

Package ‘tinytable’

September 29, 2025

Type Package

Title Simple and Configurable Tables in 'HTML', 'LaTeX', 'Markdown', 'Word', 'PNG', 'PDF', and 'Typst' Formats

Description Create highly customized tables with this simple and dependency-free package. Data frames can be converted to 'HTML', 'LaTeX', 'Markdown', 'Word', 'PNG', 'PDF', or 'Typst' tables. The user interface is minimalist and easy to learn. The syntax is concise. 'HTML' tables can be customized using the flexible 'Bootstrap' framework, and 'LaTeX' code with the 'tabularray' package.

Version 0.14.0

Imports methods

Depends R (>= 4.1.0)

Enhances knitr

Suggests base64enc, broom, data.table (>= 1.15.2), estimatr, ggplot2, gh, glue, htmltools, litedown (>= 0.6), magrittr, marginaleffects, modelsummary, pandoc, quarto, Rdatasets, rmarkdown, rstudioapi, scales, stringi, tibble, tinysnapshot (>= 0.2.0), tinytest, tinytex, xfun (>= 0.51), webshot2

URL <https://vincentarelbundock.github.io/tinytable/>

BugReports <https://github.com/vincentarelbundock/tinytable/issues>

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Vincent Arel-Bundock [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-2042-7063>>)

Maintainer Vincent Arel-Bundock <vincent.arel-bundock@umontreal.ca>

Repository CRAN

Date/Publication 2025-09-29 12:10:02 UTC

Contents

tinytable-package	2
format_tt	3
format_vector	8
group_tt	10
plot_tt	13
plot_vector	16
print.tinytable	18
rbind2,tinytable,tinytable-method	19
save_tt	21
style_tt	23
style_vector	29
subset.tinytable	31
theme_default	31
theme_empty	32
theme_grid	32
theme_html	33
theme_latex	34
theme_markdown	37
theme_revealjs	38
theme_rotate	38
theme_stripped	39
theme_tt	39
theme_typst	40
tt	40
Index	46

tinytable-package	<i>Simple and Configurable Tables in 'HTML', 'LaTeX', 'Markdown', 'Word', 'PNG', 'PDF', and 'Typst' Formats</i>
-------------------	---

Description

Create highly customized tables with this simple and dependency-free package. Data frames can be converted to 'HTML', 'LaTeX', 'Markdown', 'Word', 'PNG', 'PDF', or 'Typst' tables. The user interface is minimalist and easy to learn. The syntax is concise. 'HTML' tables can be customized using the flexible 'Bootstrap' framework, and 'LaTeX' code with the 'tabularray' package.

Package Content

Index of help topics:

format_tt	Format columns of a data frame
format_vector	Format a Vector
group_tt	Spanning labels to identify groups of rows or columns

plot_tt	Insert images and inline plots into tinytable objects
plot_vector	Create Plot Vector
print.tinytable	Print, display, or convert a tinytable object
rbind2, tinytable, tinytable-method	Combine 'tinytable' objects by rows (vertically)
save_tt	Save a Tiny Table to File
style_tt	Style a Tiny Table
style_vector	Style a Vector with Text Formatting
subset.tinytable	Subsetting a 'tinytable' object
theme_default	Default theme for TinyTable
theme_empty	Theme for a void table
theme_grid	Grid theme with borders around all cells
theme_html	HTML-specific styles and options
theme_latex	LaTeX-Specific Theme for 'tinytable'
theme_markdown	Markdown theme with optional ANSI color support and grid customization
theme_revealjs	RevealJS presentation theme
theme_rotate	Rotate table theme (LaTeX and Typst only)
theme_stripped	Striped theme with alternating row colors
theme_tt	Deprecated: Use format-specific theme functions instead
theme_typst	Typst-specific styles and options
tinytable-package	Simple and Configurable Tables in 'HTML', 'LaTeX', 'Markdown', 'Word', 'PNG', 'PDF', and 'Typst' Formats
tt	Draw a Tiny Table

Maintainer

Vincent Arel-Bundock <vincent.arel-bundock@umontreal.ca>

Author(s)

Vincent Arel-Bundock [aut, cre] (ORCID: <<https://orcid.org/0000-0003-2042-7063>>)

format_tt	<i>Format columns of a data frame</i>
-----------	---------------------------------------

Description

This function formats the columns of a data frame based on the column type (logical, date, numeric). It allows various formatting options like significant digits, decimal points, and scientific notation. It also includes custom formatting for date and boolean values. If this function is applied several times to the same cell, the last transformation is retained and the previous calls are ignored, except for the escape argument which can be applied to previously transformed data.

Usage

```

format_tt(
  x,
  i = NULL,
  j = NULL,
  digits = get_option("tinytable_format_digits", default = NULL),
  num_fmt = get_option("tinytable_format_num_fmt", default = "significant"),
  num_zero = get_option("tinytable_format_num_zero", default = FALSE),
  num_suffix = get_option("tinytable_format_num_suffix", default = FALSE),
  num_mark_big = get_option("tinytable_format_num_mark_big", default = ""),
  num_mark_dec = get_option("tinytable_format_num_mark_dec", default =
    getOption("OutDec", default = ".")),
  date = get_option("tinytable_format_date", default = NULL),
  bool = get_option("tinytable_format_bool", default = NULL),
  math = get_option("tinytable_format_math", default = FALSE),
  other = get_option("tinytable_format_other", default = NULL),
  replace = get_option("tinytable_format_replace", default = FALSE),
  escape = get_option("tinytable_format_escape", default = FALSE),
  markdown = get_option("tinytable_format_markdown", default = FALSE),
  quarto = get_option("tinytable_format_quarto", default = FALSE),
  fn = get_option("tinytable_format_fn", default = NULL),
  sprintf = get_option("tinytable_format_sprintf", default = NULL),
  linebreak = get_option("tinytable_format_linebreak", default = NULL),
  output = get_option("tinytable_format_output", default = NULL)
)

tt_format(
  x,
  i = NULL,
  j = NULL,
  digits = get_option("tinytable_format_digits", default = NULL),
  num_fmt = get_option("tinytable_format_num_fmt", default = "significant"),
  num_zero = get_option("tinytable_format_num_zero", default = FALSE),
  num_suffix = get_option("tinytable_format_num_suffix", default = FALSE),
  num_mark_big = get_option("tinytable_format_num_mark_big", default = ""),
  num_mark_dec = get_option("tinytable_format_num_mark_dec", default =
    getOption("OutDec", default = ".")),
  date = get_option("tinytable_format_date", default = NULL),
  bool = get_option("tinytable_format_bool", default = NULL),
  math = get_option("tinytable_format_math", default = FALSE),
  other = get_option("tinytable_format_other", default = NULL),
  replace = get_option("tinytable_format_replace", default = FALSE),
  escape = get_option("tinytable_format_escape", default = FALSE),
  markdown = get_option("tinytable_format_markdown", default = FALSE),
  quarto = get_option("tinytable_format_quarto", default = FALSE),
  fn = get_option("tinytable_format_fn", default = NULL),
  sprintf = get_option("tinytable_format_sprintf", default = NULL),
  linebreak = get_option("tinytable_format_linebreak", default = NULL),

```

```

    output = get_option("tinytable_format_output", default = NULL)
)

```

Arguments

x	A data frame or a vector to be formatted.
i	Numeric vector or string. - Numeric vector: Row indices where the styling should be applied. Can be a single value or a vector. - String: Table components to format "caption", "colnames", "groupi" (row group labels), "~groupi" (non-group rows), "groupj" (column group labels), "notes". - If both the i and j are omitted (default: NULL), formatting is applied to all table elements, including caption, notes, and group labels.
j	Column indices where the styling should be applied. Can be: - Integer vectors indicating column positions. - Character vector indicating column names. - A single string specifying a Perl-style regular expression used to match column names. - Unquoted expression: Non-standard evaluation is supported. When supplying an unquoted expression, it is first evaluated in the calling environment, then in an environment that includes the columns of the original data passed to tt(), and groupi indices. See examples below.
digits	Number of significant digits or decimal places.
num_fmt	The format for numeric values; one of 'significant', 'significant_cell', 'decimal', or 'scientific'.
num_zero	Logical; if TRUE, trailing zeros are kept in "decimal" format (but not in "significant" format).
num_suffix	Logical; if TRUE display short numbers with digits significant digits and K (thousands), M (millions), B (billions), or T (trillions) suffixes.
num_mark_big	Character to use as a thousands separator.
num_mark_dec	Decimal mark character. Default is the global option 'OutDec'.
date	A string passed to the format() function, such as "%Y-%m-%d". See the "Details" section in ?strptime
bool	A function to format logical columns. Defaults to title case.
math	Logical. If TRUE, wrap cell values in math mode $$. . $$. This is useful for LaTeX output or with HTML MathJax options(tinytable_html_mathjax=TRUE).
other	A function to format columns of other types. Defaults to as.character().
replace	Logical, String or Named list of vectors - TRUE: Replace NA and NaN by an empty string. - FALSE: Print NA and NaN as strings. - String: Replace NA and NaN entries by the user-supplied string.

	• Named list: Replace matching elements of the vectors in the list by their names. Example: <pre>list("-", c(NA, NaN), "Tiny" = -Inf, "Massive" = Inf)</pre>
escape	Logical or "latex" or "html". If TRUE, escape special characters to display them as text in the format of the output of a <code>tt()</code> table. • If <code>i</code> and <code>j</code> are both NULL, escape all cells, column names, caption, notes, and spanning labels created by <code>group_tt()</code>.
markdown	Logical; if TRUE, render markdown syntax in cells. Ex: <code>_italicized text_</code> is properly italicized in HTML and LaTeX.
quarto	Logical. Enable Quarto data processing and wrap cell content in a <code>data-qmd span</code> (HTML) or <code>\QuartoMarkdownBase64{}</code> macro (LaTeX). See warnings in the Global Options section below.
fn	Function for custom formatting. Accepts a vector and returns a character vector of the same length.
sprintf	String passed to the <code>?sprintf</code> function to format numbers or interpolate strings with a user-defined pattern (similar to the <code>glue</code> package, but using Base R).
linebreak	NULL or a single string. If it is a string, replaces that string with appropriate line break sequences depending on the output format (HTML: <code>
</code> , LaTeX: <code>\\</code> , Typst: <code>\\</code>). Markdown output is excluded from line break replacement.
output	Apply formatting only if the <code>tt()</code> object is rendered in the specified format. One of "latex", "html", "typst", or "markdown". If NULL (default), apply formatting regardless of the output format.

Value

A data frame with formatted columns.

Global options

Options can be set with `options()` and change the default behavior of `tinytable`. For example:

```
options(tinytable_tt_digits = 4)
tt(head(iris))
```

You can set options in a script or via `.Rprofile`. Note: be cautious with `.Rprofile` settings as they may affect reproducibility.

Default values for function arguments:

Nearly all of the package's functions retrieve their default values from global options. This allows you to set defaults once and apply them to all tables without needing to specify them each time. For example, to fix the `digits` argument of the `tt()` function globally, call:

```
options(tinytable_tt_digits = 4)
```

In addition, some more specific options are available to control the behavior of the package in specific contexts.

- `tinytable_html_mathjax`: Insert MathJax scripts (warning: may conflict if MathJax is loaded elsewhere)
- `tinytable_pdf_clean`: Delete temporary and log files for pdf output in `save_tt()`
- `tinytable_color_name_normalization`: Enable/disable automatic color name processing (default: TRUE). When enabled, R color names recognized by `col2rgb()` are converted to hex format for consistent rendering across HTML, LaTeX, and Typst formats. If R color conversion fails, LaTeX color names are used as fallback. Colors explicitly supplied as hex values with "#" prefix are passed through unchanged. Set to FALSE to disable processing and pass color names unchanged.

Quarto:

The `format_tt(quarto=TRUE)` argument enables Quarto data processing with some limitations:

1. The `\QuartoMarkdownBase64{}` LaTeX macro may not process references and markdown as expected
2. Quarto processing may conflict with `tinytable` styling/formatting

Options:

- `tinytable_quarto_disable_processing`: Disable Quarto cell processing

Example of Quarto-specific code in cells:

```
x <- data.frame(Math = "x^2^", Citation = "@Lovelace1842")
fn <- function(z) sprintf("<span data-qmd='%s'></span>", z)
tt(x) |> format_tt(i = 1, fn = fn)
```

For more details on Quarto table processing: <https://quarto.org/docs/authoring/tables.html#disabling-quarto-table-processing>

Examples

```
dat <- data.frame(
  a = rnorm(3, mean = 10000),
  b = rnorm(3, 10000)
)
tab <- tt(dat)
format_tt(tab,
  digits = 2,
  num_mark_dec = ",",
  num_mark_big = " "
)

k <- tt(data.frame(x = c(0.000123456789, 12.4356789)))
format_tt(k, digits = 2, num_fmt = "significant_cell")

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, num_fmt = "decimal", num_zero = TRUE) |>
  format_tt(j = "c", digits = 2, num_suffix = TRUE)
```

```

y <- tt(data.frame(x = c(123456789.678, 12435.6789)))
format_tt(y, digits = 3, num_mark_big = " ")

x <- tt(data.frame(Text = c("_italicized text_", "__bold text__")))
format_tt(x, markdown = TRUE)

# Line breaks using linebreak argument
d <- data.frame(Text = "First line<br>Second line")
tt(d) |> format_tt(linebreak = "<br>")

# Non-standard evaluation (NSE)
dat <- data.frame(
  w = c(143002.2092, 201399.181, 100188.3883),
  x = c(1.43402, 201.399, 0.134588),
  y = as.Date(c(897, 232, 198), origin = "1970-01-01"),
  z = c(TRUE, TRUE, FALSE)
)
tt(dat) |>
  format_tt(i = w > 150000, j = w, digits = 0, num_mark_big = ",")

tab <- data.frame(a = c(NA, 1, 2), b = c(3, NA, 5))
tt(tab) |> format_tt(replace = "--")

dat <- data.frame(
  "LaTeX" = c("Dollars $", "Percent %", "Underscore _"),
  "HTML" = c("<br>", "<sup>4</sup>", "<emph>blah</emph>")
)
tt(dat) |> format_tt(escape = TRUE)

```

format_vector

*Format a Vector***Description**

Format a Vector

Usage

```

format_vector(
  x,
  output = "html",
  digits = NULL,
  num_fmt = "significant",
  num_zero = FALSE,
  num_suffix = FALSE,
  num_mark_big = "",
  num_mark_dec = getOption("OutDec", default = "."),

```

```

    date = NULL,
    bool = NULL,
    math = FALSE,
    other = NULL,
    replace = FALSE,
    escape = FALSE,
    markdown = FALSE,
    quarto = FALSE,
    fn = NULL,
    sprintf = NULL,
    linebreak = NULL
  )

```

Arguments

x	A vector to be formatted.
output	Output format. One of "html", "latex", "typst", "markdown", etc.
digits	Number of significant digits or decimal places.
num_fmt	The format for numeric values; one of 'significant', 'significant_cell', 'decimal', or 'scientific'.
num_zero	Logical; if TRUE, trailing zeros are kept in "decimal" format (but not in "significant" format).
num_suffix	Logical; if TRUE display short numbers with digits significant digits and K (thousands), M (millions), B (billions), or T (trillions) suffixes.
num_mark_big	Character to use as a thousands separator.
num_mark_dec	Decimal mark character. Default is the global option 'OutDec'.
date	A string passed to the format() function, such as "%Y-%m-%d". See the "Details" section in ?strptime
bool	A function to format logical columns. Defaults to title case.
math	Logical. If TRUE, wrap cell values in math mode $$. . $$. This is useful for LaTeX output or with HTML MathJax options(tinytable_html_mathjax=TRUE).
other	A function to format columns of other types. Defaults to as.character().
replace	Logical, String or Named list of vectors • TRUE: Replace NA and NaN by an empty string. • FALSE: Print NA and NaN as strings. • String: Replace NA and NaN entries by the user-supplied string. • Named list: Replace matching elements of the vectors in the list by their names. Example: <pre>list("-", c(NA, NaN), "Tiny" = -Inf, "Massive" = Inf)</pre>
escape	Logical or "latex" or "html". If TRUE, escape special characters to display them as text in the format of the output of a tt() table. • If i and j are both NULL, escape all cells, column names, caption, notes, and spanning labels created by group_tt().

markdown	Logical; if TRUE, render markdown syntax in cells. Ex: <code>_italicized text_</code> is properly italicized in HTML and LaTeX.
quarto	Logical. Enable Quarto data processing and wrap cell content in a <code>data-qmd span</code> (HTML) or <code>\QuartoMarkdownBase64{}</code> macro (LaTeX). See warnings in the Global Options section below.
fn	Function for custom formatting. Accepts a vector and returns a character vector of the same length.
sprintf	String passed to the <code>?sprintf</code> function to format numbers or interpolate strings with a user-defined pattern (similar to the <code>glue</code> package, but using Base R).
linebreak	NULL or a single string. If it is a string, replaces that string with appropriate line break sequences depending on the output format (HTML: <code>
</code> , LaTeX: <code>\\\\</code> , Typst: <code>\\</code>). Markdown output is excluded from line break replacement.

Details

This function formats a vector by passing it to `format_tt()`. All formatting arguments must be of length 1 or `length(x)`.

Value

A character vector with formatted values.

Examples

```
# Format numeric vector
format_vector(c(1234.567, 9876.543), digits = 2, num_mark_big = ",")

# Format dates
dates <- as.Date(c("2023-01-01", "2023-12-31"))
format_vector(dates, date = "%B %d, %Y")

# Format logical values
format_vector(c(TRUE, FALSE, TRUE), bool = function(x) ifelse(x, "Yes", "No"))
```

group_tt	<i>Spanning labels to identify groups of rows or columns</i>
----------	--

Description

Spanning labels to identify groups of rows or columns
Alias for `group_tt()`

Usage

```
group_tt(
  x,
  i = getOption("tinytable_group_i", default = NULL),
  j = getOption("tinytable_group_j", default = NULL),
  ...
)

tt_group(
  x,
  i = getOption("tinytable_group_i", default = NULL),
  j = getOption("tinytable_group_j", default = NULL),
  ...
)
```

Arguments

- | | |
|-----|---|
| x | A data frame, data table, or tibble to be rendered as a table. |
| i | Character vector, named list, or integer vector <ul style="list-style-type: none"> • A character vector of labels with length equal to the number of rows in x • A named list of row indices to group. The names of the list will be used as labels. The indices represent the position where labels should be inserted in the original table. For example, <ul style="list-style-type: none"> – <code>i=list("Hello"=5)</code>: insert the "Hello" label after the 4th row in the original table. – <code>i=list("Hello"=2, "World"=2)</code>: insert the two labels consecutively after the 1st row in the original table. – <code>i=list("Foo Bar"=0)</code>: insert the label in the first row after the header. • Vector of positive integers: For matrix insertion: i specifies row positions and j must be a character matrix to insert in the table (see below for details). |
| j | String, named list, or character matrix <ul style="list-style-type: none"> • Named list of column indices to group, ex: <code>j=list("A"=1:2, "B"=3:6)</code>. The names of the list will be used as labels. See below for more examples. Note: empty labels must be a space: " ". • A single string when column names include the group name as a prefix, ex: <code>group1_column1</code>, <code>group1_column2</code>, etc. • Character matrix for inserting rows at positions specified by i. The matrix must have the same number of columns as the table, or be a single column with a number of elements that is a multiple of the table's column count (which will be automatically reshaped). Each row of the matrix matches an element • Unquoted expression: Non-standard evaluation is supported. When supplying an unquoted expression, it is first evaluated in the calling environment, then in the data frame passed to <code>tt()</code>. |
| ... | Other arguments are ignored. |

Details

Warning: The `style_tt()` can normally be used to style the group headers, as expected, but that feature is not available for Markdown and Word tables.

Value

An object of class `tt` representing the table.

Markdown limitations

Markdown is a text-only format that only supports these styles: italic, bold, strikethrough. The width argument is also unavailable. These limitations exist because there is no standard markdown syntax for the other styling options.

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.

Word limitations

Word tables only support these styles: italic, bold, strikethrough. The width argument is also unavailable. Moreover, the `style_tt()` function cannot be used to style headers inserted by the `group_tt()` function; instead, you should style the headers directly in the header definition using markdown syntax: `group_tt(i = list("*italic header*" = 2))`. These limitations are due to the fact that we create Word documents by converting a markdown table to .docx via the Pandoc software, which requires going through a text-only intermediate format.

Examples

```
# vector of row labels
dat <- data.frame(
  label = c("a", "a", "a", "b", "b", "c", "a", "a"),
  x1 = rnorm(8),
  x2 = rnorm(8)
)
tt(dat[, 2:3]) |> group_tt(i = dat$label)

# named lists of labels
tt(mtcars[1:10, 1:5]) |>
  group_tt(
    i = list(
      "Hello" = 3,
      "World" = 8
    ),
    j = list(
      "Foo" = 2:3,
      "Bar" = 4:5
    )
  )

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
  ))

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

x <- mtcars[1:5, 1:6]
tt(x) |>
  group_tt(j = list("Hello" = 1:2, "World" = 3:4, "Hello" = 5:6)) |>
  group_tt(j = list("Foo" = 1:3, "Bar" = 4:6))

# column names with delimiters
dat <- data.frame(
  A_id = 1,
  A_a1 = 2,
  A_a2 = "3",
  B_b1 = 4,
  B_b2 = 5,
  B_C = 6
)
tt(dat) |> group_tt(j = "_")

# matrix insertion
rowmat <- matrix(colnames(iris))
tt(head(iris, 7)) |>
  group_tt(i = c(2, 5), j = rowmat)

rowmat <- matrix(c(
  "a", "b", "c", "d", "e",
  1, 2, 3, 4, 5))
tt(head(iris, 7)) |>
  group_tt(i = 2, j = rowmat) |>
  style_tt(i = "groupi", background = "pink")

```

plot_tt

Insert images and inline plots into tinytable objects

Description

The `plot_tt()` function allows for the insertion of images and inline plots into tinytable objects. This function can handle both local and web-based images.

Usage

```

plot_tt(
  x,
  i = NULL,
  j = NULL,
  fun = NULL,
  data = NULL,
  color = "black",
  xlim = NULL,
  height = 1,
  height_plot = 400,
  width_plot = 1200,
  images = NULL,
  sprintf = "%s",
  assets = "tinytable_assets",
  ...
)

tt_plot(
  x,
  i = NULL,
  j = NULL,
  fun = NULL,
  data = NULL,
  color = "black",
  xlim = NULL,
  height = 1,
  height_plot = 400,
  width_plot = 1200,
  images = NULL,
  sprintf = "%s",
  assets = "tinytable_assets",
  ...
)

```

Arguments

- | | |
|-----|--|
| x | A tinytable object. |
| i | Integer vector, the row indices where images are to be inserted. If NULL, images will be inserted in all rows. |
| j | Integer vector, the column indices where images are to be inserted. If NULL, images will be inserted in all columns. |
| fun | String or function to generate inline plots. <ul style="list-style-type: none"> • Built-in plot types (strings): <ul style="list-style-type: none"> – "histogram": Creates histograms from numeric vectors. Accepts color argument. |

- "density": Creates density plots from numeric vectors. Accepts color argument.
- "bar": Creates horizontal bar charts from single numeric values. Accepts color and xlim arguments.
- "barpct": Creates horizontal percentage bar charts from single numeric values between 0 and 1. Accepts color and background arguments.
- "line": Creates line plots from data frames with x and y columns. Accepts color and xlim arguments.
- Custom functions:
 - Functions that return ggplot2 objects.
 - Functions that return another function which generates a base R plot, ex: `function(x) {function() hist(x)}`
 - Note: When using custom ggplot2 functions that return plots with text elements, the text size will normally need to be adjusted because the plot is inserted as a very small image in the table. Text sizes of 1 or smaller often work well (e.g., `theme(text = element_text(size = 1))`).
- See the tutorial on the [tinytable](#) website for more information.

<code>data</code>	a list of data frames or vectors to be used by the plotting functions in <code>fun</code> .
<code>color</code>	string Name of color to use for inline plots (passed to the <code>col</code> argument base graphics plots in R).
<code>xlim</code>	Numeric vector of length 2.
<code>height</code>	Numeric, the height of the images in the table in em units.
<code>height_plot</code>	Numeric, the height of generated plot images in pixels (default: 400).
<code>width_plot</code>	Numeric, the width of generated plot images in pixels (default: 1200).
<code>images</code>	Character vector, the paths to the images to be inserted. Paths are relative to the main table file or Quarto (Rmarkdown) document.
<code>sprintf</code>	Character string, a sprintf format string to format the generated cell content. Default is "%s" which displays the content as-is. Use this to wrap images or plots in custom markup.
<code>assets</code>	Path to the directory where generated assets are stored. This path is relative to the location where a table is saved.
<code>...</code>	Extra arguments are passed to the function in <code>fun</code> . Important: Custom plotting functions must always have <code>...</code> as an argument.

Details

The `plot_tt()` can insert images and inline plots into tables.

Value

A modified `tinytable` object with images or plots inserted.

Examples

```
## Not run:
# Built-in plot types
plot_data <- list(mtcars$mpg, mtcars$hp, mtcars$qsec)

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

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

# Percentage data (values between 0 and 1)
pct_data <- list(0.65, 0.82, 0.41)

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 = "bar", data = list(2, 3, 6), color = "orange") |>
  plot_tt(j = 5, fun = "barpct", data = pct_data, color = "steelblue") |>
  plot_tt(j = 6, fun = "line", data = lines, color = "blue") |>
  style_tt(j = 2:6, align = "c")

# Custom function example (must have ... argument)
custom_hist <- function(d, ...) {
  function() hist(d, axes = FALSE, ann = FALSE, col = "lightblue")
}

tt(data.frame(Variables = "mpg", Histogram = "")) |>
  plot_tt(j = 2, fun = custom_hist, data = list(mtcars$mpg))

## End(Not run)
```

plot_vector

Create Plot Vector

Description

Create Plot Vector

Usage

```
plot_vector(
  output = "html",
```

```

    fun = NULL,
    data = NULL,
    color = "black",
    xlim = NULL,
    height = 1,
    height_plot = 400,
    width_plot = 1200,
    images = NULL,
    sprintf = "%s",
    assets = "tinytable_assets",
    ...
)

```

Arguments

output	Output format. One of "html", "latex", "typst", "markdown", etc.
fun	String or function to generate inline plots. • Built-in plot types (strings): – "histogram": Creates histograms from numeric vectors. Accepts color argument. – "density": Creates density plots from numeric vectors. Accepts color argument. – "bar": Creates horizontal bar charts from single numeric values. Accepts color and xlim arguments. – "barpct": Creates horizontal percentage bar charts from single numeric values between 0 and 1. Accepts color and background arguments. – "line": Creates line plots from data frames with x and y columns. Accepts color and xlim arguments. • Custom functions: – Functions that return ggplot2 objects. – Functions that return another function which generates a base R plot, ex: <code>function(x) {function() hist(x)}</code> – Note: When using custom ggplot2 functions that return plots with text elements, the text size will normally need to be adjusted because the plot is inserted as a very small image in the table. Text sizes of 1 or smaller often work well (e.g., <code>theme(text = element_text(size = 1))</code>). • See the tutorial on the tinytable website for more information.
data	a list of data frames or vectors to be used by the plotting functions in fun.
color	string Name of color to use for inline plots (passed to the col argument base graphics plots in R).
xlim	Numeric vector of length 2.
height	Numeric, the height of the images in the table in em units.
height_plot	Numeric, the height of generated plot images in pixels (default: 400).

width_plot	Numeric, the width of generated plot images in pixels (default: 1200).
images	Character vector, the paths to the images to be inserted. Paths are relative to the main table file or Quarto (Rmarkdown) document.
sprintf	Character string, a sprintf format string to format the generated cell content. Default is "%s" which displays the content as-is. Use this to wrap images or plots in custom markup.
assets	Path to the directory where generated assets are stored. This path is relative to the location where a table is saved.
...	Extra arguments are passed to the function in fun. Important: Custom plotting functions must always have ... as an argument.

Details

This function creates plots by creating a temporary `tt()` object and applying `plot_tt()`. It returns a character vector containing the image paths or HTML tags for the plots.

Value

A character vector with plot file paths or HTML tags.

Examples

```
## Not run:
# Create histogram plots
plot_data <- list(rnorm(100), rnorm(50))
plot_vector(fun = "histogram", data = plot_data, output = "html")

# Create density plots
plot_vector(fun = "density", data = plot_data, output = "latex")

# Create bar plots from single values
bar_data <- list(0.5, 0.8, 0.3)
plot_vector(fun = "barpct", data = bar_data, output = "html")

## End(Not run)
```

print.tinytable	<i>Print, display, or convert a tinytable object</i>
-----------------	--

Description

This function is called automatically by R whenever a `tinytable` object is anprinted to the console or in an HTML viewer pane.

Usage

```
## S3 method for class 'tinytable'
print(x, output = get_option("tinytable_print_output", default = NULL), ...)
```

Arguments

- | | |
|--------|--|
| x | A data frame, data table, or tibble to be rendered as a table. |
| output | format in which a Tiny Table is printed: NULL or one of "latex", "markdown", "html", "typst", "dataframe", "tabulator". If NULL, the output is chosen based on these rules: <ul style="list-style-type: none"> • When called from a script in non-interactive mode, the default is "markdown" (interactive() == FALSE). • When called interactively in RStudio, the default is to display an HTML table in the viewer pane. • When called interactively in another development environment, the default is "markdown". • The default print output can be changed for an entire R session by calling: <code>options(tinytable_print_output = "html")</code> • The default print output can be changed for a single tinytable object by modifying the output S4 slot. |
| ... | Other arguments are ignored. |

Details

When printing to HTML in `interactive()` mode, a temporary file is created and `viewer()` is called to preview the file with the local browser (ex: Firefox or Chrome). The temporary file is then automatically cleaned up. On some operating systems, like some Linux distributions, browser do not have read access to the `/tmp/` directory. In such cases, users can specify a custom location to store temporary HTML files. Note that this prevents `tinytable` from automatically cleaning up temporary files automatically.

```
options(tinytable_tempdir = "/home/username/temp_directory")
```

Value

launch a browser window or `cat()` the table to console.

```
rbind2, tinytable, tinytable-method
```

Combine tinytable objects by rows (vertically)

Description

Combine tinytable objects by rows (vertically)

Usage

```
## S4 method for signature 'tinytable, tinytable'
rbind2(x, y, use_names = TRUE, headers = TRUE, ...)
```

Arguments

<code>x</code>	tinytable object
<code>y</code>	tinytable object
<code>use_names</code>	'TRUE' binds by matching column name, 'FALSE' by position
<code>headers</code>	Logical. TRUE inserts the colnames of y as an extra row between the two tables.
<code>...</code>	Additional arguments are ignored.

Details

`format_tt()` calls applied to `x` or `y` are evaluated before binding, to allow distinct formatting for each panel.

Calls to other `tinytable` functions such as `style_tt()` or `group_tt()` are ignored when applied to `x` or `y`. These functions should be applied to the final table instead.

Information in these S4 slots is carried over from `x` to the combined table:

- `x@output`
- `x@caption`
- `x@width`

Information in these S4 slots is concatenated and carried over to the combined table:

- `c(x@notes, y@notes)`

This function relies on the `rbindlist()` function from the `data.table` package.

Examples

```
library(tinytable)
x <- tt(mtcars[1:3, 1:2], caption = "Combine two tiny tables.")
y <- tt(mtcars[4:5, 8:10])

# rbind() does not support additional arguments
# rbind2() supports additional arguments

# basic combination
rbind(x, y)

rbind(x, y) |> format_tt(replace = "")

# omit y header
rbind2(x, y, headers = FALSE)

# bind by position rather than column names
rbind2(x, y, use_names = FALSE)
```

save_tt

*Save a Tiny Table to File***Description**

This function saves an object of class `tinytable` to a specified file and format, with an option to overwrite existing files.

Usage

```
save_tt(
  x,
  output = get_option("tinytable_save_output", default = NULL),
  overwrite = get_option("tinytable_save_overwrite", default = FALSE)
)

tt_save(
  x,
  output = get_option("tinytable_save_output", default = NULL),
  overwrite = get_option("tinytable_save_overwrite", default = FALSE)
)
```

Arguments

<code>x</code>	The <code>tinytable</code> object to be saved.
<code>output</code>	String or file path. • If <code>output</code> is "markdown", "latex", "html", "typst", or "tabulator", the table is returned in a string as an R object. • If <code>output</code> is a valid file path, the table is saved to file. The supported extensions are: .docx, .html, .png, .pdf, .tex, .typ, and .md (with aliases .txt, .Rmd and .qmd). • If the global option <code>tinytable_html_portable</code> is TRUE, the images are included in the HTML as base64 encoded string instead of link to a local file.
<code>overwrite</code>	A logical value indicating whether to overwrite an existing file.

Value

A string with the table when `output` is a format, and the file path when `output` is a valid path.

Dependencies

- .pdf output requires a full LaTeX installation on the local computer.
- .png output requires the `webshot2` package.
- .html self-contained files require the `base64enc` package.

LaTeX preamble

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

When rendering a document from Quarto or Rmarkdown directly to PDF, tinytable will populate the LaTeX preamble automatically with all the required packages. For standalone LaTeX documents, these commands should be inserted in the preamble manually:

Note: Your document will fail to compile to PDF in Quarto if you enable caching and you use tinytable due to missing LaTeX headers. To avoid this problem, set the option `#| cache: false` for the chunk(s) where you use tinytable.

```
\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}}
```

Global options

Options can be set with `options()` and change the default behavior of tinytable. For example:

```
options(tinytable_tt_digits = 4)
tt(head(iris))
```

You can set options in a script or via `.Rprofile`. Note: be cautious with `.Rprofile` settings as they may affect reproducibility.

Default values for function arguments:

Nearly all of the package's functions retrieve their default values from global options. This allows you to set defaults once and apply them to all tables without needing to specify them each time. For example, to fix the `digits` argument of the `tt()` function globally, call:

```
options(tinytable_tt_digits = 4)
```

In addition, some more specific options are available to control the behavior of the package in specific contexts.

- `tinytable_html_mathjax`: Insert MathJax scripts (warning: may conflict if MathJax is loaded elsewhere)
- `tinytable_pdf_clean`: Delete temporary and log files for pdf output in `save_tt()`
- `tinytable_color_name_normalization`: Enable/disable automatic color name processing (default: TRUE). When enabled, R color names recognized by `col2rgb()` are converted to hex format for consistent rendering across HTML, LaTeX, and Typst formats. If R color conversion fails, LaTeX color names are used as fallback. Colors explicitly supplied as hex values with `"#"` prefix are passed through unchanged. Set to FALSE to disable processing and pass color names unchanged.

Quarto:

The `format_tt(quarto=TRUE)` argument enables Quarto data processing with some limitations:

1. The `\QuartoMarkdownBase64{}` LaTeX macro may not process references and markdown as expected
2. Quarto processing may conflict with `tinytable` styling/formatting

Options:

- `tinytable_quarto_disable_processing`: Disable Quarto cell processing

Example of Quarto-specific code in cells:

```
x <- data.frame(Math = "x^2^", Citation = "@Lovelace1842")
fn <- function(z) sprintf("<span data-qmd='%s'></span>", z)
tt(x) |> format_tt(i = 1, fn = fn)
```

For more details on Quarto table processing: <https://quarto.org/docs/authoring/tables.html#disabling-quarto-table-processing>

Examples

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

fn <- file.path(tempdir(), "test.html")
tt(x) |> save_tt(fn, overwrite = TRUE)

library(tinytable)
filename <- file.path(tempdir(), "table.tex")
tt(mtcars[1:4, 1:4]) |> save_tt(filename)
```

style_tt

Style a Tiny Table

Description

Style a Tiny Table

Alias for `style_tt()`

Usage

```
style_tt(
  x,
  i = NULL,
  j = NULL,
  bold = FALSE,
  italic = FALSE,
  monospace = FALSE,
  smallcap = FALSE,
```

```

    underline = FALSE,
    strikeout = FALSE,
    color = NULL,
    background = NULL,
    fontsize = NULL,
    align = NULL,
    alignv = NULL,
    colspan = NULL,
    rowspan = NULL,
    indent = NULL,
    line = NULL,
    line_color = "black",
    line_width = 0.1,
    line_trim = NULL,
    finalize = NULL,
    ...
)

tt_style(
  x,
  i = NULL,
  j = NULL,
  bold = FALSE,
  italic = FALSE,
  monospace = FALSE,
  smallcap = FALSE,
  underline = FALSE,
  strikeout = FALSE,
  color = NULL,
  background = NULL,
  fontsize = NULL,
  align = NULL,
  alignv = NULL,
  colspan = NULL,
  rowspan = NULL,
  indent = NULL,
  line = NULL,
  line_color = "black",
  line_width = 0.1,
  line_trim = NULL,
  finalize = NULL,
  ...
)

```

Arguments

- | | |
|---|---|
| x | A table object created by <code>tt()</code> . |
| i | Numeric vector, logical matrix, string, or unquoted expression. |

	• Numeric vector: Row indices where the styling should be applied. Can be a single value or a vector. • Logical matrix: A matrix with the same number of rows and columns as <code>x</code>. <code>i=0</code> is the header, and negative values are higher level headers. Row indices refer to rows <i>after</i> the insertion of row labels by <code>group_tt()</code>, when applicable. • String: Table components "caption", "colnames", "groupi" (row group labels), "~groupi" (non-group rows), "groupj" (column group labels), "notes". • Unquoted expression: When supplying an unquoted expression, it is first evaluated in the calling environment, then in the data frame passed to <code>tt()</code>.
<code>j</code>	Column indices where the styling should be applied. Can be: • Integer vectors indicating column positions. • Character vector indicating column names. • A single string specifying a Perl-style regular expression used to match column names. • Unquoted expression: Non-standard evaluation is supported. When supplying an unquoted expression, it is first evaluated in the calling environment, then in an environment that includes the columns of the original data passed to <code>tt()</code>, and <code>groupi</code> indices. See examples below.
<code>bold</code>	Logical; if TRUE, text is styled in bold.
<code>italic</code>	Logical; if TRUE, text is styled in italic.
<code>monospace</code>	Logical; if TRUE, text is styled in monospace font.
<code>smallcap</code>	Logical; if TRUE, text is styled in small caps. In Markdown output format, text is converted to uppercase.
<code>underline</code>	Logical; if TRUE, text is underlined.
<code>strikeout</code>	Logical; if TRUE, text has a strike through line.
<code>color</code>	Text color. Colors are standardized across output formats and can be specified as: • Hex codes: "#CC79A7", "#FF0000", "#123ABC" • R color names: Any color recognized by R, such as "red", "blue", "forest-green", "lightblue" • Extended color names: 749+ named colors from the LaTeX xcolor package (see <code>tinytable:::latex_colors</code> for the full list) • LaTeX color blending (LaTeX output only): "white!80!blue", "red!50", "green!20!red"
<code>background</code>	Background color. Same color specification options as the <code>color</code> parameter. Can be NULL for default color.
<code>fontsize</code>	Font size in em units. Can be NULL for default size.
<code>align</code>	A single character or a string with a number of characters equal to the number of columns in <code>j</code> . Valid characters include 'c' (center), 'l' (left), 'r' (right), 'd' (decimal). Decimal alignment is only available in LaTeX via the <code>siunitx</code> package. The width of columns is determined by the maximum number of digits to the left and to the right in all cells specified by <code>i</code> and <code>j</code> .

<code>alignv</code>	A single character specifying vertical alignment. Valid characters include <code>`t'</code> (top), <code>`m'</code> (middle), <code>`b'</code> (bottom).
<code>colspan</code>	Number of columns a cell should span. <code>i</code> and <code>j</code> must be of length 1.
<code>rowspan</code>	Number of rows a cell should span. <code>i</code> and <code>j</code> must be of length 1.
<code>indent</code>	Text indentation in em units. Positive values only.
<code>line</code>	String determines if solid lines (rules or borders) should be drawn around the cell, row, or column. • <code>"t"</code>: top • <code>"b"</code>: bottom • <code>"l"</code>: left • <code>"r"</code>: right • Can be combined such as: <code>"lbt"</code> to draw borders at the left, bottom, and top.
<code>line_color</code>	Color of the line. See the <code>color</code> argument for details.
<code>line_width</code>	Width of the line in em units (default: 0.1).
<code>line_trim</code>	String specifying line trimming. Acceptable values: <code>"l"</code> (left), <code>"r"</code> (right), <code>"lr"</code> (both sides). When specified, shortens the lines by 0.8pt on the specified side(s). Default: <code>NULL</code> (no trimming).
<code>finalize</code>	A function applied to the table object at the very end of table-building, for post-processing. For example, the function could use regular expressions to add LaTeX commands to the text version of the table hosted in <code>x@table_string</code> , or it could programmatically change the caption in <code>x@caption</code> .
<code>...</code>	extra arguments are ignored

Details

This function applies styling to a table created by `tt()`. It allows customization of text style (bold, italic, monospace), text and background colors, font size, cell width, text alignment, column span, and indentation. The function also supports passing native instructions to LaTeX (`tabulararray`) and HTML (`bootstrap`) formats.

Value

An object of class `tt` representing the table.

Markdown limitations

Markdown is a text-only format that only supports these styles: italic, bold, ~~strikeout~~. The width argument is also unavailable. These limitations exist because there is no standard markdown syntax for the other styling options.

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.

Word limitations

Word tables only support these styles: italic, bold, strikethrough. The width argument is also unavailable. Moreover, the `style_tt()` function cannot be used to style headers inserted by the `group_tt()` function; instead, you should style the headers directly in the header definition using markdown syntax: `group_tt(i = list("*italic header*" = 2))`. These limitations are due to the fact that we create Word documents by converting a markdown table to .docx via the Pandoc software, which requires going through a text-only intermediate format.

Examples

```
if (knitr::is_html_output()) options(tinytable_print_output = "html")

library(tinytable)

tt(mtcars[1:5, 1:6])

# Alignment
tt(mtcars[1:5, 1:6]) |>
  style_tt(j = 1:5, align = "lcccr")

# Colors and styles
tt(mtcars[1:5, 1:6]) |>
  style_tt(i = 2:3, background = "black", color = "orange", bold = TRUE)

# column selection with `j`
tt(mtcars[1:5, 1:6]) |>
  style_tt(j = 5:6, background = "pink")

tt(mtcars[1:5, 1:6]) |>
  style_tt(j = "drat|wt", background = "pink")

tt(mtcars[1:5, 1:6]) |>
  style_tt(j = c("drat", "wt"), background = "pink")

tt(mtcars[1:5, 1:6], theme = "empty") |>
  style_tt(
    i = 2, j = 2,
    colspan = 3,
    rowspan = 2,
    align = "c",
    alignv = "m",
    color = "white",
    background = "black",
    bold = TRUE)

tt(mtcars[1:5, 1:6], theme = "empty") |>
  style_tt(
    i = 0:3,
    j = 1:3,
    line = "tblr",
    line_width = 0.4,
```

```

    line_color = "teal")

tt(mtcars[1:5, 1:6], theme = "striped") |>
  style_tt(
    i = c(2, 5),
    j = 3,
    strikeouts = TRUE,
    fontsize = 0.7)

# Non-standard evaluation (NSE)
dat <- data.frame(
  w = c(143002.2092, 201399.181, 100188.3883),
  x = c(1.43402, 201.399, 0.134588),
  y = as.Date(c(897, 232, 198), origin = "1970-01-01"),
  z = c(TRUE, TRUE, FALSE)
)
tt(dat) |>
  style_tt(i = w > 150000, j = c("w", "x"),
    color = "white", background = "black")

tt(mtcars[1:5, 1:6]) |>
  theme_html(class = "table table-dark table-hover")

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},
"
tt(mtcars[1:5, 1:4], theme = "empty") |>
  theme_latex(inner = inner)

# Style group rows and non-group rows
dat <- data.frame(x = 1:6, y = letters[1:6])
dat |>
  tt() |>
  group_tt(i = list("Group A" = 3)) |>
  style_tt(i = "groupi", background = "lightblue") |>
  style_tt(i = "~groupi", background = "lightgray")

# unquote expressions
dat <- mtcars[1:10,]
dat <- dat[order(dat$am),]
tt(dat) |>
  subset(mpg > 20) |>
  group_tt(am)

# style elements: captions and colnames

```

```

notes <- list(
  "x" = "Hello world",
  "a" = "Bacon ipsum dolor amet kevin t-bone porchetta.")
tt(head(iris),
  width = .5,
  caption = "This is a Caption Example.",
  notes = notes) |>
style_tt(2, 2, background = "pink", rowspan = 2, colspan = 2,
  alignv = "m", align = "c", line = "tblr") |>
style_tt("colnames", italic = TRUE) |>
style_tt("caption", smallcap = TRUE)

```

style_vector

*Style a Vector with Text Formatting***Description**

Style a Vector with Text Formatting

Usage

```

style_vector(
  x,
  output = "html",
  bold = FALSE,
  italic = FALSE,
  monospace = FALSE,
  smallcap = FALSE,
  underline = FALSE,
  strikeout = FALSE,
  color = NULL,
  fontsize = NULL,
  indent = NULL
)

```

Arguments

x	A vector to be styled.
output	Output format for styling. One of "html", "latex", "typst", "markdown", "ansi". Defaults to "html".
bold	Logical; if TRUE, text is styled in bold. Must be of length 1 or length(x).
italic	Logical; if TRUE, text is styled in italic. Must be of length 1 or length(x).
monospace	Logical; if TRUE, text is styled in monospace font. Must be of length 1 or length(x).
smallcap	Logical; if TRUE, text is styled in small caps. In Markdown output format, text is converted to uppercase. Must be of length 1 or length(x).

underline	Logical; if TRUE, text is underlined. Must be of length 1 or length(x).
strikeout	Logical; if TRUE, text has a strike through line. Must be of length 1 or length(x).
color	Text color. Must be of length 1 or length(x). There are several ways to specify colors: • HTML: – Hex code composed of # and 6 characters, ex: #CC79A7. – Keywords: black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua • LaTeX: – Hex code composed of # and 6 characters, ex: "#CC79A7". – Keywords: black, blue, brown, cyan, darkgray, gray, green, lightgray, lime, magenta, olive, orange, pink, purple, red, teal, violet, white, yellow. – Color blending using xcolor, ex: white!80!blue, green!20!red. – Color names with luminance levels from the ninecolors package.
fontsize	Font size in em units. Must be of length 1 or length(x). Can be NULL for default size.
indent	Text indentation in em units. Must be of length 1 or length(x). Positive values only.

Details

This function applies styling to a vector. It allows customization of text style (bold, italic, monospace), text color, font size, and text decorations (underline, strikeout). The styling is applied element-wise to the vector. Vectors are coerced with `as.character()` before styling.

Value

A character vector with applied styling.

Examples

```
# Basic styling
style_vector(c("Hello", "World"), bold = TRUE, color = "red")

# Different styles per element
style_vector(
  c("Bold text", "Italic text", "Monospace"),
  bold = c(TRUE, FALSE, FALSE),
  italic = c(FALSE, TRUE, FALSE),
  monospace = c(FALSE, FALSE, TRUE)
)

# Single style applied to all elements
style_vector(c("A", "B", "C"), color = "blue", fontsize = 1.2)
```

subset.tinytable	<i>Subsetting a tinytable object</i>
------------------	--------------------------------------

Description

Return subsets tinytable which meet conditions.

Usage

```
## S3 method for class 'tinytable'  
subset(x, subset, select, drop = FALSE, ...)
```

Arguments

x	object to be subsetted.
subset	logical expression indicating elements or rows to keep: missing values are taken as false.
select	expression, indicating columns to select from a data frame.
drop	passed on to [indexing operator.
...	further arguments to be passed to or from other methods.

theme_default	<i>Default theme for TinyTable</i>
---------------	------------------------------------

Description

Default theme for TinyTable

Usage

```
theme_default(x, ...)
```

Arguments

x	A tinytable object.
...	Additional arguments are ignored.

Value

A modified tinytable object.

theme_empty	<i>Theme for a void table</i>
-------------	-------------------------------

Description

This function calls styles and formatting applied to a `tinytable` object up to that point in the pipeline.

Usage

```
theme_empty(x, ...)
```

Arguments

x	A <code>tinytable</code> object.
...	Additional arguments are ignored.

theme_grid	<i>Grid theme with borders around all cells</i>
------------	---

Description

Grid theme with borders around all cells

Usage

```
theme_grid(x, ...)
```

Arguments

x	A <code>tinytable</code> object.
...	Additional arguments (ignored).

Value

A modified `tinytable` object.

theme_html	<i>HTML-specific styles and options</i>
------------	---

Description

HTML-specific styles and options

Usage

```
theme_html(
  x,
  engine = get_option("tinytable_html_engine", default = NULL),
  i = NULL,
  j = NULL,
  class = get_option("tinytable_html_class", default = NULL),
  css = get_option("tinytable_html_css", default = NULL),
  css_rule = get_option("tinytable_html_css_rule", default = NULL),
  portable = get_option("tinytable_html_portable"),
  tabulator_columns = get_option("tinytable_html_tabulator_columns"),
  tabulator_css_rule = get_option("tinytable_html_tabulator_css_rule"),
  tabulator_layout = get_option("tinytable_html_tabulator_layout", default =
    "fitDataTable"),
  tabulator_options = get_option("tinytable_html_tabulator_options"),
  tabulator_pagination = get_option("tinytable_html_tabulator_pagination"),
  tabulator_search = get_option("tinytable_html_tabulator_search"),
  tabulator_stylesheet = get_option("tinytable_html_tabulator_stylesheet"),
  ...
)
```

Arguments

x	A tinytable object.
engine	Character string specifying the HTML engine: "tinytable", "bootstrap", or "tabulator".
i	Row indices.
j	Column indices.
class	String. HTML table class.
css	Character vector. CSS style declarations.
css_rule	String. Complete CSS rules.
portable	Logical. Sets whether to create portable HTML output with embedded Javascript, CSS, and base64-encoded images.
tabulator_columns	Custom column definitions.
tabulator_css_rule	Complete CSS rules.

tabulator_layout	Character string. Table layout algorithm for column sizing. Default is "fitDataTable". Available options: "fitDataTable", "fitData", "fitDataFill", "fitDataStretch", "fitColumns".
tabulator_options	Custom Tabulator.js configuration options.
tabulator_pagination	Logical or numeric vector. Pagination settings for large tables. • NULL (default): Preserves existing pagination settings, does not change previous configuration • FALSE: Explicitly disable pagination • TRUE: Enable pagination with automatic page sizes (10, 25, 50, 100, 250) filtered by row count • Numeric vector: First element is default page size, full vector provides page size options
tabulator_search	Character or NULL. Search functionality position. • NULL (default): Preserves existing search settings, does not change previous configuration • "top": Adds search box above the table • "bottom": Adds search box below the table
tabulator_stylesheet	Character string. CSS stylesheet theme for Tabulator.js tables. Default is "bootstrap5". Available options: "default", "simple", "midnight", "modern", "site", "site_dark", "bootstrap3", "bootstrap4", "bootstrap5", "semanticui", "bulma", "materialize", or a custom HTTP URL starting with "http".
...	Additional arguments are ignored.

theme_latex

LaTeX-Specific Theme for tinytable

Description

This function provides comprehensive LaTeX-specific theming and configuration options for `tinytable` objects. It allows customization of LaTeX environments, table layout, multipage behavior, resizing, and placement within LaTeX documents.

Usage

```
theme_latex(
  x,
  inner = NULL,
  outer = NULL,
  environment = get_option("tinytable_latex_environment", default = NULL),
  environment_table = get_option("tinytable_latex_environment_table", default = TRUE),
```

```

    multipage = get_option("tinytable_latex_multipage", default = FALSE),
    rowhead = get_option("tinytable_latex_rowhead", 0L),
    rowfoot = get_option("tinytable_latex_rowfoot", 0L),
    resize_width = get_option("tinytable_latex_resize_width", 1),
    resize_direction = get_option("tinytable_latex_resize_direction", default = NULL),
    placement = get_option("tinytable_latex_placement", NULL),
    preamble = NULL,
    engine = NULL,
    ...
)

```

Arguments

x	A tinytable object to apply LaTeX theming to.
inner	Character string specifying inner tabularray options. These options control the internal formatting of the table (e.g., column alignment, spacing). Will be added to any existing inner options. Default is NULL.
outer	Character string specifying outer tabularray options. These options control the external formatting around the table. Will be added to any existing outer options. Default is NULL.
environment	Character string specifying the LaTeX table environment to use. Options are: • "tblr" - Standard tabularray table (default) • "talltblr" - Tall tabularray table for tables that may break across pages • "longtblr" - Long tabularray table for multi-page tables • "tabular" - Basic LaTeX tabular environment without tabularray features Default is controlled by tinytable_latex_environment option.
environment_table	Logical indicating whether to wrap the table in a table environment. When FALSE, only the core table structure is output without the surrounding table wrapper. Automatically set to FALSE when environment = "longtblr". Default is controlled by tinytable_latex_environment_table option.
multipage	Logical indicating whether to enable multipage table functionality. When TRUE, automatically switches to longtblr environment and sets appropriate options for tables that span multiple pages. Default is controlled by tinytable_latex_multipage option.
rowhead	Integer specifying the number of header rows to repeat on each page in multipage tables. Only valid with longtblr environment. Default is controlled by tinytable_latex_rowhead option.
rowfoot	Integer specifying the number of footer rows to repeat on each page in multipage tables. Only valid with longtblr environment. Default is controlled by tinytable_latex_rowfoot option.
resize_width	Numeric value between 0.01 and 1.0 specifying the target width as a fraction of \linewidth when resizing tables. Only used when resize_direction is specified. Default is controlled by tinytable_latex_resize_width option.

resize_direction	Character string specifying how to resize tables that are too wide or too narrow. Options are: • "down" - Only shrink tables wider than <code>\linewidth</code> • "up" - Only expand tables narrower than <code>\linewidth</code> • "both" - Resize all tables to exactly <code>resize_width * \linewidth</code> Default is controlled by <code>tinytable_latex_resize_direction</code> option.
placement	Character string specifying LaTeX float placement options for the table environment (e.g., "h", "t", "b", "p", "H"). Only used when <code>environment_table = TRUE</code>. Default is controlled by <code>tinytable_latex_placement</code> option.
preamble	Logical value specifying whether to include LaTeX preamble packages. If not NULL, overrides the table's preamble setting.
engine	Character string specifying the LaTeX engine to use for PDF compilation. Options are "xelatex", "pdflatex", or "lualatex". If not NULL, overrides the table's engine setting.
...	Additional arguments (currently unused).

Details

The function provides fine-grained control over LaTeX table output through several mechanisms:

Environment Selection: Different LaTeX environments offer different capabilities:

- `tblr`: Modern tabularray syntax with full styling support
- `talltblr`: Like `tblr` but optimized for tall tables
- `longtblr`: Supports page breaks and repeated headers/footers
- `tabular`: Basic LaTeX syntax, limited styling but maximum compatibility

Multipage Tables: When `multipage = TRUE` or when `rowhead/rowfoot` are specified, the function automatically switches to `longtblr` environment and disables the table wrapper. This allows tables to break across pages while maintaining headers and footers.

Resizing: The resize functionality uses LaTeX's `\resizebox` command to automatically adjust table width based on content and page constraints. This is particularly useful for tables with many columns.

Tabularray Options: Inner and outer options directly control tabularray formatting. Inner options affect cell content and spacing, while outer options control the table's relationship with surrounding text.

Value

A modified `tinytable` object with LaTeX-specific theming applied.

See Also

[tt\(\)](#), [style_tt\(\)](#), [save_tt\(\)](#)

theme_markdown	<i>Markdown theme with optional ANSI color support and grid customization</i>
----------------	---

Description

Markdown theme with optional ANSI color support and grid customization

Usage

```
theme_markdown(
  x,
  ansi = FALSE,
  style = NULL,
  vline = NULL,
  hline = NULL,
  hline_header = NULL,
  ...
)
```

Arguments

x	A tinytable object.
ansi	Logical. If TRUE, enables ANSI color codes for grid styling. Default is FALSE.
style	Character. Markdown style format. Can be "grid" or "gfm". Default is NULL.
vline	Logical. Enable/disable vertical lines. Default is TRUE.
hline	Logical. Enable/disable horizontal lines. Default is TRUE.
hline_header	Logical. Enable/disable the special header separator line below column names. Default is TRUE.
...	Additional arguments (ignored).

Details

When `ansi = TRUE`, colors and text styling (bold, italic, strikeout, underline) are applied using ANSI escape sequences for terminal display. ANSI colors require a terminal or application that supports ANSI escape sequences. Common supported terminals include: Terminal.app (macOS), iTerm2 (macOS), Windows Terminal, most Linux terminals, RStudio Console, and VS Code terminal. Colors may not display correctly in basic text editors or older terminals.

Value

A modified tinytable object.

theme_revealjs	<i>RevealJS presentation theme</i>
----------------	------------------------------------

Description

RevealJS presentation theme

Usage

```
theme_revealjs(
  x,
  css = get_option("tinytable_revealjs_css", default = "light"),
  fontsize = get_option("tinytable_revealjs_fontsize", default = 0.8),
  fontsize_caption = get_option("tinytable_revealjs_fontsize_caption", default = 1)
)
```

Arguments

x	A tinytable object.
css	String. CSS theme: "light" (default) or "dark".
fontsize	Numeric. Font size multiplier for table content.
fontsize_caption	Numeric. Font size multiplier for table captions.

Value

A modified tinytable object.

theme_rotate	<i>Rotate table theme (LaTeX and Typst only)</i>
--------------	--

Description

Rotate table theme (LaTeX and Typst only)

Usage

```
theme_rotate(
  x,
  angle = get_option("tinytable_rotate_angle", default = 90),
  ...
)
```

Arguments

- x A tinytable object.
- angle Numeric. Rotation angle in degrees (0-360).
- ... Additional arguments (ignored).

Value

A modified tinytable object.

theme_stripped	<i>Striped theme with alternating row colors</i>
----------------	--

Description

Striped theme with alternating row colors

Usage

theme_stripped(x, ...)

Arguments

- x A tinytable object.
- ... Additional arguments (ignored).

Value

A modified tinytable object.

theme_tt	<i>Deprecated: Use format-specific theme functions instead</i>
----------	--

Description

DEPRECATED: The theme_tt() function has been deprecated. Please use the format-specific or style-specific theme functions instead.

Usage

theme_tt(x, theme, ...)

Arguments

- x deprecated
- theme deprecated
- ... Additional arguments

Value

Throws an informative error message

theme_typst	<i>Typst-specific styles and options</i>
-------------	--

Description

Typst-specific styles and options

Usage

```
theme_typst(
  x,
  figure = get_option("tinytable_typst_figure", default = TRUE),
  align_figure = get_option("tinytable_typst_align_figure", NULL),
  ...
)
```

Arguments

x	A tinytable object.
figure	Logical, whether to wrap the table in a Typst figure environment and block.
align_figure	Character string indicating horizontal alignment: "l", "c", or "r". Defaults to <code>get_option("tinytable_theme_placement_horizontal", NULL)</code> . When NULL, uses default center alignment.
...	Additional arguments.

tt	<i>Draw a Tiny Table</i>
----	--------------------------

Description

The `tt` function renders a table in different formats with various styling options: HTML, Markdown, LaTeX, Word, PDF, PNG, or Typst. The table can be customized with additional functions:

- `style_tt()`: style fonts, colors, alignment, etc.
- `format_tt()`: format numbers, dates, strings, etc.
- `group_tt()`: row or column group labels.
- `save_tt()`: save the table to a file or return the table as a string.
- `print()`: print to a specific format, ex: `print(x, "latex")`
- `theme_*()` functions apply a collection of format-specific or visual transformations to a `tinytable`.

`tinytable` attempts to determine the appropriate way to print the table based on interactive use, RStudio availability, and output format in RMarkdown or Quarto documents. Users can call `print(x, output="markdown")` to print the table in a specific format. Alternatively, they can set a global option: `options("tinytable_print_output"="markdown")`

Usage

```
tt(x, ...)

## Default S3 method:
tt(
  x,
  digits = get_option("tinytable_tt_digits", default = NULL),
  caption = get_option("tinytable_tt_caption", default = NULL),
  notes = get_option("tinytable_tt_notes", default = NULL),
  width = get_option("tinytable_tt_width", default = NULL),
  height = get_option("tinytable_tt_height", default = NULL),
  theme = get_option("tinytable_tt_theme", default = "default"),
  colnames = get_option("tinytable_tt_colnames", default = TRUE),
  rownames = get_option("tinytable_tt_rownames", default = FALSE),
  escape = get_option("tinytable_tt_escape", default = FALSE),
  ...
)
```

Arguments

<code>x</code>	A data frame, data table, or tibble to be rendered as a table.
<code>...</code>	Additional arguments are ignored
<code>digits</code>	Number of significant digits to keep for numeric variables. When <code>digits</code> is an integer, <code>tt()</code> calls <code>format_tt(x, digits = digits)</code> before proceeding to draw the table. Note that this will apply all default argument values of <code>format_tt()</code> , such as replacing NA by <code>"</code> ". Users who need more control can use the <code>format_tt()</code> function instead.
<code>caption</code>	A string that will be used as the caption of the table. This argument should <i>not</i> be used in Quarto or Rmarkdown documents. In that context, please use the appropriate chunk options.
<code>notes</code>	Notes to append to the bottom of the table. This argument accepts several different inputs: • Single string insert a single note: <code>"blah blah"</code> • Multiple strings insert multiple notes sequentially: <code>list("Hello world", "Foo bar")</code> • A named list inserts a list with the name as superscript: <code>list("a" = list("Hello World"))</code> • A named list with positions inserts markers as superscripts inside table cells: <code>list("a" = list(i = 0:1, j = 2, text = "Hello World"))</code>
<code>width</code>	Table or column width. • Single numeric value smaller than or equal to 1 determines the full table width, in proportion of line width. • Numeric vector of length equal to the number of columns in <code>x</code> determines the width of each column, in proportion of line width. If the sum of width exceeds 1, each element is divided by <code>sum(width)</code>. This makes the table full-width with relative column sizes.

height	Row height in em units. Single numeric value greater than zero that determines the row height spacing.
theme	Function or string. • String: grid, revealjs, rotate, striped, empty • Function: Applied to the tinytable object.
colnames	TRUE, FALSE, or "label". If "label", use the <code>attr(x\$col, "label")</code> attribute if available and fall back on column names otherwise.
rownames	Logical. If TRUE, rownames are included as the first column
escape	Logical. If TRUE, escape special characters in the table. Equivalent to <code>format_tt(tt(x), escape = TRUE)</code> .

Value

An object of class `tt` representing the table.

The table object has S4 slots which hold information about the structure of the table. For example, the `table@group_index_i` slot includes the row indices for grouping labels added by `group_tt()`.

Warning: Relying on or modifying the contents of these slots is strongly discouraged. Their names and contents could change at any time, and the `tinytable` developers do not consider changes to the internal structure of the output object to be a "breaking change" for versioning or changelog purposes.

Dependencies

- .pdf output requires a full LaTeX installation on the local computer.
- .png output requires the `webshot2` package.
- .html self-contained files require the `base64enc` package.

LaTeX preamble

`tinytable` uses the `tabularray` package from your LaTeX distribution to draw tables. `tabularray`, in turn, uses the special `tblr`, `talltblr`, and `longtblr` environments.

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

Note: Your document will fail to compile to PDF in Quarto if you enable caching and you use `tinytable` due to missing LaTeX headers. To avoid this problem, set the option `#| cache: false` for the chunk(s) where you use `tinytable`.

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

Markdown limitations

Markdown is a text-only format that only supports these styles: italic, bold, strikethrough. The width argument is also unavailable. These limitations exist because there is no standard markdown syntax for the other styling options.

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.

Word limitations

Word tables only support these styles: italic, bold, strikethrough. The width argument is also unavailable. Moreover, the `style_tt()` function cannot be used to style headers inserted by the `group_tt()` function; instead, you should style the headers directly in the header definition using markdown syntax: `group_tt(i = list("*italic header*" = 2))`. These limitations are due to the fact that we create Word documents by converting a markdown table to .docx via the Pandoc software, which requires going through a text-only intermediate format.

Tabulator (interactive tables)

Experimental Feature: The `Tabulator.js` integration is experimental and the API may change in future versions.

The `Tabulator.js` library provides powerful interactive table features including sorting, filtering, pagination, data export, and real-time editing capabilities. This theme customizes the appearance and behavior of `Tabulator` tables.

Features:

- Sorting and filtering of all columns
- Pagination with configurable page sizes
- Search functionality across all columns
- Multiple CSS themes and custom styling
- Real-time data export options
- Accessibility features (ARIA compliant)

Limitations:

- Limited `style_tt()` support (only `align` and `alignv`)
- Row-based formatting (`format_tt()` with `i` argument) not supported
- Global stylesheets affect all tables in multi-table documents
- Date formatting uses Luxon tokens, not R's `strptime` format
- Boolean formatting requires `format_tt()` with `bool` argument for custom display

Global options

Options can be set with `options()` and change the default behavior of `tinytable`. For example:

```
options(tinytable_tt_digits = 4)
tt(head(iris))
```

You can set options in a script or via `.Rprofile`. Note: be cautious with `.Rprofile` settings as they may affect reproducibility.

Default values for function arguments:

Nearly all of the package's functions retrieve their default values from global options. This allows you to set defaults once and apply them to all tables without needing to specify them each time. For example, to fix the `digits` argument of the `tt()` function globally, call:

```
options(tinytable_tt_digits = 4)
```

In addition, some more specific options are available to control the behavior of the package in specific contexts.

- `tinytable_html_mathjax`: Insert MathJax scripts (warning: may conflict if MathJax is loaded elsewhere)
- `tinytable_pdf_clean`: Delete temporary and log files for pdf output in `save_tt()`
- `tinytable_color_name_normalization`: Enable/disable automatic color name processing (default: TRUE). When enabled, R color names recognized by `col2rgb()` are converted to hex format for consistent rendering across HTML, LaTeX, and Typst formats. If R color conversion fails, LaTeX color names are used as fallback. Colors explicitly supplied as hex values with `"#"` prefix are passed through unchanged. Set to FALSE to disable processing and pass color names unchanged.

Quarto:

The `format_tt(quarto=TRUE)` argument enables Quarto data processing with some limitations:

1. The `\QuartoMarkdownBase64{}` LaTeX macro may not process references and markdown as expected
2. Quarto processing may conflict with `tinytable` styling/formatting

Options:

- `tinytable_quarto_disable_processing`: Disable Quarto cell processing

Example of Quarto-specific code in cells:

```
x <- data.frame(Math = "x^2^", Citation = "@Lovelace1842")
fn <- function(z) sprintf("<span data-qmd='%s'></span>", z)
tt(x) |> format_tt(i = 1, fn = fn)
```

For more details on Quarto table processing: <https://quarto.org/docs/authoring/tables.html#disabling-quarto-table-processing>

Order of Operations

When building tables, `tinytable` applies operations in this specific order:

1. `format_tt()`: Number and string formatting operations are applied first to ensure all data is properly formatted before any other operations.

2. `group_tt()`: Row and column grouping operations are applied to insert group headers and modify table structure.
3. `theme` argument in `tt()`: The theme specified in the `theme` argument is applied immediately when creating the table object.
4. `theme_*`() with delayed execution via `build_prepare()`: Theme functions that need to be applied after the table structure is finalized (such as those that depend on final row/column indices) are executed at this stage. Examples include `theme_default()` which applies default borders after groups are inserted. Internally, `build_prepare()` allows calling `style_tt()` with full knowledge of the output format and access to internal slots like `x@output`.
5. `style_tt()` and `theme_*`() with immediate execution: Direct styling calls and theme functions without delayed execution are applied in the order they appear in the code.
6. `theme_*`() with delayed execution via `build_finalize()`: Final theme operations that need to modify the rendered table string are applied last. Examples include `theme_rotate()` which wraps the entire table output. Internally, `build_finalize()` allows post-processing the rendered table in text format using regular expressions and string manipulations.

This order ensures that structural changes (grouping) happen before styling, and that operations requiring the final table structure are deferred appropriately. When conflicts arise between operations at the same level, "last applied wins" - the most recent operation takes precedence.

Examples

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

tt(x)

tt(x,
  theme = "striped",
  width = 0.5,
  caption = "Data about cars."
)

tt(x, notes = "Hello World!")

fn <- list(i = 0:1, j = 2, text = "Hello World!")
tab <- tt(x, notes = list("*" = fn))
print(tab, "latex")

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

# use variable labels stored in attributes as column names
dat = mtcars[1:5, c("cyl", "mpg", "hp")]
attr(dat$cyl, "label") <- "Cylinders"
attr(dat$mpg, "label") <- "Miles per Gallon"
attr(dat$hp, "label") <- "Horse Power"
tt(dat, colnames = "label")
```

Index

*** package**
 tinytable-package, 2

format_tt, 3
format_vector, 8

group_tt, 10

plot_tt, 13
plot_vector, 16
print.tinytable, 18

rbind2
 (rbind2, tinytable, tinytable-method),
 19
rbind2, tinytable, tinytable-method, 19

save_tt, 21
save_tt(), 36
style_tt, 23
style_tt(), 36
style_vector, 29
subset.tinytable, 31

theme_default, 31
theme_empty, 32
theme_grid, 32
theme_html, 33
theme_latex, 34
theme_markdown, 37
theme_revealjs, 38
theme_rotate, 38
theme_stripped, 39
theme_tt, 39
theme_typst, 40
tinytable (tinytable-package), 2
tinytable-package, 2
tt, 40
tt(), 36
tt_format (format_tt), 3
tt_group (group_tt), 10
tt_plot (plot_tt), 13
tt_save (save_tt), 21
tt_style (style_tt), 23