tinytable

Easy, beautiful, and customizable tables in R

Table of contents

1 Tiny Tables **3**
- 1.1 Output formats . . . 3
- 1.2 Themes . . . 4
- 1.3 Alignment . . . 5
- 1.4 Formatting (numbers, dates, strings, etc.) . . . 6
- 1.5 Width . . . 8
- 1.6 Line breaks and text wrapping . . . 8
- 1.7 Captions and cross-references . . . 9
- 1.8 Footnotes . . . 10
- 1.9 Math . . . 11

2 Style **12**
- 2.1 Cells, rows, columns . . . 13
- 2.2 Colors . . . 15
- 2.3 Fonts . . . 16
- 2.4 Spanning cells . . . 17
- 2.5 Headers . . . 17
- 2.6 Conditional styling . . . 18
- 2.7 Vectorized styling (heatmaps) . . . 19

3 Groups and labels **20**
- 3.1 Rows . . . 21
- 3.2 Columns . . . 22

4 HTML customization **25**
- 4.1 Themes . . . 25
- 4.2 CSS declarations . . . 25
- 4.3 CSS rules . . . 25

5 LaTeX / PDF customization **25**
5.1 Preamble 25
5.2 Introduction to `tabularray` 26
5.3 `tabularray` keys 28

`tinytable` is a small but powerful `R` package to draw HTML, LaTeX, PDF, Markdown, and Typst tables. The interface is minimalist, but it gives users direct and convenient access to powerful frameworks to create endlessly customizable tables.

Install it from Github:

```
library(remotes)
install_github("vincentarelbundock/tinytable")
```

This tutorial introduces the main functions of the package. It is available in two versions:

- PDF
- HTML

1 Tiny Tables

Load the library and set some global options:

```
library(tinytable)

options(digits = 3) # how many significant digits to print by default
options("tt_tabularray_placement" = "H") # for LaTeX
```

Draw a first table:

```
x <- mtcars[1:4, 1:5]
tt(x)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

1.1 Output formats

`tinytable` can produce tables in HTML, Markdown, or LaTeX (PDF) format. To choose, we use the `output` argument:

```
tt(x, output = "html")
tt(x, output = "latex")
tt(x, output = "markdown")
```

When calling `tinytable` from a Quarto or Rmarkdown document, `tinytable` detects the output format automatically and generates an HTML or LaTeX table as appropriate. This means that we do not need to explicitly specify the `output` format.

With the `save_tt()` function, users can also save tables directly to PNG images or PDF documents, or any of the basic formats. All we need to do is supply a valid file name with the appropriate extension (ex: `.png`, `.html`, `.pdf`, etc.):

```
# PNG
tt(x, output = "html") |> save_tt("path/to/file.png")

# HTML
tt(x, output = "html") |> save_tt("path/to/file.html")

# PDF
tt(x, output = "latex") |> save_tt("path/to/file.pdf")

# Text
tt(x, output = "markdown") |> save_tt("path/to/file.txt")
```

1.2 Themes

`tinytable` offers a few basic themes out of the box: "default", "striped", "grid", "void." Those themes can be applied with the `theme` argument of the `tt()` function. As we will see below, it is easy to go much beyond those basic settings to customize your own tables. Here we only illustrate a few of the simplest settings:

```
tt(x, theme = "striped")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

```
tt(x, theme = "grid")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

```
tt(x, theme = "void")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

1.3 Alignment

To align columns, we use a single character, or a string where each letter represents a column:

```
dat <- data.frame(
  a = c("a", "aa", "aaa"),
  b = c("b", "bb", "bbb"),
  c = c("c", "cc", "ccc"))

tt(dat) |> style_tt(align = "c")
```

a	b	c
a	b	c
aa	bb	cc
aaa	bbb	ccc

```
tt(dat) |> style_tt(j = 1:3, align = "lcr")
```

a	b	c
a	b	c
aa	bb	cc
aaa	bbb	ccc

1.4 Formatting (numbers, dates, strings, etc.)

The `tt()` function is minimalist; it's inteded purpose is simply to draw nice tables. Users who want to format numbers, dates, strings, and other variables in different ways should process their data *before* supplying it to the `tt()` table-drawing function. To do so, we can use the `format_tt()` function supplied by the `tinytable`.

In a very simple case—such as printing 2 significant digits of all numeric variables—we can use the `digits` argument of `tt()`:

```
dat <- data.frame(
     w = c(143002.2092, 201399.181, 100188.3883),
     x = c(1.43002, 201.399, 0.100188),
     y = as.Date(sample(1:1000, 3)),
     z = c(TRUE, TRUE, FALSE))

tt(dat, digits = 2)
```

w	x	y	z
143002	1.4	1971-11-23	True
201399	201.4	1971-06-30	True
100188	0.1	1970-03-09	False

We can get more fine-grained control over formatting with calling `format_tt()` before `tt()`:

```
dat |>
  format_tt(
    j = 2:4,
    digits = 1,
    date = "%B %d %Y") |>
  format_tt(
    j = 1,
    digits = 2,
```

```
    num_mark_big = " ",
    num_mark_dec = ",",
    num_fmt = "decimal") |>
  tt()
```

w	x	y	z
143 002,21	1.4	November 23 1971	True
201 399,18	201.4	June 30 1971	True
100 188,39	0.1	March 09 1970	False

We can use a regular expression in `j` to select columns, and the `?sprintf` function to format strings, numbers, and to do string interpolation (similar to the `glue` package, but using Base `R`):

```
dat <- data.frame(
     a = c("Burger", "Halloumi", "Tofu"),
     b = c(1.43002, 201.399, 0.100188))

dat |>
  format_tt(j = "a", sprintf = "Food: %s") |>
  tt()
```

a	b
Food: Burger	1.43
Food: Halloumi	201.40
Food: Tofu	0.10

Finally, if you like the `format_tt()` interface, you can use it with vectors too:

```
k <- c(98938272783457, 7288839482, 29111727, 93945, 922)
format_tt(k, digits = 1, num_suffix = TRUE)
```

```
[1] "98.9T" "7.3B"  "29.1M" "93.9K" "922"
```

1.5 Width

The `width` arguments accepts a number between 0 and 1, indicating what proportion of the linewidth the table should cover:

```
tt(x, width = 0.5)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

```
tt(x, width = 1)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

1.6 Line breaks and text wrapping

When the `width` argument is specified and a cell includes long text, the text is automatically wrapped to match the table.

```
lorem <- data.frame(
  Lorem = "Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium dolo
  Ipsum = " Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, se
)
tt(lorem, width = 3/4)
```

Table 1: A full width table with wrapped text.

Lorem	Ipsum
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.	Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos.

Manual line breaks work sligthly different in LaTeX (PDF) or HTML. This table shows the two strategies. For HTML, we insert a `<br>` tag. For LaTeX, we wrap the string in curly braces `{}`, and then insert two (escaped) backslashes: `\\\\`

```
d <- data.frame(
  "{Sed ut \\\\ perspiciatis unde}",
  "dicta sunt<br> explicabo. Nemo"
) |> setNames(c("LaTeX line break", "HTML line break"))
tt(d, width = 1)
```

LaTeX line break	HTML line break
Sed ut perspiciatis unde	dicta sunt explicabo. Nemo

1.7 Captions and cross-references

In Quarto, one can specify captions and use cross-references using code like this:

````
@tbl-blah shows that...

```{r}
#| label: tbl-blah
#| tbl-cap: "Blah blah blah"
library(tinytable)
tt(mtcars[1:4, 1:4])
```
````

And here is the rendered version of the code chunk above:

Table 2: Blah blah blah

mpg	cyl	disp	hp
21.0	6	160	110
21.0	6	160	110
22.8	4	108	93
21.4	6	258	110

Table 2 shows that…

```
library(tinytable)
tt(mtcars[1:4, 1:4], placement = NULL)
```

For standalone LaTeX tables, you can use the `caption` argument like so:

```
tt(x, caption = "Blah blah.\\label{tbl-blah}")
```

Be aware that this more approach may not work well in Quarto or Rmarkdown documents.

1.8 Footnotes

The `notes` argument accepts single strings or named lists of strings:

```
n <- "Fusce id ipsum consequat ante pellentesque iaculis eu a ipsum. Mauris id ex in nulla

tt(lorem, notes = n, width = 1)
```

Table 3: A full-width table with wrapped text in cells and a footnote.

Lorem	Ipsum
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.	Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos.

Fusce id ipsum consequat ante pellentesque iaculis eu a ipsum. Mauris id ex in nulla consectetur aliquam. In nec tempus diam. Aliquam arcu nibh, dapibus id ex vestibulum, feugiat consequat erat. Morbi feugiat dapibus malesuada. Quisque vel ullamcorper felis. Aenean a sem at nisi tempor pretium sit amet quis lacus.

When `notes` is a named list, the names are used as identifiers and displayed as superscripts:

```
tt(x, notes = list(a = "Blah.", b = "Blah blah."))
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

[a] Blah.
[b] Blah blah.

1.9 Math

In LaTeX and MathJax (for HTML), there are two main ways to enclose mathematical expressions, either between dollar signs or escaped parentheses: `$...$` or `\(...\)`. The first strategy is discouraged by MathJax, because dollar signs are very common in non-mathematical contexts, which can cause rendering errors. In that spirit, `tinytable` will not render dollar-enclosed strings as mathematical expressions in HTML. Following the default MathJax settings, `tinytable` expects users to employ the escaped parentheses strategy:

```
dat <- data.frame(Math = c(
  "\\( x^2 + y^2 = z^2 \\)",
  "\\( \\frac{1}{2} \\)"
))
tt(dat) |> style_tt(align = "c")
```

Math
$x^2 + y^2 = z^2$
$\frac{1}{2}$

In LaTeX (PDF), you can also use the `mode` inner setting from `tabularray` to render math in tables without delimiters (see Section 5 for details on `tabularray`):

```
dat <- data.frame(Math = c("x^2 + y^2 = z^2", "\\frac{1}{2}"))
tt(dat) |>
  style_tt(align = "c", tabularray_inner = "column{1}={mode=math},")
```

Math
$x^2 + y^2 = z^2$
$\frac{1}{2}$

2 Style

The main styling function for the `tinytable` package is `style_tt()`. Via this function, you can access three main interfaces to customize tables:

1. A general interface to frequently used style choices which works for both HTML and LaTeX (PDF): colors, font style and size, row and column spans, etc. This is accessed through several distinct arguments in the `style_tt()` function, such as `italic`, `color`, etc.
2. A specialized interface which allows users to use the powerful `tabularray` package to customize LaTeX tables. This is accessed by passing `tabularray` settings as strings to the `tabularray_inner` and `tabularray_outer` arguments of `style_tt()`.
3. A specialized interface which allows users to use the powerful `Bootstrap` framework to customize HTML tables. This is accessed by passing CSS declarations and rules to the `bootstrap_css` and `bootstrap_css_rule` arguments of `style_tt()`.

These functions can be used to customize rows, columns, or individual cells. They control many features, including:

- Text color
- Background color
- Widths
- Heights
- Alignment
- Text Wrapping
- Column and Row Spacing
- Cell Merging
- Multi-row or column spans
- Border Styling
- Font Styling: size, underline, italic, bold, strikethrough, etc.
- Header Customization

The `style_*()` functions can modify individual cells, or entire columns and rows. The portion of the table that is styled is determined by the `i` (rows) and `j` (columns) arguments.

2.1 Cells, rows, columns

To style individual cells, we use the `style_cell()` function. The first two arguments—`i` and `j`—identify the cells of interest, by row and column numbers respectively. To style a cell in the 2nd row and 3rd column, we can do:

```
tt(x) |>
  style_tt(
    i = 2,
    j = 3,
    background = "black",
    color = "white")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

The `i` and `j` accept vectors of integers to modify several cells at once:

```
tt(x) |>
  style_tt(
    i = 2:3,
    j = c(1, 3, 4),
    italic = TRUE,
    color = "orange")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	*160*	*110*	3.90
22.8	4	*108*	*93*	3.85
21.4	6	258	110	3.08

We can style all cells in a table by omitting both the `i` and `j` arguments:

```
tt(x) |> style_tt(color = "orange")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

We can style entire rows by omitting the `j` argument:

```
tt(x) |> style_tt(i = 1:2, color = "orange")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

We can style entire columns by omitting the `i` argument:

```
tt(x) |> style_tt(j = c(2, 4), bold = TRUE)
```

mpg	**cyl**	disp	**hp**	drat
21.0	**6**	160	**110**	3.90
21.0	**6**	160	**110**	3.90
22.8	**4**	108	**93**	3.85
21.4	**6**	258	**110**	3.08

The `j` argument accepts integer vectors, but also a string with a Perl-style regular expression, which makes it easier to select columns by name:

```
tt(x) |> style_tt(j = "mpg|drat", color = "orange")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

Of course, we can also call the `style_tt()` function several times to apply different styles to different parts of the table:

```
tt(x) |>
  style_tt(i = 1, j = 1:2, color = "orange") |>
  style_tt(i = 1, j = 3:4, color = "green")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

2.2 Colors

The `color` and `background` arguments in the `style_tt()` function are used for specifying the text color and the background color for cells of a table created by the `tt()` function. This argument plays a crucial role in enhancing the visual appeal and readability of the table, whether it's rendered in LaTeX or HTML format. The way we specify colors differs slightly between the two formats:

For HTML Output:

- Hex Codes: You can specify colors using hexadecimal codes, which consist of a `#` followed by 6 characters (e.g., `#CC79A7`). This allows for a wide range of colors.
- Keywords: There's also the option to use color keywords for convenience. The supported keywords are basic color names like `black`, `red`, `blue`, etc.

For LaTeX Output:

- Hexadecimal Codes: Similar to HTML, you can use hexadecimal codes. However, in LaTeX, you need to include these codes as strings (e.g., `"#CC79A7"`).

- Keywords: LaTeX supports a different set of color keywords, which include standard colors like `black`, `red`, `blue`, as well as additional ones like `cyan`, `darkgray`, `lightgray`, etc.
- Color Blending: An advanced feature in LaTeX is color blending, which can be achieved using the `xcolor` package. You can blend colors by specifying ratios (e.g., `white!80!blue` or `green!20!red`).
- Luminance Levels: The `ninecolors` package in LaTeX offers colors with predefined luminance levels, allowing for more nuanced color choices (e.g., "azure4", "magenta8").

Note that the keywords used in LaTeX and HTML are slightly different.

```
tt(x) |> style_tt(i = 1:4, j = 1, color = "#FF5733")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

Note that when using Hex codes in a LaTeX table, we need extra declarations in the LaTeX preamble. See `?tt` for details.

2.3 Fonts

The font size is specified in terms of `pt` units, where 1pt=1.333px:

```
tt(x) |> style_tt(j = "mpg|hp|qsec", fontsize = 18)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

2.4 Spanning cells

Sometimes, it can be useful to make a cell stretch across multiple colums, for example when we want to insert a label. To achieve this, we can use the `colspan` argument. Here, we make the 2nd cell of the 2nd row stretch across three columns:

```
tt(x)|> style_tt(
  i = 2, j = 2,
  colspan = 3,
  align = "c",
  color = "white",
  background = "black")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0		6		3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

Here is the original table for comparison:

```
tt(x)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

2.5 Headers

The header can be omitted from the table by deleting the column names in the `x` data frame:

```
k <- x
colnames(k) <- NULL
tt(k)
```

21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

The header is row 0, and can thus be styled as expected:

```
tt(x) |> style_tt(i = 0, color = "white", background = "black")
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

2.6 Conditional styling

We can use the standard `which` function from Base `R` to create indices and apply conditional stying on rows. And we can use a regular expression in `j` to apply conditional styling on columns:

```
k <- mtcars[1:10, c("mpg", "am", "vs")]

tt(k) |>
  style_tt(
    i = which(k$am == k$vs),
    background = "teal",
    color = "white")
```

mpg	am	vs
21.0	1	0
21.0	1	0
22.8	1	1
21.4	0	1
18.7	0	0
18.1	0	1
14.3	0	0
24.4	0	1
22.8	0	1
19.2	0	1

2.7 Vectorized styling (heatmaps)

The `color`, `background`, and `fontsize` arguments are vectorized. This allows easy specification of different colors in a single call:

```
tt(x) |>
  style_tt(
    i = 1:4,
    color = c("red", "blue", "green", "orange"))
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

When using a single value for a vectorized argument, it gets applied to all values:

```
tt(x) |>
  style_tt(
    j = 2:3,
    color = c("orange", "green"),
    background = "black")
```

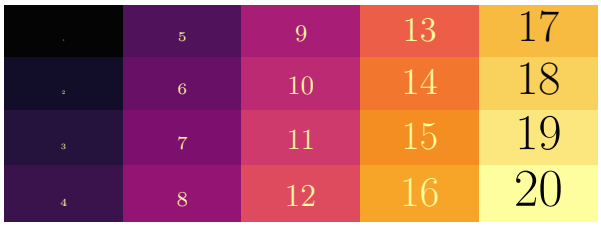
mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08

We can also produce more complex heatmap-like tables:

```
# A table without header
k <- data.frame(matrix(1:20, ncol = 5))
colnames(k) <- NULL

# 20 levels of Inferno colors
bg <- hcl.colors(20, "Inferno")
fg <- ifelse(as.matrix(k) < 17, tail(bg, 1), head(bg, 1))
fs <- 1:20

tt(k, width = .5, theme = "void") |>
  style_tt(j = 1:5, align = "ccccc") |>
  style_tt(
    i = 1:4,
    j = 1:5,
    color = fg,
    background = bg,
    fontsize = fs)
```



3 Groups and labels

The `group_tt()` function can label groups of rows (`i`) or columns (`j`).

3.1 Rows

The `i` argument accepts a named list of integers. The numbers identify the positions where row group labels are to be inserted. The names includes the text that should be inserted:

```
dat <- mtcars[1:9, 1:8]

tt(dat) |>
  group_tt(i = list(
    "I like (fake) hamburgers" = 3,
    "She prefers halloumi" = 4,
    "They love tofu" = 7))
```

mpg	cyl	disp	hp	drat	wt	qsec	vs
21.0	6	160	110	3.90	2.62	16.5	0
21.0	6	160	110	3.90	2.88	17.0	0
I like (fake) hamburgers							
22.8	4	108	93	3.85	2.32	18.6	1
She prefers halloumi							
21.4	6	258	110	3.08	3.21	19.4	1
18.7	8	360	175	3.15	3.44	17.0	0
18.1	6	225	105	2.76	3.46	20.2	1
They love tofu							
14.3	8	360	245	3.21	3.57	15.8	0
24.4	4	147	62	3.69	3.19	20.0	1
22.8	4	141	95	3.92	3.15	22.9	1

The `group_tt()` function only includes a few arguments: `x`, `i`, `j`, and `indent`. But whenever we call `group_tt()`, the function will automatically apply a `style_tt()` call to all the new group labels, using any extra argument supplied to `group_tt()` (arguments are pushed via `...`). This means that we can apply all the usual stying options to row labels:

```
tt(dat) |>
  group_tt(
    align = "c",
    color = "white",
    background = "gray",
```

```
    bold = TRUE,
    i = list(
      "I like (fake) hamburgers" = 3,
      "She prefers halloumi" = 4,
      "They love tofu" = 7))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs
	21.0	6	160	110	3.90	2.62	16.5	0
	21.0	6	160	110	3.90	2.88	17.0	0
I like (fake) hamburgers								
	22.8	4	108	93	3.85	2.32	18.6	1
She prefers halloumi								
	21.4	6	258	110	3.08	3.21	19.4	1
	18.7	8	360	175	3.15	3.44	17.0	0
	18.1	6	225	105	2.76	3.46	20.2	1
They love tofu								
	14.3	8	360	245	3.21	3.57	15.8	0
	24.4	4	147	62	3.69	3.19	20.0	1
	22.8	4	141	95	3.92	3.15	22.9	1

3.2 Columns

The syntax for column groups is very similar, but we use the `j` argument instead. The named list specifies the labels to appear in column-spanning labels, and the values must be a vector of consecutive and non-overlapping integers that indicate which columns are associated to which labels:

```
tt(dat) |>
  group_tt(
    j = list(
      "Hamburgers" = 1:3,
      "Halloumi" = 4:5,
      "Tofu" = 7))
```

Hamburgers			Halloumi			Tofu	
mpg	cyl	disp	hp	drat	wt	qsec	vs
21.0	6	160	110	3.90	2.62	16.5	0
21.0	6	160	110	3.90	2.88	17.0	0
22.8	4	108	93	3.85	2.32	18.6	1
21.4	6	258	110	3.08	3.21	19.4	1
18.7	8	360	175	3.15	3.44	17.0	0
18.1	6	225	105	2.76	3.46	20.2	1
14.3	8	360	245	3.21	3.57	15.8	0
24.4	4	147	62	3.69	3.19	20.0	1
22.8	4	141	95	3.92	3.15	22.9	1

As above, we can pass additional styling options to the `style_tt()` function automatically via `...`. This means that all the arguments like `italic`, `bold`, `color` and friends can be used to style spanning column headers:

```
dat <- mtcars[1:9, 1:8]
tt(dat) |>
  group_tt(color = "teal", italic = TRUE,
    j = list("Hamburgers" = 1:3,
             "Halloumi" = 4:5,
             "Tofu" = 7),
    i = list("I like (fake) hamburgers" = 3,
             "She prefers halloumi" = 4,
             "They love tofu" = 7))
```

Hamburgers			*Halloumi*			*Tofu*	
mpg	cyl	disp	hp	drat	wt	qsec	vs
21.0	6	160	110	3.90	2.62	16.5	0
21.0	6	160	110	3.90	2.88	17.0	0
I like (fake) hamburgers							
22.8	4	108	93	3.85	2.32	18.6	1
She prefers halloumi							
21.4	6	258	110	3.08	3.21	19.4	1
18.7	8	360	175	3.15	3.44	17.0	0
18.1	6	225	105	2.76	3.46	20.2	1
They love tofu							
14.3	8	360	245	3.21	3.57	15.8	0
24.4	4	147	62	3.69	3.19	20.0	1
22.8	4	141	95	3.92	3.15	22.9	1

Or call twice for different stylings for row and column groups:

```
dat <- mtcars[1:9, 1:8]
tt(dat) |>
  group_tt(color = "teal", italic = TRUE,
    j = list("Hamburgers" = 1:3,
             "Halloumi" = 4:5,
             "Tofu" = 7)) |>
  group_tt(align = "c", color = "white", background = "teal", bold = TRUE,
    i = list("I like (fake) hamburgers" = 3,
             "She prefers halloumi" = 4,
             "They love tofu" = 7))
```

Hamburgers			*Halloumi*			*Tofu*	
mpg	cyl	disp	hp	drat	wt	qsec	vs
21.0	6	160	110	3.90	2.62	16.5	0
21.0	6	160	110	3.90	2.88	17.0	0
I like (fake) hamburgers							
22.8	4	108	93	3.85	2.32	18.6	1
She prefers halloumi							
21.4	6	258	110	3.08	3.21	19.4	1
18.7	8	360	175	3.15	3.44	17.0	0
18.1	6	225	105	2.76	3.46	20.2	1
They love tofu							
14.3	8	360	245	3.21	3.57	15.8	0
24.4	4	147	62	3.69	3.19	20.0	1
22.8	4	141	95	3.92	3.15	22.9	1

4 HTML customization

The HTML customization options described in this section are not available for LaTeX (or PDF) documents. Please refer to the web documentation to read this part of the tutorial.

4.1 Themes

4.2 CSS declarations

4.3 CSS rules

5 LaTeX / PDF customization

5.1 Preamble

Warning: Some of the features of this package may require a recent version of the `tabularray` package. Please update your local LaTeX distribution before using `tinytable`.

In Rmarkdown and Quarto documents, `tinytable` will automatically populate your LaTeX preamble with the necessary packages and commands. When creating your own LaTeX documents, you should insert these commands in the preamble:

```
\usepackage{tabularray}
\usepackage{float}
\usepackage{codehigh}
\usepackage[normalem]{ulem}
\UseTblrLibrary{booktabs}
\NewTableCommand{\tinytableDefineColor}[3]{\definecolor{#1}{#2}{#3}}
\newcommand{\tinytableTabularrayUnderline}[1]{\underline{#1}}
\newcommand{\tinytableTabularrayStrikeout}[1]{\sout{#1}}
```

5.2 Introduction to `tabularray`

`tabularray` offers a robust solution for creating and managing tables in LaTeX, standing out for its flexibility and ease of use. It excels in handling complex table layouts and offers enhanced functionality compared to traditional LaTeX table environments. This package is particularly useful for users requiring advanced table features, such as complex cell formatting, color management, and versatile table structures.

A key feature of Tabularray is its separation of style from content. This approach allows users to define the look and feel of their tables (such as color, borders, and text alignment) independently from the actual data within the table. This separation simplifies the process of formatting tables and enhances the clarity and maintainability of LaTeX code. The `tabularray` documentation is fantastic. It will teach you how to customize virtually every aspect of your tables: https://ctan.org/pkg/tabularray?lang=en

Tabularray introduces a streamlined interface for specifying table settings. It employs two types of settings blocks: Inner and Outer. The Outer block is used for settings that apply to the entire table, like overall alignment, while the Inner block handles settings for specific elements like columns, rows, and cells. The `style_tt()` function includes `tabularray_inner` and `tabularray_outer` arguments to set these respective features.

Consider this `tabularray` example, which illustrates the use of inner settings:

```
\begin{table}
\centering
\begin{tblr}[         %% tabularray outer open
]                     %% tabularray outer close
{                     %% tabularray inner open
column{1-4}={halign=c},
hlines = {bg=white},
vlines = {bg=white},
cell{1,6}{odd} = {bg=teal7},
cell{1,6}{even} = {bg=green7},
```

```
cell{2,4}{1,4} = {bg=red7},
cell{3,5}{1,4} = {bg=purple7},
cell{2}{2} = {r=4,c=2}{bg=azure7},
}                       %% tabularray inner close
mpg & cyl & disp & hp \\
21 & 6 & 160 & 110 \\
21 & 6 & 160 & 110 \\
22.8 & 4 & 108 & 93 \\
21.4 & 6 & 258 & 110 \\
18.7 & 8 & 360 & 175 \\
\end{tblr}
\end{table}
```

The Inner block, enclosed in `{}`, defines specific styles like column formats (`column{1-4}={halign=c}`), horizontal and vertical line colors (`hlines={fg=white}`, `vlines={fg=white}`), and cell colorations (`cell{1,6}{odd}={bg=teal7}`, etc.). The last line of the inner block also species that the second cell of row 2 (`cell{2}{2}`) should span 4 rows and 2 columns (`{r=4,c=3}`), be centered (`halign=c`), and with a background color with the 7th luminance level of the azure color (`bg=azure7`).

We can create this code easily by passing a string to the `tabularray_inner` argument of the `style_tt()` function:

```
inner <- "
column{1-4}={halign=c},
hlines = {fg=white},
vlines = {fg=white},
cell{1,6}{odd} = {bg=teal7},
cell{1,6}{even} = {bg=green7},
cell{2,4}{1,4} = {bg=red7},
cell{3,5}{1,4} = {bg=purple7},
cell{2}{2} = {r=4,c=2}{bg=azure7},
"
mtcars[1:5, 1:4] |>
  tt(output = "latex", theme = "void") |>
  style_tt(tabularray_inner = inner)
```

Table 4: LaTeX table with colors and a spanning cell.

mpg	cyl	disp	hp
21.0	6		110
21.0			110
22.8			93
21.4			110
18.7	8	360	175

5.3 `tabularray` keys

Inner specifications:

Key	Description and Values	Initial Value
`rulesep`	space between two hlines or vlines	`2pt`
`stretch`	stretch ratio for struts added to cell text	`1`
`abovesep`	set vertical space above every row	`2pt`
`belowsep`	set vertical space below every row	`2pt`
`rowsep`	set vertical space above and below every row	`2pt`
`leftsep`	set horizontal space to the left of every column	`6pt`
`rightsep`	set horizontal space to the right of every column	`6pt`
`colsep`	set horizontal space to both sides of every column	`6pt`
`hspan`	horizontal span algorithm: `default`, `even`, or `minimal`	`default`
`vspan`	vertical span algorithm: `default` or `even`	`default`
`baseline`	set the baseline of the table	`m`

Outer specifications:

Key	Description and Values	Initial Value
`baseline`	set the baseline of the table	`m`
`long`	change the table to a long table	None
`tall`	change the table to a tall table	None
`expand`	you need this key to use verb commands	None

Cells:

Key	Description and Values	Initial Value
halign	horizontal alignment: l (left), c (center), r (right) or j (justify)	j
valign	vertical alignment: t (top), m (middle), b (bottom), h (head) or f (foot)	t
wd	width dimension	None
bg	background color name	None
fg	foreground color name	None
font	font commands	None
mode	set cell mode: math, imath, dmath or text	None
cmd	execute command for the cell text	None
preto	prepend text to the cell	None
appto	append text to the cell	None
r	number of rows the cell spans	1
c	number of columns the cell spans	1

Rows:

Key	Description and Values	Initial Value
halign	horizontal alignment: l (left), c (center), r (right) or j (justify)	j
valign	vertical alignment: t (top), m (middle), b (bottom), h (head) or f (foot)	t
ht	height dimension	None
bg	background color name	None
fg	foreground color name	None
font	font commands	None
mode	set mode for row cells: math, imath, dmath or text	None
cmd	execute command for every cell text	None
abovesep	set vertical space above the row	2pt
belowsep	set vertical space below the row	2pt
rowsep	set vertical space above and below the row	2pt
preto	prepend text to every cell (like > specifier in rowspec)	None
appto	append text to every cell (like < specifier in rowspec)	None

Columns:

Key	Description and Values	Initial Value
halign	horizontal alignment: l (left), c (center), r (right) or j (justify)	j

Key	Description and Values	Initial Value
`valign`	vertical alignment: `t` (top), `m` (middle), `b` (bottom), `h` (head) or `f` (foot)	`t`
`wd`	width dimension	None
`co`	coefficient for the extendable column (`X` column)	None
`bg`	background color name	None
`fg`	foreground color name	None
`font`	font commands	None
`mode`	set mode for column cells: `math`, `imath`, `dmath` or `text`	None
`cmd`	execute command for every cell text	None
`leftsep`	set horizontal space to the left of the column	`6pt`
`rightsep`	set horizontal space to the right of the column	`6pt`
`colsep`	set horizontal space to both sides of the column	`6pt`
`preto`	prepend text to every cell (like `>` specifier in `colspec`)	None
`appto`	append text to every cell (like `<` specifier in `colspec`)	None

hlines:

Key	Description and Values	Initial Value
`dash`	dash style: `solid`, `dashed` or `dotted`	`solid`
`text`	replace hline with text (like `!` specifier in `rowspec`)	None
`wd`	rule width dimension	`0.4pt`
`fg`	rule color name	None
`leftpos`	crossing or trimming position at the left side	`1`
`rightpos`	crossing or trimming position at the right side	`1`
`endpos`	adjust leftpos/rightpos for only the leftmost/rightmost column	`false`

vlines:

Key	Description and Values	Initial Value
`dash`	dash style: `solid`, `dashed` or `dotted`	`solid`
`text`	replace vline with text (like `!` specifier in `colspec`)	None
`wd`	rule width dimension	`0.4pt`
`fg`	rule color name	None
`abovepos`	crossing or trimming position at the above side	`0`
`belowpos`	crossing or trimming position at the below side	`0`