

tinytable (LaTeX)

Easy, beautiful, and customizable tables in R

Table of contents

1	Tiny Tables	4
1.1	Width and height	4
1.2	Footnotes	7
1.3	Captions and cross-references	8
1.4	Output formats	9
1.5	Combination and exploration	12
1.6	Select columns	14
1.7	Rename columns	16
2	Formatting	16
2.1	Numbers, dates, strings, etc.	16
2.2	Significant digits and decimals	19
2.3	Math	19
2.4	Replacement	21
2.5	Escape special characters	22
2.6	Line breaks	23
2.7	Markdown	24
2.8	Custom functions	25
2.9	Captions, notes, groups, and column names	27
3	Style	27
3.1	Cells, rows, columns	28
3.2	Colors	32
3.3	Alignment	33
3.4	Font size	35
3.5	Spanning cells (merging cells)	35
3.6	Headers	38
3.7	Conditional styling	39
3.8	Vectorized styling (heatmaps)	40

3.9	Lines (borders)	42
3.10	Markdown	44
4	Groups and labels	45
4.1	Rows	45
4.1.1	Styling row groups	48
4.1.2	Automatic row groups	49
4.1.3	Row matrix insertion	50
4.2	Columns	52
4.2.1	Styling column groups	53
4.2.2	Column names with delimiters	55
4.3	Case studies	56
4.3.1	Repeated column names	56
5	Themes	58
5.1	Visual themes	58
5.2	Custom themes	60
5.3	Tabular (LaTeX and HTML)	61
5.4	Combining themes	61
5.5	User-written themes	63
5.5.1	<code>theme_mitex()</code>	63
6	Plots and images	63
6.1	Inserting images in tables	64
6.2	Inline plots	65
6.2.1	Built-in plots	65
6.2.2	Custom plots: Base R	66
6.2.3	Custom plots: <code>ggplot2</code>	67
7	LaTeX / PDF customization	69
7.1	Preamble	69
7.2	Introduction to <code>tabularray</code>	70
7.3	<code>tabularray</code> keys	72
8	Tips and Tricks	75
8.1	LaTeX	75
8.1.1	Preamble	75
8.1.2	<code>setspace</code>	75
8.1.3	Multi-line cells with <code>minipage</code>	77
8.1.4	Global styles	78
8.1.5	Beamer	78
8.1.6	Label and caption position	79
8.1.7	Placement	80

8.1.8	Rotate	81
8.1.9	Multipage	82
8.2	Markdown	84
8.2.1	<code>style_tt()</code> does not apply to row headers	84
8.2.2	<code>rowspan</code> and <code>colspan</code>	84
8.3	Removing elements with <code>theme_empty()</code>	85
9	Gallery	88
9.1	Students	88
9.2	Lemurs	88
9.3	Wines	89
9.4	AI Stocks	90

1 Tiny Tables

`tinytable` is a small but powerful R package to draw HTML, LaTeX, Word, 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 the latest version from R-Universe or CRAN:

```
install.packages("tinytable",  
  repos = c("https://vincentarelbundock.r-universe.dev", "https://cran.r-project.org")  
)
```

This tutorial introduces the main functions of the package. It is also [available as a single PDF document](#).

Load the library and set some global options:

```
library(tinytable)  
options(tinytable_tt_digits = 3)  
options(tinytable_latex_placement = "H")
```

Draw a first table:

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

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

1.1 Width and height

The `width` arguments indicating what proportion of the line width the table should cover. This argument accepts a number between 0 and 1 to control the whole table width, or a vector of numeric values between 0 and 1, representing each column.

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

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

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

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

We can control individual columns by supplying a vector. In that case, the sum of `width` elements determines the full table width. For example, this table takes 70% of available width, with the first column 3 times as large as the other ones.

```
tt(x, width = c(.3, .1, .1, .1, .1))
```

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

When the sum of the `width` vector exceeds 1, it is automatically normalized to full-width. This is convenient when we only want to specify column width in relative terms:

```
tt(x, width = c(3, 2, 1, 1, 1))
```

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

When specifying a table `width`, the text is automatically wrapped to appropriate size:

```
lorem <- data.frame(
  Lorem = "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.",
  Ipsum = "Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui placeat aut tempora quibusdam qui voluptatem et dignam deserunt fugit neque dolorum et odio qui deserunt consequatur vel ita qui recusandae minus totam voluptatem ut reprehenderit sint amet."
)

tt(lorem, width = 3 / 4)
```

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.

The `height` argument controls the height of each row in em units:

```
tt(mtcars[1:4, 1:5], height = 3)
```

mpg	cyl	disp	hp	drat
21	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

1.2 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 con-  
sectetur 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."
```

Table 1: 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	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

^a Blah.

^b Blah blah.

We can also add markers in individual cells by providing coordinates:

```
tt(x, notes = list(
  a = list(i = 0:1, j = 1, text = "Blah."),
  b = "Blah blah."
))
```

mpg ^a	cyl	disp	hp	drat
21 ^a	6	160	110	3.9
21	6	160	110	3.9
22.8	4	108	93	3.85
21.4	6	258	110	3.08

^a Blah.

^b Blah blah.

1.3 Captions and cross-references

In Quarto, one should always specify captions cross-references using chunk options, and should *not* use the `caption` argument. This is because Quarto automatically post-processes tables, and may introduce conflict with the captions inserted by `tinytable`. For example:

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

```
```{r}
#| label: tbl-blah
#| tbl-cap: "Blah blah blah"
```



Table 2: Blah blah blah

mpg	cyl	disp	hp
21	6	160	110
21	6	160	110
22.8	4	108	93
21.4	6	258	110

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

```

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

Table 2 shows that...

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

One exception to the injunction above is when rendering a Quarto document to LaTeX using `theme_latex(multipage = TRUE, rowhead = 1)`. In that case, one must *avoid* using the Quarto chunk option, because these options trigger Quarto post-processing that will conflict with the `longtblr` environment used to split long tables across multiple pages.

The alternative is to use to refer to tables using standard LaTeX syntax: `\ref{tbl-ex-multipage}`. Then, use the `caption` argument in `tt()` to specify both the label and the caption:

```
tt(iris, caption = "Example table.\\label{tbl-ex-multipage}") |>
  theme_latex(multipage = TRUE, rowhead = 1)
```

For standalone tables in any format (i.e., outside Quarto), you can use the `caption` argument like so:

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

1.4 Output formats

`tinytable` can produce tables in HTML, Word, Markdown, LaTeX, Typst, PDF, or PNG format. An appropriate output format for printing is automatically selected based on (1)

whether the function is called interactively, (2) is called within RStudio, and (3) the output format of the Rmarkdown or Quarto document, if applicable. Alternatively, users can specify the print format in `print()` or by setting a global option:

```
tt(x) |> print("markdown")
tt(x) |> print("html")
tt(x) |> print("latex")

options(tinytable_print_output = "markdown")
```

With the `save_tt()` function, users can also save tables directly to PNG (images), PDF or Word documents, and to 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.):

```
tt(x) |> save_tt("path/to/file.png")
tt(x) |> save_tt("path/to/file.pdf")
tt(x) |> save_tt("path/to/file.docx")
tt(x) |> save_tt("path/to/file.html")
tt(x) |> save_tt("path/to/file.tex")
tt(x) |> save_tt("path/to/file.md")
```

`save_tt()` can also return a string with the table in it, for further processing in R. In the first case, the table is printed to console with `cat()`. In the second case, it returns as a single string as an R object.

```
tt(mtcars[1:10, 1:5]) |>
  group_tt(
    i = list(
      "Hello" = 3,
      "World" = 8
    ),
    j = list(
      "Foo" = 2:3,
      "Bar" = 4:5
    )
  ) |>
  print("markdown")
```

```
+-----+-----+-----+-----+
|      | Foo      | Bar      |
+-----+-----+-----+-----+
```

| mpg | cyl | disp | hp | drat |
|-------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| Hello | | | | |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |
| 18.7 | 8 | 360 | 175 | 3.15 |
| 18.1 | 6 | 225 | 105 | 2.76 |
| 14.3 | 8 | 360 | 245 | 3.21 |
| World | | | | |
| 24.4 | 4 | 147 | 62 | 3.69 |
| 22.8 | 4 | 141 | 95 | 3.92 |
| 19.2 | 6 | 168 | 123 | 3.92 |

```
tt(mtcars[1:10, 1:5]) |>
  group_tt(
    i = list(
      "Hello" = 3,
      "World" = 8
    ),
    j = list(
      "Foo" = 2:3,
      "Bar" = 4:5
    )
  ) |>
  save_tt("markdown")
```

```
[1] "+-----+-----+-----+-----+\n|      | Foo      | Bar      |\n+n-----+-----+-----"
```

1.5 Combination and exploration

Tables can be explored, modified, and combined using many of the usual base R functions:

```
a <- tt(mtcars[1:2, 1:2])  
a
```

| mpg | cyl |
|-----|-----|
| 21 | 6 |
| 21 | 6 |

```
dim(a)
```

```
[1] 2 2
```

```
ncol(a)
```

```
[1] 2
```

```
nrow(a)
```

```
[1] 2
```

```
names(a)
```

```
[1] "mpg" "cyl"
```

Tables can be combined with the usual `rbind()` function:

```
a <- tt(mtcars[1:3, 1:2], caption = "Combine two tiny tables.")  
b <- tt(mtcars[4:5, 8:10])  
  
rbind(a, b)
```

Table 3: Combine two tiny tables.

| mpg | cyl | vs | am | gear |
|------|-----|----|----|------|
| 21 | 6 | NA | NA | NA |
| 21 | 6 | NA | NA | NA |
| 22.8 | 4 | NA | NA | NA |
| NA | NA | vs | am | gear |
| NA | NA | 1 | 0 | 3 |
| NA | NA | 0 | 0 | 3 |

```
rbind(a, b) |> format_tt(replace = "")
```

Table 4: Combine two tiny tables.

| mpg | cyl | vs | am | gear |
|------|-----|----|----|------|
| 21 | 6 | | | |
| 21 | 6 | | | |
| 22.8 | 4 | | | |
| | | vs | am | gear |
| | | 1 | 0 | 3 |
| | | 0 | 0 | 3 |

The `rbind2()` S4 method is slightly more flexible than `rbind()`, as it supports arguments `headers` and `use_names`.

Omit y header:

```
rbind2(a, b, headers = FALSE)
```

Table 5: Combine two tiny tables.

| mpg | cyl | vs | am | gear |
|------|-----|----|----|------|
| 21 | 6 | NA | NA | NA |
| 21 | 6 | NA | NA | NA |
| 22.8 | 4 | NA | NA | NA |
| NA | NA | 1 | 0 | 3 |
| NA | NA | 0 | 0 | 3 |

Bind tables by position rather than column names:

```
rbind2(a, b, use_names = FALSE)
```

Table 6: Combine two tiny tables.

| mpg | cyl | gear |
|------|-----|------|
| 21 | 6 | NA |
| 21 | 6 | NA |
| 22.8 | 4 | NA |
| vs | am | gear |
| 1 | 0 | 3 |
| 0 | 0 | 3 |

1.6 Select columns

The `subset()` function from base R can be used to select columns from a `tinytable`. This is especially useful when applying conditional styling based on column values, and then removing them. For example, if we have a table with 6 rows and three `Species`:

```
dat <- do.call(rbind, by(iris, ~Species, head, n = 2))
dat
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---------------|--------------|-------------|--------------|-------------|------------|
| setosa.1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| setosa.2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| versicolor.51 | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| versicolor.52 | 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| virginica.101 | 6.3 | 3.3 | 6.0 | 2.5 | virginica |
| virginica.102 | 5.8 | 2.7 | 5.1 | 1.9 | virginica |

We highlight the `versicolor` rows in pink and remove the `Species` column:

```
tt(dat) |>
  style_tt(Species == "versicolor", background = "pink") |>
  subset(select = -Species)
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|--------------|-------------|--------------|-------------|
| 5.1 | 3.5 | 1.4 | 0.2 |
| 4.9 | 3 | 1.4 | 0.2 |
| 7 | 3.2 | 4.7 | 1.4 |
| 6.4 | 3.2 | 4.5 | 1.5 |
| 6.3 | 3.3 | 6 | 2.5 |
| 5.8 | 2.7 | 5.1 | 1.9 |

Or

```
tt(dat) |>
  style_tt(Species == "versicolor", background = "pink") |>
  subset(select = c(Sepal.Length, Sepal.Width))
```

| Sepal.Length | Sepal.Width |
|--------------|-------------|
| 5.1 | 3.5 |
| 4.9 | 3 |
| 7 | 3.2 |
| 6.4 | 3.2 |
| 6.3 | 3.3 |
| 5.8 | 2.7 |

1.7 Rename columns

As noted above, `tinytable` tries to be standards-compliant, by defining methods for many base R functions. The benefit of this approach is that instead of having to learn a `tinytable`-specific syntax, users can rename columns using all the tools they already know:

```
a <- tt(mtcars[1:2, 1:2])
names(a) <- c("a", "b")
a
```

| a | b |
|----|---|
| 21 | 6 |
| 21 | 6 |

In a pipe-based workflow, we can use the `setNames()` function from base R:

```
mtcars[1:2, 1:2] |>
  tt() |>
  setNames(c("a", "b"))
```

| a | b |
|----|---|
| 21 | 6 |
| 21 | 6 |

2 Formatting

```
library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")
x <- mtcars[1:4, 1:5]
```

2.1 Numbers, dates, strings, etc.

The `tt()` function is minimalist; its intended 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(1.43402, 201.399, 0.134588),  
  y = as.Date(sample(1:1000, 3), origin = "1970-01-01"),  
  z = c(TRUE, TRUE, FALSE)  
)  
  
tt(dat, digits = 2)
```

| w | x | y | z |
|--------|-------|------------|-------|
| 143002 | 1.43 | 1971-01-15 | TRUE |
| 201399 | 201.4 | 1970-01-12 | TRUE |
| 100188 | 0.13 | 1970-03-30 | FALSE |

We can get more fine-grained control over formatting by calling `format_tt()` after `tt()`, optionally by specifying the columns to format with `j`:

```
tt(dat) |>  
  format_tt(  
    j = 2:4,  
    digits = 1,  
    date = "%B %d %Y",  
    bool = tolower  
  ) |>  
  format_tt(  
    j = 1,  
    digits = 2,  
    num_mark_big = " ",  
    num_mark_dec = ",",  
    num_zero = TRUE,  
    num_fmt = "decimal"  
  )
```

| w | x | y | z |
|------------|-------|-----------------|-------|
| 143 002,21 | 1.4 | January 15 1971 | true |
| 201 399,18 | 201.4 | January 12 1970 | true |
| 100 188,39 | 0.1 | March 30 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", "Beans"),
  b = c(1.43202, 201.399, 0.146188, 0.0031),
  c = c(98938272783457, 7288839482, 29111727, 93945)
)
tt(dat) |>
  format_tt(j = "a", sprintf = "Food: %s") |>
  format_tt(j = 2, digits = 1) |>
  format_tt(j = "c", digits = 2, num_suffix = TRUE)
```

| a | b | c |
|----------------|---------|------|
| Food: Burger | 1.432 | 99T |
| Food: Halloumi | 201.399 | 7.3B |
| Food: Tofu | 0.146 | 29M |
| Food: Beans | 0.003 | 94K |

Finally, if you like the `format_tt()` interface, you can use it directly with numbers, vectors, or data frames:

```
format_tt(pi, digits = 1)
```

```
[1] "3"
```

```
format_tt(dat, digits = 1, num_suffix = TRUE)
```

```
      a      b      c
1 Burger  1 99T
2 Halloumi 201 7B
3 Tofu    0.1 29M
4 Beans  0.003 94K
```

2.2 Significant digits and decimals

By default, `format_tt()` formats numbers to ensure that the smallest value in a vector (column) has at least a certain number of significant digits. For example,

```
k <- data.frame(x = c(0.000123456789, 12.4356789))
tt(k, digits = 2)
```

| x |
|----------|
| 0.00012 |
| 12.43568 |

We can alter this behavior to ensure to round significant digits on a per-cell basis, using the `num_fmt` argument in `format_tt()`:

```
tt(k) |> format_tt(digits = 2, num_fmt = "significant_cell")
```

| x |
|---------|
| 0.00012 |
| 12 |

The numeric formatting options in `format_tt()` can also be controlled using global options:

```
options("tinytable_tt_digits" = 2)
options("tinytable_format_num_fmt" = "significant_cell")
tt(k)
```

| x |
|---------|
| 0.00012 |
| 12 |

2.3 Math

To insert LaTeX-style mathematical expressions in a `tinytable`, we enclose the expression in dollar signs: `$...$`. Note that you must double backslashes in mathematical expressions in R strings.

In LaTeX, expression enclosed between $\$$ will automatically rendered as a mathematical expression.

In HTML, users must first load the MathJax JavaScript library to render math. This can be done in two ways. First, one can use a global option. This will insert MathJax scripts alongside every table, which is convenient, but could enter in conflict with other scripts if the user (or notebook) has already inserted MathJax code:

```
options(tinytable_html_mathjax = TRUE)
```

Alternatively, users can load MathJax explicitly in their HTML file. In a Quarto notebook, this can be done by using a code chunk like this:

```
```${=html}
<script id="MathJax-script" async src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-ch
<script>
MathJax = {
 tex: {
 inlineMath: [['$', '$'], ['\\(', '\\)']]
 },
 svg: {
 fontCache: 'global'
 }
};
</script>
```
```

Then, we can do:

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

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

$$\frac{1}{2}$$

To avoid inserting $\$ \dots \$$ in every cell manually, we can use the `math` argument of `format_tt()`:

```
options(tinytable_html_mathjax = TRUE)

dat <- data.frame("y^2 = e^x" = c(-2, -pi), check.names = FALSE)

tt(dat, digits = 3) |> format_tt(math = TRUE)
```

$$\frac{y^2 = e^x}{-2}$$

$$-3.14$$

Note that math rendering may not work automatically in Rmarkdown document. See the [notebooks vignette](#) for advice on Rmarkdown documents.

In LaTeX (PDF), you can also use the `mode` inner setting from `tabulararray` to render math in tables without delimiters (see Section 7 for details on `tabulararray`):

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

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

$$\frac{1}{2}$$

2.4 Replacement

Missing values can be replaced by a custom string using the `replace` argument:

```
tab <- data.frame(a = c(NA, 1, 2), b = c(3, NA, 5))

tt(tab)
```

| a | b |
|----|----|
| NA | 3 |
| 1 | NA |
| 2 | 5 |

```
tt(tab) |> format_tt(replace = "-")
```

| a | b |
|---|---|
| - | 3 |
| 1 | - |
| 2 | 5 |

Warning: When using `quarto=TRUE`, the dash may be interpreted as the start of a list.

We can also specify multiple value replacements at once using a named list of vectors:

```
tmp <- data.frame(x = 1:5, y = c(pi, NA, NaN, -Inf, Inf))
dict <- list("-" = c(NA, NaN), "-∞" = -Inf, "∞" = Inf)
tt(tmp) |> format_tt(replace = dict, digits = 2)
```

| x | y |
|---|-----|
| 1 | 3.1 |
| 2 | - |
| 3 | - |
| 4 | -∞ |
| 5 | ∞ |

2.5 Escape special characters

LaTeX and HTML use special characters to indicate strings which should be interpreted rather than displayed as text. For example, including underscores or dollar signs in LaTeX can cause compilation errors in some documents. To display those special characters, we need to substitute or escape them with backslashes, depending on the output format. The `escape` argument of `format_tt()` can be used to do this automatically:

```
dat <- data.frame(
  "LaTeX" = c("Dollars $", "Percent %", "Underscore _", "Backslash \\"),
  "HTML" = c("<br>", "<sup>4</sup>", "<emph>blah</emph>", "&amp;"),
  "Typst" = c("Dollars $", "Percent %", "Underscore _", "Backslash \\")
)

tt(dat) |> format_tt(escape = TRUE)
```

| LaTeX | HTML | Typst |
|--------------|-------------------|--------------|
| Dollars \$ |
 | Dollars \$ |
| Percent % | ⁴ | Percent % |
| Underscore _ | <emph>blah</emph> | Underscore _ |
| Backslash \ | & | Backslash \ |

When applied to a `tt()` table, `format_tt()` will determine the type of escaping to do automatically. When applied to a string or vector, we must specify the type of escaping to apply:

```
format_tt("_ Dollars $", escape = "latex")
```

```
[1] "\\_ Dollars \\$"
```

2.6 Line breaks

LaTeX, Typst, and HTML use different character sequences to indicate line breaks. We can create a single table that works in all three formats by using the `linebreak` argument of `format_tt()`. The idea is to choose one specific character sequence to represent line breaks, and to supply it to the `linebreak` argument.

In this example, we use `
` to represent line breaks in our data. This is the standard approach in HTML, and we rely on `format_tt()` substitute it to an appropriate string in other formats:

```
d <- data.frame(Text = "First line<br>Second line")
tt(d, width = .4) |> format_tt(linebreak = "<br>")
```

| Text |
|---------------------------|
| First line
Second line |

The `linebreak` argument automatically converts your specified string to:

- HTML: `
`
- LaTeX: `\\`
- Typst: `\`
- Markdown: No conversion (preserves original string)

2.7 Markdown

Markdown can be rendered in cells by using the `markdown` argument of the `format_tt()` function (note: this requires installing the `markdown` as an optional dependency).

```
dat <- data.frame(markdown = c(
  "This is _italic_ text.",
  "This sentence ends with a superscript.2"
))

tt(dat) |>
  format_tt(j = 1, markdown = TRUE) |>
  style_tt(j = 1, align = "c")
```

| markdown |
|---|
| This is <i>italic</i> text. |
| This sentence ends with a superscript. ² |

Markdown syntax can be particularly useful when formatting URLs in a table:

```
dat <- data.frame(
  `Package (link)` = c(
    "[`marginaleffects`](https://www.marginaleffects.com/)",
    "[`modelsummary`](https://www.modelsummary.com/)",
    "[`tinytable`](https://vincentarelbundock.github.io/tinytable/)",
    "[`countrycode`](https://vincentarelbundock.github.io/countrycode/)",
    "[`WDI`](https://vincentarelbundock.github.io/WDI/)",
    "[`softbib`](https://vincentarelbundock.github.io/softbib/)",
    "[`tinysnapshot`](https://vincentarelbundock.github.io/tinysnapshot/)",
    "[`altdoc`](https://etiennebacher.github.io/altdoc/)",
    "[`tinyplot`](https://grantmcdermott.com/tinyplot/)",
    "[`parameters`](https://easystats.github.io/parameters/)",
    "[`insight`](https://easystats.github.io/insight/)"
  ),
  Purpose = c(
    "Interpreting statistical models",
    "Data and model summaries",
    "Draw beautiful tables easily",
    "Convert country codes and names",
    "Download data from the World Bank",
  )
)
```



```

    "Software bibliographies in R",
    "Snapshots for unit tests using `tinytest`",
    "Create documentation website for R packages",
    "Extension of base R plot functions",
    "Extract from model objects",
    "Extract information from model objects"
  ),
  check.names = FALSE
)

tt(dat) |> format_tt(j = 1, markdown = TRUE)

```

Table 7: Vincent sometimes contributes to these R packages.

| Package (link) | Purpose |
|---------------------------------|---|
| marginaleffects | Interpreting statistical models |
| modelsummary | Data and model summaries |
| tinytable | Draw beautiful tables easily |
| countrycode | Convert country codes and names |
| WDI | Download data from the World Bank |
| softbib | Software bibliographies in R |
| tinysnapshot | Snapshots for unit tests using ‘tinytest’ |
| altdoc | Create documentation website for R packages |
| tinyplot | Extension of base R plot functions |
| parameters | Extract from model objects |
| insight | Extract information from model objects |

2.8 Custom functions

On top of the built-in features of `format_tt`, a custom formatting function can be specified via the `fn` argument. The `fn` argument takes a function that accepts a single vector and returns a string (or something that coerces to a string like a number).

```

tt(x) |>
  format_tt(j = "mpg", fn = function(x) paste(x, "mi/gal")) |>
  format_tt(j = "drat", fn = \(x) signif(x, 2))

```

| mpg | cyl | disp | hp | drat |
|-------------|-----|------|-----|------|
| 21 mi/gal | 6 | 160 | 110 | 3.9 |
| 21 mi/gal | 6 | 160 | 110 | 3.9 |
| 22.8 mi/gal | 4 | 108 | 93 | 3.8 |
| 21.4 mi/gal | 6 | 258 | 110 | 3.1 |

For example, the [scales package](#) which is used internally by `ggplot2` provides a bunch of useful tools for formatting (e.g. dates, numbers, percents, logs, currencies, etc.). The `label_*()` functions can be passed to the `fn` argument.

Note that we call `format_tt(escape = TRUE)` at the end of the pipeline because the column names and cells include characters that need to be escaped in LaTeX: `_`, `%`, and `$`. This last call is superfluous in HTML.

```
thumbdrives <- data.frame(
  date_lookup = as.Date(c("2024-01-15", "2024-01-18", "2024-01-14", "2024-01-16")),
  price = c(18.49, 19.99, 24.99, 24.99),
  price_rank = c(1, 2, 3, 3),
  memory = c(16e9, 12e9, 10e9, 8e9),
  speed_benchmark = c(0.6, 0.73, 0.82, 0.99)
)

tt(thumbdrives) |>
  format_tt(j = 1, fn = scales::label_date("%B %d %Y")) |>
  format_tt(j = 2, fn = scales::label_currency()) |>
  format_tt(j = 3, fn = scales::label_ordinal()) |>
  format_tt(j = 4, fn = scales::label_bytes()) |>
  format_tt(j = 5, fn = scales::label_percent()) |>
  format_tt(escape = TRUE)
```

| date_lookup | price | price_rank | memory | speed_benchmark |
|-----------------|---------|------------|--------|-----------------|
| January 15 2024 | \$18.49 | 1st | 16 GB | 60% |
| January 18 2024 | \$19.99 | 2nd | 12 GB | 73% |
| January 14 2024 | \$24.99 | 3rd | 10 GB | 82% |
| January 16 2024 | \$24.99 | 3rd | 8 GB | 99% |

2.9 Captions, notes, groups, and column names

The `format_tt()` function can also be used to format captions, notes, and column names.

```
tab <- data.frame(
  "A_B" = rnorm(5),
  "B_C" = rnorm(5),
  "C_D" = rnorm(5))

tt(tab, digits = 2, notes = "_Source_: Simulated data.") |>
  group_tt(i = list("Down" = 1, "Up" = 3)) |>
  format_tt("colnames", fn = \(x) sub("_", " / ", x)) |>
  format_tt("notes", markdown = TRUE) |>
  format_tt("groupi", replace = list("↓" = "Down", "↑" = "Up"))
```

| A / B | B / C | C / D |
|--------------------------------|-------|-------|
| ↓ | | |
| -0.47 | 1.98 | -0.9 |
| 1.71 | -0.44 | 0.41 |
| ↑ | | |
| -0.2 | 0.61 | -1.54 |
| 0.77 | 0.4 | -0.17 |
| 0.92 | -0.59 | -0.53 |
| <i>Source: Simulated data.</i> | | |

3 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 `inner` and `outer` arguments of `theme_latex()`.

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.

```
library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")
x <- mtcars[1:4, 1:5]
```

3.1 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 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 | 6 | 160 | 110 | 3.9 |
| <i>21</i> | 6 | <i>160</i> | <i>110</i> | 3.9 |
| <i>22.8</i> | 4 | <i>108</i> | <i>93</i> | 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 |
|-------------|----------|------------|------------|-------------|
| <i>21</i> | <i>6</i> | <i>160</i> | <i>110</i> | <i>3.9</i> |
| <i>21</i> | <i>6</i> | <i>160</i> | <i>110</i> | <i>3.9</i> |
| <i>22.8</i> | <i>4</i> | <i>108</i> | <i>93</i> | <i>3.85</i> |
| <i>21.4</i> | <i>6</i> | <i>258</i> | <i>110</i> | <i>3.08</i> |

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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

The `j` argument accepts integer vectors, character 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 = c("mpg", "drat"), color = "orange")
```

| mpg | cyl | disp | hp | drat |
|------------|-----|------|-----|-------------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

Here we use a “negative lookahead” to exclude certain columns:

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

The `i` argument also accepts unquoted expressions for non-standard evaluation. This allows us to style rows based on data conditions:

```
tt(x) |>
  style_tt(i = mpg > 21, background = "lightblue", bold = TRUE)
```

| mpg | cyl | disp | hp | drat |
|-------------|----------|------------|------------|-------------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

There is also a `groupi` object with indices that can be manipulated as an unquoted numeric expression.

```
tt(head(mtcars, 10)) |>
  group_tt(i = list("Hello" = 3, "World" = 5)) |>
  group_tt(j = list("Cyl" = 1:3, "Disp" = 4:6)) |>
  style_tt(groupi, background = "pink", align = "c") |>
  style_tt(groupi + 1, color = "white", background = "teal")
```

| Cyl | | | Disp | | | | | | | |
|-------|-----|------|------|------|------|------|----|----|------|------|
| mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 | 1 | 4 | 4 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 0 | 1 | 4 | 4 |
| Hello | | | | | | | | | | |
| 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.6 | 1 | 1 | 4 | 1 |
| 21.4 | 6 | 258 | 110 | 3.08 | 3.21 | 19.4 | 1 | 0 | 3 | 1 |
| World | | | | | | | | | | |
| 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17 | 0 | 0 | 3 | 2 |
| 18.1 | 6 | 225 | 105 | 2.76 | 3.46 | 20.2 | 1 | 0 | 3 | 1 |
| 14.3 | 8 | 360 | 245 | 3.21 | 3.57 | 15.8 | 0 | 0 | 3 | 4 |
| 24.4 | 4 | 147 | 62 | 3.69 | 3.19 | 20 | 1 | 0 | 4 | 2 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 | 0 | 4 | 2 |
| 19.2 | 6 | 168 | 123 | 3.92 | 3.44 | 18.3 | 1 | 0 | 4 | 4 |

3.2 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.
- 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 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.

3.3 Alignment

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

```
dat <- data.frame(
  a = c("a", "aaa", "aaaaa"),
  b = c("b", "bbb", "bbbbb"),
  c = c("c", "ccc", "ccccc")
)
```

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

| a | b | c |
|-------|-------|-------|
| a | b | c |
| aaa | bbb | ccc |
| aaaaa | bbbbb | ccccc |

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

| a | b | c |
|-------|-------|-------|
| a | b | c |
| aaa | bbb | ccc |
| aaaaa | bbbbb | ccccc |

In LaTeX documents (only), we can use decimal-alignment:

```
z <- data.frame(pi = c(pi * 100, pi * 1000, pi * 10000, pi * 100000))
tt(z) |>
  format_tt(j = 1, digits = 8, num_fmt = "significant_cell") |>
  style_tt(j = 1, align = "d")
```

| pi |
|------------|
| 314.159 27 |
| 3141.5927 |
| 31 415.927 |
| 314 159.27 |

3.4 Font size

The font size is specified in em units.

```
tt(x) |> style_tt(i = 1:4, j = "mpg|hp|qsec", fontsize = 1.5)
```

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

3.5 Spanning cells (merging cells)

Sometimes, it can be useful to make a cell stretch across multiple columns or rows, 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 and two rows:

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

| mpg | cyl | disp | hp | drat |
|------|----------|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | | | 3.9 |
| 22.8 | | | | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

Here is the original table for comparison:

```
tt(x)
```

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

Spanning cells can be particularly useful when we want to suppress redundant labels:

```
tab <- aggregate(mpg ~ cyl + am, FUN = mean, data = mtcars)
tab <- tab[order(tab$cyl, tab$am), ]
tab
```

```
      cyl am      mpg
1      4  0 22.90000
4      4  1 28.07500
2      6  0 19.12500
5      6  1 20.56667
3      8  0 15.05000
6      8  1 15.40000
```

```
tt(tab, digits = 2) |>
  style_tt(i = c(1, 3, 5), j = 1, rowspan = 2, alignv = "t")
```

| cyl | am | mpg |
|-----|----|-----|
| 4 | 0 | 23 |
| | 1 | 28 |
| 6 | 0 | 19 |
| | 1 | 21 |
| 8 | 0 | 15 |
| | 1 | 15 |

The `rowspan` feature is also useful to create multi-row labels. For example, in this table there is a linebreak, but all the text fits in a single cell:

```
tab <- data.frame(Letters = c("A<br>B", ""), Numbers = c("First", "Second"))

tt(tab) |>
  theme_html(class = "table-bordered")
```

| Letters | Numbers |
|---------|---------|
| A
B | First |
| | Second |

Now, we use `colspan` to ensure that that cells in the first column take up less space and are combined into one:

```
tt(tab) |>
  theme_html(class = "table-bordered") |>
  style_tt(1, 1, rowspan = 2)
```

| Letters | Numbers |
|---------|---------|
| A
B | First |
| | Second |

We can combine several spans to create complex tables like this one:

```
df <- structure(list(
  Col1 = c("Col Header", "Item 0", "Item 1", "Item 2", "Total"),
  Col2 = c("Span 1", "X", "xx", "xx", "xxxx"),
  Col2.1 = c("Span 1", "Y", "xx", "xx", "xxxx"),
  Col2.2 = c("Span 2", "X", "xx", "xx", "xxxx"),
  Col2.3 = c("Span 2", "Y", "xx", "xx", "xxxx")),
  class = "data.frame", row.names = c(NA, -5L))
df |>
  setNames(NULL) |>
  tt() |>
  style_tt(1, 1, rowspan = 2, bold = TRUE) |>
  style_tt(1, c(2, 4), colspan = 2, bold = TRUE) |>
  style_tt(5, c(2, 4), colspan = 2) |>
  theme_grid()
```

| Col Header | Span 1 | | Span 2 | |
|------------|--------|----|--------|----|
| | X | Y | X | Y |
| Item 1 | xx | xx | xx | xx |
| Item 2 | xx | xx | xx | xx |
| Total | xxxx | | xxxx | |

3.6 Headers

The header can be omitted from the table by using the `colnames` argument.

```
tt(x, colnames = FALSE)
```

| | | | | |
|------|---|-----|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

The first is row 0, and higher level headers (ex: column spanning labels) have negative indices like -1. They can be styled as expected:

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

When styling columns without specifying `i`, the headers are styled in accordance with the rest of the column:

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

3.7 Conditional styling

We can use the standard `which` function from Base R to create indices and apply conditional styling 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),
    j = "am|vs",
    background = "teal",
    color = "white"
  )
```

| mpg | am | vs |
|------|----|----|
| 21 | 1 | 0 |
| 21 | 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 |

We can also use non-standard evaluation to apply conditional styling directly with unquoted expressions:

```
tt(k) |>
  style_tt(i = mpg > 22, background = "lightgreen", bold = TRUE)
```

| mpg | am | vs |
|-------------|----------|----------|
| 21 | 1 | 0 |
| 21 | 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 |

Users can also supply a logical matrix of the same size as `x` to indicate which cell should be styled. For example, we can change the colors of certain entries in a correlation matrix as follows:

```
cormat <- data.frame(cor(mtcars[1:5]))
tt(cormat, digits = 2) |>
  style_tt(i = abs(cormat) > .8, background = "black", color = "white")
```

| mpg | cyl | disp | hp | drat |
|-------|-------|-------|-------|-------|
| 1 | -0.85 | -0.85 | -0.78 | 0.68 |
| -0.85 | 1 | 0.9 | 0.83 | -0.7 |
| -0.85 | 0.9 | 1 | 0.79 | -0.71 |
| -0.78 | 0.83 | 0.79 | 1 | -0.45 |
| 0.68 | -0.7 | -0.71 | -0.45 | 1 |

3.8 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 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 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

We can also produce more complex heatmap-like tables to illustrate different font sizes in em units:

```
# font sizes
fs <- seq(.1, 2, length.out = 20)

# headless table
k <- data.frame(matrix(fs, ncol = 5))
```

| | | | | |
|-----|-----|-----|-----|-----|
| 0 | 0.5 | 0.9 | 1.3 | 1.7 |
| 0.1 | 0.6 | 1 | 1.4 | 1.8 |
| 0.2 | 0.7 | 1.1 | 1.5 | 1.9 |
| 0.3 | 0.8 | 1.2 | 1.6 | 2 |

```
# colors
bg <- hcl.colors(20, "Inferno")
fg <- ifelse(as.matrix(k) < 1.7, tail(bg, 1), head(bg, 1))

# table
tt(k, width = .7, theme = "empty", colnames = FALSE) |>
  style_tt(j = 1:5, align = "ccccc", alignv = "m") |>
  style_tt(
    i = 1:4,
    j = 1:5,
    color = fg,
    background = bg,
    fontsize = fs
  )
```

3.9 Lines (borders)

The `style_tt` function allows us to customize the borders that surround each cell of a table, as well as horizontal and vertical rules. To control these lines, we use the `line`, `line_width`, and `line_color` arguments. Here's a brief overview of each of these arguments:

- **line:** This argument specifies where solid lines should be drawn. It is a string that can consist of the following characters:
 - "t": Draw a line at the top of the cell, row, or column.
 - "b": Draw a line at the bottom of the cell, row, or column.
 - "l": Draw a line at the left side of the cell, row, or column.
 - "r": Draw a line at the right side of the cell, row, or column.
 - You can combine these characters to draw lines on multiple sides, such as "tbl" to draw lines at the top, bottom, and left sides of a cell.
- **line_width:** This argument controls the width of the solid lines in em units (default: 0.1 em). You can adjust this value to make the lines thicker or thinner.

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

- `line_color`: Specifies the color of the solid lines. You can use color names, hexadecimal codes, or other color specifications to define the line color.

Here is an example where we draw lines around every border (“t”, “b”, “l”, and “r”) of specified cells.

```
tt(x, theme = "empty") |>
  style_tt(
    i = 0:3,
    j = 1:3,
    line = "tblr",
    line_width = 0.4,
    line_color = "orange"
  )
```

And here is an example with horizontal rules:

```
tt(x, theme = "empty") |>
  style_tt(i = 0, line = "t", line_color = "orange", line_width = 0.4) |>
  style_tt(i = 1, line = "t", line_color = "purple", line_width = 0.2) |>
  style_tt(i = 4, line = "b", line_color = "orange", line_width = 0.4)
```

| | | | |
|---|---|---|---|
| 1 | 3 | 5 | 7 |
| 2 | 4 | 6 | 8 |

```
dat <- data.frame(1:2, 3:4, 5:6, 7:8)

tt(dat, theme = "empty", colnames = FALSE) |>
  style_tt(
    line = "tblr", line_color = "white", line_width = 0.5,
    background = "blue", color = "white"
  )
```

3.10 Markdown

Markdown is a text-only format with limited styling options. The only supported arguments are: **bold**, *italic*, and **strikeout**. These limitations exist because there is no standard markdown syntax for other styling options (ex: colors and background).

However, in terminals (consoles) that support it, **tinytable** can display colors and text styles using ANSI escape codes by setting `theme_markdown(ansi = TRUE)`. This allows for rich formatting in compatible terminal environments.

Here's an example with multiple ANSI styles:

```
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 35),
  Score = c(95.5, 87.2, 92.8)
)

tt(data, caption = "Three friends.") |>
  style_tt(i = c(0, 3), color = "orange") |>
  style_tt(i = 1, background = "teal", color = "black", bold = TRUE) |>
  style_tt(i = 2, j = 2, underline = TRUE, color = "red") |>
  style_tt(i = 3, strikeout = TRUE) |>
  group_tt(j = list("Characteristics" = 2:3)) |>
  style_tt(i = "caption", bold = TRUE, color = "red") |>
  theme_markdown(ansi = TRUE)
```

| Characteristics | | |
|--------------------|-----------|-----------------|
| Name | Age | Score |
| Alice | 25 | 95.5 |
| Bob | <u>30</u> | 87.2 |
| Charlie | 35 | 92.8 |

Table: **Three friends.**

Figure 1: ANSI Terminal Output

4 Groups and labels

```
library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")
x <- mtcars[1:4, 1:5]
```

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

4.1 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 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 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 |
| 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 | 1 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 |

The numbers in the `i` list indicate that a label must be inserted at position `#` in the original table (without row groups). For example,

```
tt(head(iris)) |>
  group_tt(i = list("After 0" = 1, "After 3a" = 4, "After 3b" = 4, "After 5" = 6))
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--------------|-------------|--------------|-------------|---------|
| After 0 | | | | |
| 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 |
| After 3a | | | | |
| After 3b | | | | |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 3.6 | 1.4 | 0.2 | setosa |
| After 5 | | | | |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |

It is also possible to use unquoted expressions (non-standard evaluation) to specify row groups. For example,

```
tmp <- do.call(rbind, by(iris, ~Species, head, n = 2))
tt(tmp) |>
  group_tt(i = Species) |>
  subset(select = ~Species) |>
  style_tt(align = "c") |>
  style_tt(i = "groupi", align = "c", color = "teal", line = "b")
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|--------------|-------------|--------------|-------------|
| setosa | | | |
| 5.1 | 3.5 | 1.4 | 0.2 |
| 4.9 | 3 | 1.4 | 0.2 |
| versicolor | | | |
| 7 | 3.2 | 4.7 | 1.4 |
| 6.4 | 3.2 | 4.5 | 1.5 |
| virginica | | | |
| 6.3 | 3.3 | 6 | 2.5 |
| 5.8 | 2.7 | 5.1 | 1.9 |

4.1.1 Styling row groups

We can style group rows in the same way as regular rows (caveat: not in Word or Markdown):

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

tab |> style_tt(
  i = c(3, 5, 9),
  align = "c",
  color = "white",
  background = "gray",
  bold = TRUE)
```

| mpg | cyl | disp | hp | drat | wt | qsec | vs |
|--------------------------|-----|------|-----|------|------|------|----|
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 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 |
| 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 | 1 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 |

Calculating the location of rows can be cumbersome. Instead of doing this by hand, we can use the “groupi” shortcut to style rows and “~groupi” (the complement) to style all non-group rows.

```
tab |>
  style_tt("groupi", color = "white", background = "teal") |>
  style_tt("~groupi", j = 1, indent = 2)
```


| mpg | cyl | disp | hp | drat | wt | qsec | vs |
|--------------------------|-----|------|-----|------|------|------|----|
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 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 |
| 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 | 1 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 |

4.1.2 Automatic row groups

We can use the `group_tt()` function to group rows and label them using spanners (almost) automatically. For example,

```
# subset and sort data
df <- mtcars |>
  head(10) |>
  sort_by(~am)

# draw table
tt(df) |> group_tt(i = df$am)
```

| mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|------|-----|------|-----|------|------|------|----|----|------|------|
| 0 | | | | | | | | | | |
| 21.4 | 6 | 258 | 110 | 3.08 | 3.21 | 19.4 | 1 | 0 | 3 | 1 |
| 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17 | 0 | 0 | 3 | 2 |
| 18.1 | 6 | 225 | 105 | 2.76 | 3.46 | 20.2 | 1 | 0 | 3 | 1 |
| 14.3 | 8 | 360 | 245 | 3.21 | 3.57 | 15.8 | 0 | 0 | 3 | 4 |
| 24.4 | 4 | 147 | 62 | 3.69 | 3.19 | 20 | 1 | 0 | 4 | 2 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 | 0 | 4 | 2 |
| 19.2 | 6 | 168 | 123 | 3.92 | 3.44 | 18.3 | 1 | 0 | 4 | 4 |
| 1 | | | | | | | | | | |
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 | 1 | 4 | 4 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 0 | 1 | 4 | 4 |
| 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.6 | 1 | 1 | 4 | 1 |

4.1.3 Row matrix insertion

While the traditional `group_tt(i = list(...))` approach is useful for adding individual labeled rows, sometimes you need to insert multiple rows of data at specific positions. The matrix insertion feature provides a more efficient way to do this.

Instead of creating multiple named list entries, you can specify row positions as an integer vector in `i` and provide a character matrix in `j`. This is particularly useful when you want to insert the same content (like headers or separators) at multiple positions:

```
rowmat <- matrix(colnames(iris))

tt(head(iris, 7)) |>
  group_tt(i = c(2, 5), j = rowmat)
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--------------|-------------|--------------|-------------|---------|
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 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 |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 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 |

The matrix is expected to have the same number of columns as the table. However, if you provide a single-column matrix with a number of elements that is a multiple of the table's column count, it will be automatically reshaped to match the table structure. This makes it easy to provide data in a linear format:

```
rowmat <- matrix(c(
  "_", "_", "_", "_", "_",
  "/", "/", "/", "/", "/"))
tt(head(iris, 7)) |> group_tt(i = 2, j = rowmat)
```

| 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 |

We can also insert rows of the group matrix in different positions:

```
tt(head(iris, 7)) |> group_tt(i = c(1, 8), j = rowmat)
```

| 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 |
| / | / | / | / | / |

4.2 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 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 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 |
| 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 | 1 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 |

We can stack several extra headers on top of one another:

```
x <- mtcars[1:4, 1:5]
tt(x) |>
  group_tt(j = list("Foo" = 2:3, "Bar" = 5)) |>
  group_tt(j = list("Hello" = 1:2, "World" = 4:5))
```

| Hello | | | World | |
|-------|-----|------|-------|------|
| | | | Foo | |
| | | | Bar | |
| mpg | cyl | disp | hp | drat |
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

4.2.1 Styling column groups

To style column headers, we use zero or negative indices:

```
tt(x) |>
  group_tt(j = list("Foo" = 2:3, "Bar" = 5)) |>
  group_tt(j = list("Hello" = 1:2, "World" = 4:5)) |>
  style_tt(i = 0, color = "orange") |>
```

```
style_tt(i = -1, color = "teal") |>
style_tt(i = -2, color = "yellow")
```

| Hello | | | World | |
|-------|-----|------|-------|------|
| Foo | | | Bar | |
| mpg | cyl | disp | hp | drat |
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

Alternatively, we can use string shortcuts:

```
tt(x) |>
  group_tt(j = list("Foo" = 2:3, "Bar" = 5)) |>
  group_tt(j = list("Hello" = 1:2, "World" = 4:5)) |>
  style_tt("groupj", color = "orange") |>
  style_tt("colnames", color = "teal")
```

| Hello | | | World | |
|-------|-----|------|-------|------|
| Foo | | | Bar | |
| mpg | cyl | disp | hp | drat |
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

Here is a table with both row and column headers, as well as some styling:

```
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
  ),
  j = list(
    "Hamburgers" = 1:3,
    "Halloumi" = 4:5,
    "Tofu" = 7
  )
) |>
style_tt(
  i = c(3, 5, 9),
  align = "c",
  background = "teal",
  color = "white"
) |>
style_tt(i = -1, color = "teal")

```

| Hamburgers | | | Halloumi | | Tofu | | |
|--------------------------|-----|------|----------|------|------|------|----|
| mpg | cyl | disp | hp | drat | wt | qsec | vs |
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.5 | 0 |
| 21 | 6 | 160 | 110 | 3.9 | 2.88 | 17 | 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 |
| 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 | 1 |
| 22.8 | 4 | 141 | 95 | 3.92 | 3.15 | 22.9 | 1 |

4.2.2 Column names with delimiters

Group labels can be specified using column names with delimiters. For example, some of the columns in this data frame have group identifiers. Note that the first column does not have a group identifier, and that the last column has a group identifier but no column name.

```
dat <- data.frame(
  "A_D" = rnorm(3),
  "A_B_D" = rnorm(3),
  "A_B_" = rnorm(3),
  "_C_E" = rnorm(3),
  check.names = FALSE
)

tt(dat) |> group_tt(j = "_")
```

| A | | | |
|---------|-------|--------|---------|
| B | | | C |
| D | D | | E |
| -0.3755 | -1.29 | 0.292 | 2.2989 |
| -0.0728 | 1.04 | -2.04 | -0.3826 |
| -0.3532 | 1.33 | -0.311 | 0.0827 |

4.3 Case studies

4.3.1 Repeated column names

In some contexts, users wish to repeat the column names to treat them as group labels. Consider this dataset:

```
library(tinytable)
library(magrittr)

dat = data.frame(
  Region = as.character(state.region),
  State = row.names(state.x77),
  state.x77[, 1:3]) |>
  sort_by(~ Region + State) |>
  subset(Region %in% c("North Central", "Northeast"))
dat = do.call(rbind, by(dat, dat$Region, head, n = 3))
row.names(dat) = NULL
dat
```

| Region | State | Population | Income | Illiteracy |
|--------|-------|------------|--------|------------|
|--------|-------|------------|--------|------------|

| | | | | | |
|---|---------------|---------------|-------|------|-----|
| 1 | North Central | Illinois | 11197 | 5107 | 0.9 |
| 2 | North Central | Indiana | 5313 | 4458 | 0.7 |
| 3 | North Central | Iowa | 2861 | 4628 | 0.5 |
| 4 | Northeast | Connecticut | 3100 | 5348 | 1.1 |
| 5 | Northeast | Maine | 1058 | 3694 | 0.7 |
| 6 | Northeast | Massachusetts | 5814 | 4755 | 1.1 |

Here, we may want to repeat the column names for every region. The `group_tt()` function does not support this directly, but it is easy to achieve this effect by:

1. Insert column names as new rows in the data.
2. Create a row group variable (here: `region`)
3. Style the column names and group labels

Normally, we would call `style_tt(i = "groupi")` to style the row groups, but here we need the actual indices to also style one row below the groups. We can use the `@group_index_i` slot to get the indices of the row groups.

```
region_names <- unique(dat$Region)
region_indices <- rep(match(region_names, dat$Region), each = 2)

rowmat <- do.call(rbind, lapply(region_names, function(name) {
  rbind(
    c(name, rep("", 3)),
    colnames(dat)[2:5]
  )
}))

rowmat
```

| | [,1] | [,2] | [,3] | [,4] |
|------|-----------------|--------------|----------|--------------|
| [1,] | "North Central" | " | " | " |
| [2,] | "State" | "Population" | "Income" | "Illiteracy" |
| [3,] | "Northeast" | " | " | " |
| [4,] | "State" | "Population" | "Income" | "Illiteracy" |

```
odd <- function(x) x[seq(1, length(x), 2)]
even <- function(x) x[seq(2, length(x), 2)]

tt(dat[, 2:5], colnames = FALSE) |>
  group_tt(i = region_indices, j = rowmat) |>
  style_tt(even(groupi), bold = TRUE) |>
```

```
style_tt(odd(groupi), j = 1, align = "c", colspan = 4,
  background = "lightgrey")
```

| North Central | | | |
|---------------|------------|--------|------------|
| State | Population | Income | Illiteracy |
| Illinois | 11197 | 5107 | 0.9 |
| Indiana | 5313 | 4458 | 0.7 |
| Iowa | 2861 | 4628 | 0.5 |
| Northeast | | | |
| State | Population | Income | Illiteracy |
| Connecticut | 3100 | 5348 | 1.1 |
| Maine | 1058 | 3694 | 0.7 |
| Massachusetts | 5814 | 4755 | 1.1 |

5 Themes

`tinytable` offers a very flexible theming framework, which includes a few basic visual looks, as well as other functions to apply collections of transformations to `tinytable` objects in a repeatable way. These themes can be applied by supplying a string or function to the `theme` argument in `tt()`. Alternatively, users can call the specific theme functions like `theme_stripped()`, `theme_grid()`, etc.

The main difference between theme functions and the other options in package, is that whereas `style_tt()` and `format_tt()` aim to be output agnostic, theme functions supply transformations that can be output-specific, and which can have their own sets of distinct arguments. See below for a few examples.

```
library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")
x <- mtcars[1:4, 1:5]
```

5.1 Visual themes

To begin, let's explore a few of the basic looks supplied by themes:

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

```
tt(x) |> theme_stripped()
```

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

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

| mpg | cyl | disp | hp | drat |
|------|-----|------|-----|------|
| 21 | 6 | 160 | 110 | 3.9 |
| 21 | 6 | 160 | 110 | 3.9 |
| 22.8 | 4 | 108 | 93 | 3.85 |
| 21.4 | 6 | 258 | 110 | 3.08 |

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

5.2 Custom themes

Users can also define their own themes to apply consistent visual tweaks to tables. For example, this defines a themeing function and sets a global option to apply it to all tables consistently:¹

```
theme_vincent <- function(x, ...) {  
  out <- x |>  
  style_tt(color = "teal") |>  
  theme_default()  
  out@caption <- "Always use the same caption."  
  out@width <- .5  
  return(out)  
}  
  
options(tinytable_tt_theme = theme_vincent)  
  
tt(mtcars[1:2, 1:2])
```

Table 8: Always use the same caption.

| mpg | cyl |
|-----|-----|
| 21 | 6 |
| 21 | 6 |

```
tt(mtcars[1:3, 1:3])
```

Table 9: Always use the same caption.

| mpg | cyl | disp |
|------|-----|------|
| 21 | 6 | 160 |
| 21 | 6 | 160 |
| 22.8 | 4 | 108 |

```
options(tinytable_tt_theme = NULL)
```

Here is a slightly more complex example. The benefit of this approach is that we apply a function via the `style_tt()` function and its `finalize` argument, so we can leverage some of the object components that are only available at the printing stage:

¹Note: Captions must be defined in Quarto chunks for Typst output, which explains why they are not displayed in the Typst version of this document.

```

theme_slides <- function(x, ...) {
  fn <- function(table) {
    if (isTRUE(table@output == "typst")) {
      table@table_string <- paste0("#figure([\n", table@table_string, "\n])")
    }
    return(table)
  }
  x <- style_tt(x, finalize = fn)
  return(x)
}

tt(head(iris), theme = theme_slides)

```

Note: the code above is not evaluated because it only applies to Typst output.

5.3 Tabular (LaTeX and HTML)

The `tabular` theme is designed to provide a more “raw” table, without a floating table environment in LaTeX, and without CSS or Javascript in HTML.

```

tt(x) |>
  theme_latex(environment = "tabular", table = FALSE) |>
  print("latex")

```

```

\begin{table}[H]
\centering
\begin{tabular}{llllll}
mpg & cyl & disp & hp & drat & \\
21 & 6 & 160 & 110 & 3.9 & \\
21 & 6 & 160 & 110 & 3.9 & \\
22.8 & 4 & 108 & 93 & 3.85 & \\
21.4 & 6 & 258 & 110 & 3.08 & \\
\end{tabular}
\end{table}

```

5.4 Combining themes

Themes are just functions that apply a set of transformations to a `tinytable` object. This means that users can combine themes to create new looks. For example, when we call `tt(x)` without specifying the `theme` argument, the `theme_default()` is applied automatically.

```
x <- head(iris, 3)
tt(x)
```

| 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 |

If we add a call to `theme_stripped()`, we add grey background stripes, but keep the other default stylings (ex: top and bottom horizontal rules).

```
tt(x) |> theme_stripped()
```

| 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 |

Alternatively, we could use `theme_empty()` to remove the default theme, and `theme_stripped()` to get a very minimal look with just the stripes.

```
tt(x) |> theme_empty() |> theme_stripped()
```

| 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 |

Or use the `theme` argument to get the same effect.

```
tt(x, theme = "empty") |> theme_stripped()
```

| 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 |

5.5 User-written themes

This section provides a few user-written themes that can be used to extend the functionality of `tinytable`. These themes are not included in the package by default, but they can be easily added to your workflow. If you would like your own custom theme to appear here, please open an issue on the [tinytable GitHub repository](#) or submit a pull request.

5.5.1 `theme_mitex()`

This theme was written by Kazuharu Yanagimoto. Thanks for your contribution!

The [MiTeX project](#) aims to bring LaTeX support to Typst documents. This theme replace every instance of matching pairs of dollars signs `$.$.` by a MiTeX function call: `#mitex(...)`. This allows you to use LaTeX math in Typst documents.

Warning: The substitution code is very simple and it may not work properly when there are unmatched `$` symbols in the document.

```
theme_mitex <- function(x, ...) {  
  fn <- function(table) {  
    if (isTRUE(table@output == "typst")) {  
      table@table_string <- gsub(  
        "\\$(.*?)\\$",  
        "#mitex(`\\1`)",  
        table@table_string)  
    }  
    return(table)  
  }  
  x <- style_tt(x, finalize = fn)  
  return(x)  
}
```

6 Plots and images



The `plot_tt()` function can embed images and plots in a `tinytable`. We can insert images by specifying their paths and positions (`i/j`).

```
library(tinytable)  
options(tinytable_tt_digits = 3)  
options(tinytable_latex_placement = "H")  
x <- mtcars[1:4, 1:5]
```

6.1 Inserting images in tables

To insert images in a table, we use the `plot_tt()` function. The `path_img` values must be relative to the main document saved by `save_tt()` or to the Quarto (or Rmarkdown) document in which the code is executed.

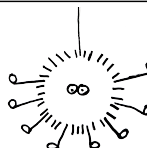
```
dat <- data.frame(  
  Species = c("Spider", "Squirrel"),  
  Image = ""  
)  
  
img <- c(  
  "figures/spider.png",  
  "figures/squirrel.png"  
)  
  
tt(dat) |>  
  plot_tt(j = 2, images = img, height = 3)
```

| Species | Image |
|----------|---|
| Spider |  |
| Squirrel |  |

In HTML tables, it is possible to insert tables directly from a web address, but not in LaTeX.

We can also combine text and images using the `sprintf` argument and `%s` placeholder:

```
tt(head(iris)) |>  
  plot_tt(1, 1,  
    images = "figures/spider.png",  
    sprintf = "Boris: %s",  
    height = 5) |>  
  style_tt(i = 1, j = 1, alignv = "m")
```


| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|-------------|--------------|-------------|---------|
|  | | | | |
| Boris: | | | | |
| 4.9 | 3.5 | 1.4 | 0.2 | setosa |
| 4.7 | 3 | 1.4 | 0.2 | setosa |
| 4.6 | 3.2 | 1.3 | 0.2 | setosa |
| 5 | 3.1 | 1.5 | 0.2 | setosa |
| 5.4 | 3.6 | 1.4 | 0.2 | setosa |
| | 3.9 | 1.7 | 0.4 | setosa |

6.2 Inline plots

We can draw inline plots three ways, with

1. Built-in templates for histograms, density plots, and bar plots
2. Custom plots using base R plots.
3. Custom plots using `ggplot2`.

To draw custom plots, one simply has to define a custom function, whose structure we illustrate below.

6.2.1 Built-in plots

There are several types of inline plots available by default.









6.2.1.1 Bar plots

Bar plots can be created with single or dual colors. With two colors, the first color is the bar and the second is the background:

```
dat <- data.frame(
  Metric = c("Sales", "Conversion", "Growth", "Efficiency"),
  Value = c(75, 45, 92, 38),
  Percentage = c(0.75, 0.45, 0.92, 0.38)
)

tt(dat) |>
  plot_tt(j = 2, fun = "bar", data = as.list(dat$Value), color = "darkorange") |>
```

```
plot_tt(j = 3, fun = "bar", data = as.list(dat$Percentage),
        color = c("steelblue", "lightgrey"), xlim = c(0, 1))
```

| Metric | Value | Percentage |
|------------|---|---|
| Sales |  |  |
| Conversion |  |  |
| Growth |  |  |
| Efficiency |  |  |










6.2.1.2 Other plot types

```
plot_data <- list(mtcars$mpg, mtcars$hp, mtcars$qsec)

dat <- data.frame(
  Variables = c("mpg", "hp", "qsec"),
  Histogram = "",
  Density = "",
  Line = ""
)

# random data for sparklines
lines <- lapply(1:3, \(x) data.frame(x = 1:10, y = rnorm(10)))

tt(dat) |>
  plot_tt(j = 2, fun = "histogram", data = plot_data) |>
  plot_tt(j = 3, fun = "density", data = plot_data, color = "darkgreen") |>
  plot_tt(j = 4, fun = "line", data = lines, color = "blue") |>
  style_tt(j = 2:4, align = "c")
```

| Variables | Histogram | Density | Line |
|-----------|---|---|--|
| mpg |  |  |  |
| hp |  |  |  |
| qsec |  |  |  |




6.2.2 Custom plots: Base R

Important: Custom functions must have ... as an argument.

To create a custom inline plot using Base R plotting functions, we create a function that returns another function. `tinytable` will then call that second function internally to generate the plot.

This is easier than it sounds! For example:

```
f <- function(d, ...) {  
  function() hist(d, axes = FALSE, ann = FALSE, col = "lightblue")  
}  
  
plot_data <- list(mtcars$mpg, mtcars$hp, mtcars$qsec)  
  
dat <- data.frame(Variables = c("mpg", "hp", "qsec"), Histogram = "")  
  
tt(dat) |>  
  plot_tt(j = 2, fun = f, data = plot_data)
```

| Variables | Histogram |
|-----------|--|
| mpg |  |
| hp |  |
| qsec |  |

6.2.3 Custom plots: ggplot2




Important: Custom functions must have `...` as an argument.

To create a custom inline plot using `ggplot2`, we create a function that returns a `ggplot` object:

```
library(ggplot2)  
  
f <- function(d, color = "black", ...) {  
  d <- data.frame(x = d)  
  ggplot(d, aes(x = x)) +  
    geom_histogram(bins = 30, color = color, fill = color) +  
    scale_x_continuous(expand = c(0, 0)) +  
    scale_y_continuous(expand = c(0, 0)) +  
    theme_void()  
}
```

```
plot_data <- list(mtcars$mpg, mtcars$hp, mtcars$qsec)

tt(dat) |>
  plot_tt(j = 2, fun = f, data = plot_data, color = "pink")
```

| Variables | Histogram |
|-----------|---|
| mpg |  |
| hp |  |
| qsec |  |

We can insert arbitrarily complex plots by customizing the `ggplot2` call:

```
penguins <- read.csv(
  "https://vincentarelbundock.github.io/Rdatasets/csv/palmerpenguins/penguins.csv",
  na.strings = ""
) |> na.omit()

# split data by species
dat <- split(penguins, penguins$species)
body <- lapply(dat, \(x) x$body_mass_g)
flip <- lapply(dat, \(x) x$flipper_length_mm)

# create nearly empty table
tab <- data.frame(
  "Species" = names(dat),
  "Body Mass" = "",
  "Flipper Length" = "",
  "Body vs. Flipper" = "",
  check.names = FALSE
)










# custom ggplot2 function to create inline plot
f <- function(d, ...) {
  ggplot(d, aes(x = flipper_length_mm, y = body_mass_g, color = sex)) +
    geom_point(size = 2) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_y_continuous(expand = c(0, 0)) +
    scale_color_manual(values = c("#E69F00", "#56B4E9")) +
    theme_void() +
```

```

    theme(legend.position = "none")
}

# `tinytable` calls
tt(tab) |>
  plot_tt(j = 2, fun = "histogram", data = body, height = 2) |>
  plot_tt(j = 3, fun = "density", data = flip, height = 2) |>
  plot_tt(j = 4, fun = f, data = dat, height = 2) |>
  style_tt(algnav = "m") |>
  style_tt(j = 2:4, align = "c")

```

| Species | Body Mass | Flipper Length | Body vs. Flipper |
|-----------|---|---|---|
| Adelie |  |  |  |
| Chinstrap |  |  |  |
| Gentoo |  |  |  |

7 LaTeX / PDF customization

```

library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")
x <- mtcars[1:4, 1:5]

```

7.1 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{tabulararray}
\usepackage{float}
\usepackage{graphicx}
\usepackage{rotating}
\usepackage[normalem]{ulem}
\NewTableCommand{\tinytableDefineColor}[3]{\definecolor{#1}{#2}{#3}}
\newcommand{\tinytableTabulararrayUnderline}[1]{\underline{#1}}
\newcommand{\tinytableTabulararrayStrikeout}[1]{\sout{#1}}

```

7.2 Introduction to tabulararray

`tabulararray` 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 `Tabulararray` 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 `tabulararray` documentation is fantastic. It will teach you how to customize virtually every aspect of your tables: <https://ctan.org/pkg/tabulararray?lang=en>

`Tabulararray` 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 `theme_latex()` function includes `inner` and `outer` arguments to set these respective features.

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

```

\begin{table}
\centering
\begin{tblr}[           %% tabulararray outer open
]                       %% tabulararray outer close
{                       %% tabulararray 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 `inner` argument of the `theme_latex()` function:

```

## / tbl-cap: "\\LaTeX{} table with colors and a spanning cell."
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(theme = "empty") |>
  theme_latex(inner = inner)

```

| mpg | cyl | disp | hp |
|------|-----|------|-----|
| 21 | 6 | | 110 |
| 21 | | | 110 |
| 22.8 | | | 93 |
| 21.4 | | | 110 |
| 18.7 | 8 | 360 | 175 |

7.3 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 |

| 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 |
| 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 |
| valign | vertical alignment: t (top), m (middle), b (bottom), h (head) or f (foot) | t |

| Key | Description and Values | Initial Value |
|-----------------------|---|---------------|
| <code>wd</code> | width dimension | None |
| <code>co</code> | coefficient for the extendable column (X column) | None |
| <code>bg</code> | background color name | None |
| <code>fg</code> | foreground color name | None |
| <code>font</code> | font commands | None |
| <code>mode</code> | set mode for column cells: <code>math</code> , <code>imath</code> , <code>dmath</code> or <code>text</code> | None |
| <code>cmd</code> | execute command for every cell text | None |
| <code>leftsep</code> | set horizontal space to the left of the column | 6pt |
| <code>rightsep</code> | set horizontal space to the right of the column | 6pt |
| <code>colsep</code> | set horizontal space to both sides of the column | 6pt |
| <code>preto</code> | prepend text to every cell (like <code>></code> specifier in <code>colspec</code>) | None |
| <code>appto</code> | append text to every cell (like <code><</code> specifier in <code>colspec</code>) | None |

hlines:

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

vlines:

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

8 Tips and Tricks

8.1 LaTeX

8.1.1 Preamble

`tinytable` uses the `tabulararray` package from your LaTeX distribution to draw tables. `tabulararray`, in turn, provides special `tblr`, `talltblr`, and `longtblr` environments to display tabular data.

When rendering a document from Quarto or Rmarkdown directly to PDF, `tinytable` will populate the LaTeX preamble automatically with all the required packages (except when code chunks are cached). For standalone LaTeX documents, these commands should be inserted in the preamble manually:

```
\usepackage{tabulararray}
\usepackage{float}
\usepackage{graphicx}
\usepackage{rotating}
\usepackage[normalem]{ulem}
\UseTblrLibrary{siunitx}
\newcommand{\tinytableTabulararrayUnderline}[1]{\underline{#1}}
\newcommand{\tinytableTabulararrayStrikeout}[1]{\sout{#1}}
\NewTableCommand{\tinytableDefineColor}[3]{\definecolor{#1}{#2}{#3}}
```

8.1.2 setspace

Some users have encountered unexpected spacing behavior when generating tables that are *not* wrapped in a `\begin{table}` environment (ex: `multipage` or raw `tblr`).

One issue stems from the fact that the `\begin{table}` environment resets any spacing commands in the preamble or body by default, such as:

```
\usepackage{setspace}
\doublespacing
```

This means that when using `theme_latex(environment="tabular")` —which does not wrap the table in a `table` environment— the spacing is *not* reset, and tables are double spaced. This is not a bug, since double-spacing is in fact what the user requested. Nevertheless, the behavior can seem surprising for those used to the automagical `table` environment spacing reset.

One workaround is to add the following to the document preamble when using multi-page/longtblr:

```
\usepackage{etoolbox}
\AtBeginEnvironment{longtblr}{\begin{singlespacing}}
\AtEndEnvironment{longtblr}{\end{singlespacing}}
```

Example Quarto doc:

```
---
title: longtblr and setspacing
format:
  pdf:
    include-in-header:
      - text: |
          % Tinytable preamble
          \usepackage{tabularray}
          \usepackage{float}
          \usepackage{graphicx}
          \usepackage{codehigh}
          \usepackage[normalem]{ulem}
          \UseTblrLibrary{booktabs}
          \newcommand{\tinytableTabularrayUnderline}[1]{\underline{#1}}
          \newcommand{\tinytableTabularrayStrikeout}[1]{\sout{#1}}
          \NewTableCommand{\tinytableDefineColor}[3]{\definecolor{#1}{#2}{#3}}
          % Spacing Commands
          \usepackage{setspace}
          \doublespacing
          % Fix Spacing in longtblr
          \usepackage{etoolbox}
          \AtBeginEnvironment{longtblr}{\begin{singlespacing}}
          \AtEndEnvironment{longtblr}{\end{singlespacing}}
---

```\{=latex}
\begin{longtblr}[%% tabularray outer open
] %% tabularray outer close
{ %% tabularray inner open
colspec={Q[]Q[]Q[]Q[]},
} %% tabularray inner close
```

```

\toprule
foo & bar & baz \\
foo & bar & baz \\
foo & bar & baz \\
\bottomrule
\end{longtblr}
...

```

### 8.1.3 Multi-line cells with minipage

In some contexts, users may want create cells with LaTeX or markdown code that spans multiple lines. This usually works well for HTML tables. But sometimes, in LaTeX, multi-line content with special environments must be wrapped in a `minipage` environment.

In the example that follows, we create a Markdown list using asterisks. Then, we call `litedown::mark()` to render that list as bullet points (an `itemize` environment in LaTeX). Finally, we define a custom function called `minipage` to wrap the bullet point in an environment.

```

library(tinytable)
library(litedown)

dat <- data.frame(
 A = c("Blah *blah* blah", "- Thing 1\n- Thing 2"),
 B = c("6%", "$5.29")
)

minipage <- function(x) {
 sprintf("\\begin{minipage}{\\textwidth}%s\\end{minipage}", mark(x, "latex"))
}

tab <- tt(dat, width = c(0.3, 0.2)) |>
 style_tt(j = 2, align = "c") |>
 format_tt(escape = TRUE) |>
 format_tt(j = 1, fn = mark, output = "html") |>
 format_tt(j = 1, fn = minipage, output = "latex")

tab

```

A	B
Blah <i>blah</i> blah	
<ul style="list-style-type: none"> <li>• Thing 1</li> <li>• Thing 2</li> </ul>	6%
Blah <i>blah</i> blah	
<ul style="list-style-type: none"> <li>• Thing 1</li> <li>• Thing 2</li> </ul>	\$5.29

### 8.1.4 Global styles

`tabularray` allows very powerful styling and themeing options. See [the reference manual](#) for more information.

For example, you can change the size of footnotes in all tables of a document with:

```

format:
 pdf:
 keep-tex: true
 header-includes: |
 \SetTblrStyle{foot}{font=\LARGE}

```{r}
library(tinytable)
library(magrittr)
tt(head(iris), notes = "Blah blah")
```

```

### 8.1.5 Beamer

Due to [a bug in the upstream package `rmarkdown`](#), Quarto or Rmarkdown presentations compiled to Beamer cannot include adequate package loading commands in the preamble automatically. This bug prevents `tinytable::usepackage_latex()` from modifying the preamble. Here's a workaround.

Save this LaTeX code as `preamble.tex`:

```

\RequirePackage{tabulararray}
\RequirePackage{booktabs}
\RequirePackage{float}
\usepackage[normalem]{ulem}
\usepackage{graphicx}
\UseTblrLibrary{booktabs}
\UseTblrLibrary{siunitx}
\NewTableCommand{\tinytableDefineColor}[3]{\definecolor{#1}{#2}{#3}}
\newcommand{\tinytableTabulararrayUnderline}[1]{\underline{#1}}
\newcommand{\tinytableTabulararrayStrikeout}[1]{\sout{#1}}

```

Then, load `preamble.tex` in your YAML header:

```

output:
 beamer_presentation:
 includes:
 in_header: preamble.tex

```

With these changes, the table should appear with colors as expected.

### 8.1.6 Label and caption position

In LaTeX, we can use `tabulararray` options in the preamble or the table to change the location of the label and caption. The example below shows a Quarto document with caption at the bottom.

```

output:
 pdf_document:
header-includes:
 - \usepackage{tabulararray}

```\symbreak
\DefTblrTemplate{firsthead,middlehead,lasthead}{default}{}
\DefTblrTemplate{firstfoot,middlefoot}{default}{}
\DefTblrTemplate{lastfoot}{default}%
{
  \UseTblrTemplate{caption}{default}
}

```

```

}
...

```{r, echo=FALSE}
library(modelsummary)
library(tinytable)
mod <- list()
mod[['One variable']] <- lm(mpg ~ hp, mtcars)
mod[['Two variables']] <- lm(mpg ~ hp + drat, mtcars)

modelsummary(mod,
 title = "Regression Models")|>
 theme_latex(outer = "label={tblr:test}")
...

Table \ref{tblr:test}

```

### 8.1.7 Placement

The `theme_latex()` function includes placement control over the positioning of the table in LaTeX documents, using floating parameters like `H` (from the `float` LaTeX package) to specify where the table should appear.

```

options(tinytable_latex_placement = NULL)
x <- head(mtcars)
tt(x) |>
 theme_latex(placement = "H") |>
 print(output = "latex")

```

```

\begin{table}[H]
\centering
\begin{tblr}[
%% tabularray outer open
]
%% tabularray outer close
{
%% tabularray inner open
colspec={Q[]Q[]Q[]Q[]Q[]Q[]Q[]Q[]Q[]Q[]},
hline{2}={1-11}{solid, black, 0.05em},
hline{1}={1-11}{solid, black, 0.1em},
hline{8}={1-11}{solid, black, 0.1em},
}
%% tabularray inner close
mpg & cyl & disp & hp & drat & wt & qsec & vs & am & gear & carb \\
21 & 6 & 160 & 110 & 3.9 & 2.62 & 16.5 & 0 & 1 & 4 & 4 \\

```



Table 17: Rotated table.						
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		

```

21 & 6 & 160 & 110 & 3.9 & 2.88 & 17 & 0 & 1 & 4 & 4 & \\
22.8 & 4 & 108 & 93 & 3.85 & 2.32 & 18.6 & 1 & 1 & 4 & 1 & \\
21.4 & 6 & 258 & 110 & 3.08 & 3.21 & 19.4 & 1 & 0 & 3 & 1 & \\
18.7 & 8 & 360 & 175 & 3.15 & 3.44 & 17 & 0 & 0 & 3 & 2 & \\
18.1 & 6 & 225 & 105 & 2.76 & 3.46 & 20.2 & 1 & 0 & 3 & 1 & \\
\end{tblr}
\end{table}

```

### 8.1.8 Rotate

```

tt(head(iris), caption = "Rotated table.") |>
 theme_rotate(angle = 45)

```

### 8.1.9 Multipage

The `multipage` theme is designed for LaTeX documents to allow long tables to continue across multiple pages. This theme ensures that tables are not truncated and that all data is presented clearly.

```
tmp <- rbind(mtcars, mtcars)[, 1:6]

cap <- "A long 80\\% width table with repeating headers."

tt(tmp, width = .8, caption = cap) |>
 theme_latex(multipage = TRUE, rowhead = 1)
```

Table 18: A long 80% width table with repeating headers.

mpg	cyl	disp	hp	drat	wt
21	6	160	110	3.9	2.62
21	6	160	110	3.9	2.88
22.8	4	108	93	3.85	2.32
21.4	6	258	110	3.08	3.21
18.7	8	360	175	3.15	3.44
18.1	6	225	105	2.76	3.46
14.3	8	360	245	3.21	3.57
24.4	4	146.7	62	3.69	3.19
22.8	4	140.8	95	3.92	3.15
19.2	6	167.6	123	3.92	3.44
17.8	6	167.6	123	3.92	3.44
16.4	8	275.8	180	3.07	4.07
17.3	8	275.8	180	3.07	3.73
15.2	8	275.8	180	3.07	3.78
10.4	8	472	205	2.93	5.25
10.4	8	460	215	3	5.42
14.7	8	440	230	3.23	5.34
32.4	4	78.7	66	4.08	2.2
30.4	4	75.7	52	4.93	1.61

Continued on next page

Table 18: A long 80% width table with repeating headers. (Continued)

mpg	cyl	disp	hp	drat	wt
33.9	4	71.1	65	4.22	1.83
21.5	4	120.1	97	3.7	2.46
15.5	8	318	150	2.76	3.52
15.2	8	304	150	3.15	3.44
13.3	8	350	245	3.73	3.84
19.2	8	400	175	3.08	3.85
27.3	4	79	66	4.08	1.94
26	4	120.3	91	4.43	2.14
30.4	4	95.1	113	3.77	1.51
15.8	8	351	264	4.22	3.17
19.7	6	145	175	3.62	2.77
15	8	301	335	3.54	3.57
21.4	4	121	109	4.11	2.78
21	6	160	110	3.9	2.62
21	6	160	110	3.9	2.88
22.8	4	108	93	3.85	2.32
21.4	6	258	110	3.08	3.21
18.7	8	360	175	3.15	3.44
18.1	6	225	105	2.76	3.46
14.3	8	360	245	3.21	3.57
24.4	4	146.7	62	3.69	3.19
22.8	4	140.8	95	3.92	3.15
19.2	6	167.6	123	3.92	3.44
17.8	6	167.6	123	3.92	3.44
16.4	8	275.8	180	3.07	4.07
17.3	8	275.8	180	3.07	3.73
15.2	8	275.8	180	3.07	3.78
10.4	8	472	205	2.93	5.25
10.4	8	460	215	3	5.42
14.7	8	440	230	3.23	5.34

Continued on next page

Table 18: A long 80% width table with repeating headers. (Continued)

mpg	cyl	disp	hp	drat	wt
32.4	4	78.7	66	4.08	2.2
30.4	4	75.7	52	4.93	1.61
33.9	4	71.1	65	4.22	1.83
21.5	4	120.1	97	3.7	2.46
15.5	8	318	150	2.76	3.52
15.2	8	304	150	3.15	3.44
13.3	8	350	245	3.73	3.84
19.2	8	400	175	3.08	3.85
27.3	4	79	66	4.08	1.94
26	4	120.3	91	4.43	2.14
30.4	4	95.1	113	3.77	1.51
15.8	8	351	264	4.22	3.17
19.7	6	145	175	3.62	2.77
15	8	301	335	3.54	3.57
21.4	4	121	109	4.11	2.78

## 8.2 Markdown

### 8.2.1 `style_tt()` does not apply to row headers

This is an important limitation, but it is difficult to get around. See this issue for discussion: <https://github.com/vincentarelbundock/tinytable/issues/125>

Users can use markdown styling directly in `group_tt()` to circumvent this. This is documented in the tutorial.

### 8.2.2 `rowspan` and `colspan`

These arguments are already implemented in the form of “pseudo-spans”, meaning that we flush the content of adjacent cells, but do not modify the row or column borders. This is probably adequate for most needs.

One alternative would be to remove line segments in `finalize_grid()`. I tried this but it is tricky and the results were brittle, so I rolled it back. I’m open to considering a PR if someone wants

to contribute code, but please discuss the feature design in an issue with me before working on this.

### 8.3 Removing elements with `theme_empty()`

In some cases, it is useful to remove elements of an existing `tinytable` object. For example, packages like `modelsummary` often return tables with default styling—such as borders and lines in specific position. If the user adds group labels manually, the original lines and borders will be misaligned.

The code below produces a regression table with group labels but misaligned horizontal rule.

```
#!/ warning: false
library(modelsummary)
library(tinytable)

mod <- lm(mpg ~ factor(cyl) + hp + wt - 1, data = mtcars)

modelsummary(mod) |>
 group_tt(
 i = list(
 "Cylinders" = 1,
 "Others" = 7
)
)
```

To fix this, we can strip the lines and add them back in the correct position.

```
modelsummary(mod) |>
 theme_empty() |>
 group_tt(
 i = list(
 "Cylinders" = 1,
 "Others" = 7
)
) |>
 style_tt(i = 12, line = "b", line_width = .05)
```

	(1)
Cylinders	
factor(cyl)4	35.846 (2.041)
factor(cyl)6	32.487 (2.811)
factor(cyl)8	32.660 (3.835)
Others	
hp	−0.023 (0.012)
wt	−3.181 (0.720)
Num.Obs.	32
R2	0.989
R2 Adj.	0.986
AIC	154.5
BIC	163.3
Log.Lik.	−71.235
RMSE	2.24

	(1)
Cylinders	
factor(cyl)4	35.846
	(2.041)
factor(cyl)6	32.487
	(2.811)
factor(cyl)8	32.660
	(3.835)
Others	
hp	−0.023
	(0.012)
wt	−3.181
	(0.720)
<hr/>	
Num.Obs.	32
R2	0.989
R2 Adj.	0.986
AIC	154.5
BIC	163.3
Log.Lik.	−71.235
RMSE	2.24

## 9 Gallery

This gallery shows advanced `tinytable` examples. A link to the full code and data required to reproduce the full HTML tables is given above each screenshot.

### 9.1 Students

Original table designed by [Illak Blog](#), who created the first version in `gt` format for the RStudio Table Contest. Code and data for the `tinytable` translation are [available on GitHub](#).

UNIDADES EDUCATIVAS					ALUMNOS			PERSONAL DOCENTE		
MODALIDAD	NIVEL	TOTAL	ESTATALES PRIVADAS		TOTAL	ESTATALES PRIVADAS		TOTAL	ESTATALES PRIVADAS	
Común	Inicial	1,757	1,444 (82.19%)	313 (17.81%)	132,342	96,631 (73.02%)	35,711 (26.98%)	10,512	6,768 (64.38%)	3,744 (35.62%)
	Primario	2,042	1,724 (84.43%)	318 (15.57%)	354,360	256,150 (72.29%)	98,210 (27.71%)	31,420	23,544 (74.93%)	7,876 (25.07%)
	Secundario	932	517 (55.47%)	415 (44.53%)	346,705	208,603 (60.17%)	138,102 (39.83%)	60,133	40,415 (67.21%)	19,718 (32.79%)
	Superior	207	83 (40.10%)	124 (59.90%)	76,736	45,014 (58.66%)	31,722 (41.34%)	10,159	5,525 (54.39%)	4,634 (45.61%)
Especial	Todos los niveles	101	60 (59.41%)	41 (40.59%)	5,019	2,765 (55.09%)	2,254 (44.91%)	2,764	1,827 (66.10%)	937 (33.90%)
Hospitalaria y Domiciliaria	Todos los niveles	3	2 (66.67%)	1 (33.33%)	396	386 (97.47%)	10 (2.53%)	111	100 (90.09%)	11 (9.91%)
Jóvenes y Adultos	Primario	178	177 (99.44%)	1 (0.56%)	4,560	4,543 (99.63%)	17 (0.37%)	466	464 (99.57%)	2 (0.43%)
	Secundario	136	111 (81.62%)	25 (18.38%)	38,332	33,425 (87.20%)	4,907 (12.80%)	7,385	6,936 (93.92%)	449 (6.08%)
Artística		33	28 (84.85%)	5 (15.15%)	9,700	8,659 (89.27%)	1,041 (10.73%)	594	510 (85.86%)	84 (14.14%)
Formación Profesional		83	83 (100.00%)		32,623	32,623 (100.00%)		641		

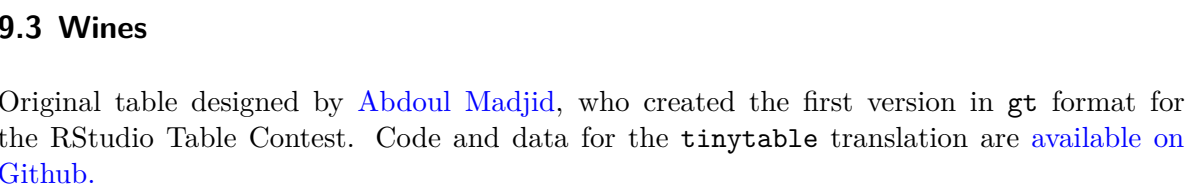
Adapted from [a table by Illak Zapata](#).

### 9.2 Lemurs

Original table designed by [Nicola Rennie](#), who created the first version in `gt` format for the 2022 RStudio Table Contest. Code and data for the `tinytable` translation are [available on GitHub](#).







































Lemurs are a unique group of primates native to Madagascar, an island off the coast of east Africa. Although they are related to monkeys and apes, lemurs make up a separate branch of the primate family tree and are classified as a superfamily, made up of five individual lemur families and more than 100 different species. Founded in 1966 on the campus of Duke University in Durham, NC, the Duke Lemur Center is a world leader in the study, care, and protection of lemurs—Earth's most threatened group of mammals. The Duke Lemur Center houses the world's largest and most diverse population of lemurs outside their native Madagascar.



Exceptional Wines

Great wines improve with age. Let's dive into some of the most extraordinary cuvées in the world. Those whose grapes possess ethereal aromas and pure minerality that give focus and energy. Those demonstrating great character, balance with good acidity and plush tannins.




































							
 La Romanee Grand Cru Monopole	Domaine du Comte Liger-Belair 2013	 Bourgogne	 Pinot Noir		4.9 ★★★★★ 27 reviews	\$8,360	13.0%
 Sauternes	Château d'Yquem 1945	 Bordeaux	 Sémillon		4.9 ★★★★★ 39 reviews	\$4,680	13.5%
 Wraith Cabernet Sauvignon	Hundred Acre 2014	 California	 Cabernet Sauvignon		4.9 ★★★★★ 79 reviews	\$805	15.5%
 Vintage Port	Ferreira 2018	 Duriense	 Tinto Cao, Tinta Barroca, Cima da Roca, Touriga Nacional		4.9 ★★★★★ 95 reviews	\$115	20.0%
 Grand Vin Pauillac (Premier Grand Cru Classé)	Château Latour 1990	 Bordeaux	 Cabernet Sauvignon, Merlot, Petit Verdot		4.8 ★★★★★ 1513 reviews	\$1,580	13.5%
 Pessac-Léognan (Premier Grand Cru Classé)	Château Haut-Brion 1989	 Bordeaux	 Merlot, Cabernet Sauvignon, Cabernet Franc		4.8 ★★★★★ 1245 reviews	\$1,350	13.0%
 Unico	Vega Sicilia 1999	 Castilla y León	 Tempranillo, Cabernet Sauvignon		4.8 ★★★★★ 1636 reviews	\$620	13.5%

Adapted from [a beautiful table by Abdoul Madjid](#).

9.4 AI Stocks

Original table designed by [Arnav Chauhan](#), who created the first version for the 2024 Posit Table Contest. Code and data for the `tinytable` version are [available on GitHub](#).

TOP AI STOCK PERFORMANCE  
This table summarizes the performance of top 14 AI Stocks by Market Cap (May 2024)

Company	Price 07 Oct 2025	Price % Change	Price % Change	52-Week High	52-Week Low	Price Trend	Volume 07 Oct 2025	Volume % Change	Volume Trend
 Adobe Inc. ADBE INC.	\$348.31	▼ -0.52%		\$552.96	\$333.65		3.27M	▼ -38.69%	
 Aurora Innovation, Inc. AURORA INNOVATION, INC.	\$5.40	▼ -2.35%		\$10.19	\$4.99		13.03M	▼ -30.89%	
 Dynatrace, Inc. DYNATRACE, INC.	\$48.13	▼ -2.08%		\$62.42	\$41.21		2.84M	▼ -26.07%	
 Alphabet Inc. ALPHABET INC.	\$247.13	▼ -1.74%		\$255.24	\$146.27		13.80M	▼ -24.61%	
 IBM Corporation IBM CORPORATION	\$293.87	▲ 1.54%		\$293.87	\$199.27		7.19M	▲ 149.33%	
 Mobileye Global, Inc. MOBILEYE GLOBAL, INC.	\$15.09	▲ 0.47%		\$21.85	\$11.77		6.41M	▼ -2.53%	
 Meta Platforms, Inc. META PLATFORMS, INC.	\$713.08	▼ -0.36%		\$789.47	\$483.96		12.04M	▼ -44.42%	
 Microsoft Corporation MICROSOFT CORPORATION	\$523.98	▼ -0.87%		\$534.76	\$353.33		14.60M	▼ -31.73%	
 NVIDIA Corporation NVIDIA CORPORATION	\$185.04	▼ -0.27%		\$188.89	\$94.30		139.44M	▼ -11.57%	