<script>
$(document).ready(function(){
    $('[data-toggle="tooltip"]').tooltip();
});
</script>
```

In a rmarkdown document, you can just drop it outside of any R chunks. Unfortunately however, for rmarkdown pages with a **floating TOC** (like this page), you can't use bootstrap tooltips because there is a conflict in namespace between Bootstrap and JQueryUI (tocify.js). As a result, I can't provide a live demo here. If you want to have a tooltip together with a floating TOC, you should use `popover` which has a very similar effect.

Popover Message

The popover message looks very similar with tooltip but it can hold more contents. Unlike tooltip which can minimally work without you manually enable that module, you **have to** enable the `popover` module to get it work. The upper side is that there is no conflict between Bootstrap & JQueryUI this time, you can use it without any concern.

```
<script>
$(document).ready(function(){
    $('[data-toggle="popover"]').popover();
});
</script>
```

```
popover_dt <- data.frame(
  position = c("top", "bottom", "right", "left"),
  stringsAsFactors = FALSE
)
popover_dt$`Hover over these items` <- cell_spec(
  paste("Message on", popover_dt$position), # Cell texts
  popover = spec_popover(
    content = popover_dt$position,
    title = NULL,                           # title will add a Title Panel on top
    position = popover_dt$position
  ))
kable(popover_dt, escape = FALSE) %>%
  kable_styling("striped", full_width = FALSE)
```

position	Hover over these items
top	Message on top
bottom	Message on bottom
right	Message on right
left	Message on left

Links

You can add links to text via `text_spec("Google", link = "https://google.com")` : Google (https://google.com). If you want your hover message to be more obvious, it might be a good idea to put a `#` (go back to the top of the page) or `javascript:void(0)` (literally do nothing) in the `link` option. `text_spec("Hover on me", link = "javascript:void(0)", popover = "Hello")` : Hover on me

Integration with `formattable`

You can combine the good parts from `kableExtra` & `formattable` together into one piece. Read more at http://haozhu233.github.io/kableExtra/use_kableExtra_with_formattable.html (http://haozhu233.github.io/kableExtra/use_kableExtra_with_formattable.html)

```
library(formattable)
mtcars[1:5, 1:4] %>%
  mutate(
    car = row.names(.),
    mpg = color_tile("white", "orange")(mpg),
    cyl = cell_spec(cyl, angle = (1:5)*60,
                    background = "red", color = "white", align = "center"),
    disp = ifelse(disp > 200,
                  cell_spec(disp, color = "red", bold = T),
                  cell_spec(disp, color = "green", italic = T)),
    hp = color_bar("lightgreen")(hp)
  ) %>%
  select(car, everything()) %>%
  kable(escape = F) %>%
  kable_styling("hover", full_width = F) %>%
  column_spec(5, width = "3cm") %>%
  add_header_above(c(" ", "Hello" = 2, "World" = 2))
```

	Hello		World	
car	**mpg**	**cyl**	**disp**	**hp**
Mazda RX4	21.0	ᔕ	*160*	110
Mazda RX4 Wag	21.0	ᔓ	*160*	110
Datsun 710	22.8	ᔭ	*108*	93
Hornet 4 Drive	21.4	ᔓ	**258**	110
Hornet Sportabout	18.7	ᔕ	**360**	175

Grouped Columns / Rows

Add header rows to group columns

Tables with multi-row headers can be very useful to demonstrate grouped data. To do that, you can pipe your kable object into `add_header_above()`. The header variable is supposed to be a named character with the names as new column names and values as column span. For your convenience, if column span equals to 1, you can ignore the `=1` part so the function below can be written as `add_header_above(c(" ","Group 1" = 2, "Group 2" = 2, "Group 3" = 2)).

```
kable(dt) %>%
  kable_styling("striped") %>%
  add_header_above(c(" " = 1, "Group 1" = 2, "Group 2" = 2, "Group 3" = 2))
```

	Group 1		Group 2		Group 3	
	mpg	**cyl**	**disp**	**hp**	**drat**	**wt**

Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

In fact, if you want to add another row of header on top, please feel free to do so.

```
kable(dt) %>%
  kable_styling(c("striped", "bordered")) %>%
  add_header_above(c(" ", "Group 1" = 2, "Group 2" = 2, "Group 3" = 2)) %>%
  add_header_above(c(" ", "Group 4" = 4, "Group 5" = 2)) %>%
  add_header_above(c(" ", "Group 6" = 6))
```

	Group 6					
	Group 4				Group 5	
	Group 1		Group 2		Group 3	
	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Group rows via labeling

Sometimes we want a few rows of the table being grouped together. They might be items under the same topic (e.g., animals in one species) or just different data groups for a categorical variable (e.g., age < 40, age > 40). With the new function `group_rows()` in `kableExtra`, this kind of task can be completed in one line. Please see the example below. Note that when you count for the start/end rows of the group, you don't need to count for the header rows nor other group label rows. You only need to think about the row numbers in the "original R dataframe".

```
kable(mtcars[1:10, 1:6], caption = "Group Rows") %>%
  kable_styling("striped", full_width = F) %>%
  group_rows("Group 1", 4, 7) %>%
  group_rows("Group 2", 8, 10)
```

Group Rows

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160.0	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875
Datsun 710	22.8	4	108.0	93	3.85	2.320
Group 1						
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440
Valiant	18.1	6	225.0	105	2.76	3.460
Duster 360	14.3	8	360.0	245	3.21	3.570
Group 2						
Merc 240D	24.4	4	146.7	62	3.69	3.190
Merc 230	22.8	4	140.8	95	3.92	3.150
Merc 280	19.2	6	167.6	123	3.92	3.440

Another way to use `group_rows` is to provide an grouping index, similar with `add_header_above()`. This feature is only available in kableExtra > 0.5.2.

```
# Not evaluated. This example generates the same table as above.
kable(mtcars[1:10, 1:6], caption = "Group Rows") %>%
  kable_styling("striped", full_width = F) %>%
  group_rows(index = c(" " = 3, "Group 1" = 4, "Group 2" = 3))
```

For advanced users, you can even define your own css for the group labeling.

```
kable(dt) %>%
  kable_styling("striped", full_width = F) %>%
  group_rows("Group 1", 3, 5, label_row_css = "background-color: #666; color: #fff;
")
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Group 1						

Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Row indentation

Unlike `group_rows()`, which will insert a labeling row, sometimes we want to list a few sub groups under a total one. In that case, `add_indent()` is probably more apporiate. For advanced users, you can even define your own css for the group labeling.

```
kable(dt) %>%
  kable_styling("striped", full_width = F) %>%
  add_indent(c(1, 3, 5))
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Group rows via multi-row cell

Function `group_rows` is great for showing simple structural information on rows but sometimes people may need to show structural information with multiple layers. When it happens, you may consider to use `collapse_rows` instead, which will put repeating cells in columns into multi-row cells. The vertical allignment of the cell is controlled by `valign` with default as “top”.

```
collapse_rows_dt <- data.frame(C1 = c(rep("a", 10), rep("b", 5)),
                 C2 = c(rep("c", 7), rep("d", 3), rep("c", 2), rep("d", 3)),
                 C3 = 1:15,
                 C4 = sample(c(0,1), 15, replace = TRUE))
kable(collapse_rows_dt, align = "c") %>%
  kable_styling(full_width = F) %>%
  column_spec(1, bold = T) %>%
  collapse_rows(columns = 1:2, valign = "top")
```

C1	C2	C3	C4
a	c	1	1

		2	0
		3	1
		4	0
		5	1
		6	0
		7	0
	d	8	1
		9	1
		10	1
b	c	11	0
		12	1
	d	13	0
		14	1
		15	1

Table Footnote

> Now it's recommended to use the new `footnote` function instead of `add_footnote` to make table footnotes.

Documentations for `add_footnote` can be found here (http://haozhu233.github.io/kableExtra/legacy_features#add_footnote).

There are four notation systems in `footnote`, namely `general`, `number`, `alphabet` and `symbol`. The last three types of footnotes will be labeled with corresponding marks while `general` won't be labeled. You can pick any one of these systems or choose to display them all for fulfill the APA table footnotes requirements.

```
kable(dt, align = "c") %>%
  kable_styling(full_width = F) %>%
  footnote(general = "Here is a general comments of the table. ",
           number = c("Footnote 1; ", "Footnote 2; "),
           alphabet = c("Footnote A; ", "Footnote B; "),
           symbol = c("Footnote Symbol 1; ", "Footnote Symbol 2")
           )
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Note:
Here is a general comments of the table.
[1] Footnote 1;
[2] Footnote 2;
[a] Footnote A;
[b] Footnote B;
[*] Footnote Symbol 1;
[†] Footnote Symbol 2

You can also specify title for each category by using the `***_title` arguments. Default value for `general_title` is "Note:" and "" for the rest three. You can also change the order using `footnote_order`. You can even display footnote as chunk texts (default is as a list) using `footnote_as_chunk`. The font format of the titles are controlled by `title_format` with options including "italic" (default), "bold" and "underline".

```
kable(dt, align = "c") %>%
  kable_styling(full_width = F) %>%
  footnote(general = "Here is a general comments of the table. ",
           number = c("Footnote 1; ", "Footnote 2; "),
           alphabet = c("Footnote A; ", "Footnote B; "),
           symbol = c("Footnote Symbol 1; ", "Footnote Symbol 2"),
           general_title = "General: ", number_title = "Type I: ",
           alphabet_title = "Type II: ", symbol_title = "Type III: ",
           footnote_as_chunk = T, title_format = c("italic", "underline")
           )
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

General: Here is a general comments of the table.

Type I: [1] Footnote 1; [2] Footnote 2;

Type II: [a] Footnote A; [b] Footnote B;
Type III: * Footnote Symbol 1; † Footnote Symbol 2

If you need to add footnote marks in table, you need to do it manually (no fancy) using `footnote_mark_***()`. Remember that similar with `cell_spec`, you need to tell this function whether you want it to do it in `HTML` (default) or `LaTeX`. You can set it for all using the `knitr.table.format` global option. ALso, if you have ever use `footnote_mark_***()`, you need to put `escape = F` in your `kable` function to avoid escaping of special characters.

```
dt_footnote <- dt
names(dt_footnote)[2] <- paste0(names(dt_footnote)[2],
                                footnote_marker_symbol(1))
row.names(dt_footnote)[4] <- paste0(row.names(dt_footnote)[4],
                                footnote_marker_alphabet(1))
kable(dt_footnote, align = "c",
      # Remember this escape = F
      escape = F) %>%
  kable_styling(full_width = F) %>%
  footnote(alphabet = "Footnote A; ",
           symbol = "Footnote Symbol 1; ",
           alphabet_title = "Type II: ", symbol_title = "Type III: ",
           footnote_as_chunk = T)
```

	mpg	cyl*	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive[a]	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

Type II: [a] Footnote A;
Type III: * Footnote Symbol 1;

HTML Only Features

Scroll box

If you have a huge table and you don't want to reduce the font size to unreadable, you may want to put your HTML table in a scroll box, of which users can pick the part they like to read. Note that scroll box isn't printer friendly, so be aware of that when you use this feature.

When you use `scroll_box`, you can specify either `height` or `width`. When you specify `height`, you will get a vertically scrollable box and vice versa. If you specify both, you will get a two-way scrollable box.

```
kable(cbind(mtcars, mtcars)) %>%
  kable_styling() %>%
  scroll_box(width = "500px", height = "200px")
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	g
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	
Hornet 4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	

You can also specify width using a percentage.

```
kable(cbind(mtcars, mtcars)) %>%
  kable_styling() %>%
  scroll_box(width = "100%", height = "200px")
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	mpg
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	21.0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	21.0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	22.8
Hornet 4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	21.4

Save HTML table directly

If you need to save those HTML tables but you don't want to generate them through rmarkdown, you can try to use the `save_kable()` function. You can choose whether to let those HTML files be self contained (default is yes). Self contained files packed CSS into the HTML file so they are quite large when there are many.

```
kable(mtcars) %>%
  kable_styling() %>%
  save_kable(file = "table1.html", self_contained = T)
```

From other packages

Since the structure of `kable` is relatively simple, it shouldn't be too difficult to convert HTML or LaTeX tables generated by other packages to a `kable` object and then use `kableExtra` to modify the outputs. If you are a package author, feel free to reach out to me and we can collaborate.

`tables`

The latest version of `tables` (https://CRAN.R-project.org/package=tables) comes with a `toKable()` function, which is compatiable with functions in `kableExtra` (>=0.9.0).