Package ‘sumExtras’

January 22, 2026

Title Extra Functions for 'gtsummary' Table Styling

Version 0.3.0

Description Provides additional convenience functions for 'gtsummary' (Sjoberg et al. (2021) <doi:10.32614/RJ-2021-053>) & 'gt' tables, including automatic variable labeling from dictionaries, standardized missing value display, and consistent formatting helpers for streamlined table styling workflows.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Suggests broom (>= 1.0.5), broom.helpers (>= 1.20.0), ggplot2, knitr, labelled, quarto, scales, survey, testthat (>= 3.0.0), tibble

VignetteBuilder knitr

Config/testthat/edition 3

Imports dplyr, gt (>= 0.9.0), gtsummary (>= 1.7.0), purrr, rlang

URL `https://github.com/kyleGrealis/sumExtras`

BugReports `https://github.com/kyleGrealis/sumExtras/issues`

NeedsCompilation no

Author Kyle Grealis [aut, cre] (ORCID: <`https://orcid.org/0000-0002-9223-8854`>), Raymond Balise [ctb] (ORCID: <`https://orcid.org/0000-0002-9856-5901`>), Daniel Maya [ctb] (ORCID: <`https://orcid.org/0000-0002-0164-7768`>)

Maintainer Kyle Grealis <`kyleGrealis@proton.me`>

Repository CRAN

Date/Publication 2026-01-22 10:00:02 UTC

Contents

`add_auto_labels` *Add automatic labels from dictionary to a gtsummary table*

Description

Automatically apply variable labels from a dictionary or label attributes to `tbl_summary`, `tbl_svysummary`, or `tbl_regression` objects. Intelligently preserves manual label overrides set in the original table call while applying dictionary labels or reading label attributes from data. The dictionary can be passed explicitly or will be searched for in the calling environment. If no dictionary is found, the function will attempt to read label attributes from the underlying data.

Usage

```
add_auto_labels(tbl, dictionary)
```

Arguments

`tbl`	A gtsummary table object created by `tbl_summary()`, `tbl_svysummary()`, or `tbl_regression()`.
`dictionary`	A data frame or tibble with `Variable` and `Description` columns. If not provided (missing), the function will search for a `dictionary` object in the calling environment. If no dictionary is found, the function will attempt to read label attributes from the data. Set to `NULL` explicitly to skip dictionary search and only use attributes.

Details

Label Priority Hierarchy:

The function applies labels according to this priority (highest to lowest):

1. **Manual labels** - Labels set via `label = list(...)` in `tbl_summary()` etc. are always preserved
2. **Dictionary vs Attributes** - Controlled by `options(sumExtras.preferDictionary)`:
    - If `TRUE`: Dictionary labels take precedence over attribute labels

- If `FALSE` (default): Attribute labels take precedence over dictionary labels

3. **Default** - If no label source is available, uses variable name

Dictionary Format:

The dictionary must be a data frame with columns:

- `Variable`: Character column with exact variable names from datasets
- `Description`: Character column with human-readable labels

Label Attributes:

The function reads label attributes from data using `attr(data$var, "label")`, following the same label convention used by **haven**, **Hmisc**, and **ggplot2 4.0+**.

Your data may already have labels from various sources - imported from statistical software packages, set by other R packages, added manually, or from collaborative projects. This function discovers and applies them seamlessly within gtsummary tables.

Because sumExtras uses native R's attribute storage, labels work across any package that respects the `"label"` attribute convention, including:

- **ggplot2 4.0+** - automatic axis and legend labels
- **gt** - table label support
- **Hmisc** - label utilities and display functions

This approach requires zero package dependencies and is fully compatible with the labelled package if you choose to use it, but does not require it.

Implementation Note:

This function relies on internal gtsummary structures (`tbl$call_list`, `tbl$inputs`, `tbl$table_body`) to detect manually set labels. While robust error handling is implemented, major updates to gtsummary may require corresponding updates to sumExtras. Requires gtsummary >= 1.7.0.

Value

A gtsummary table object with labels applied. Manual labels set via `label = list(...)` in the original table call are always preserved.

Options

Set `options(sumExtras.preferDictionary = TRUE)` to prioritize dictionary labels over label attributes when both are available. Default is `FALSE`, which prioritizes attributes over dictionary labels.

See Also

- `apply_labels_from_dictionary()` for setting label attributes on data for ggplot2/other packages
- `gtsummary::modify_table_body()` for advanced table customization

Other labeling functions: `apply_labels_from_dictionary()`

Examples

```
# Create a dictionary
my_dict <- tibble::tribble(
  ~Variable, ~Description,
  "age", "Age at Enrollment",
  "trt", "Treatment Group",
  "grade", "Tumor Grade"
)

# Basic usage: pass dictionary explicitly
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, grade)) |>
  add_auto_labels(dictionary = my_dict)

# Automatic dictionary search (dictionary in environment)
dictionary <- my_dict
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, grade)) |>
  add_auto_labels()  # Finds dictionary automatically

# Working with pre-labeled data (no dictionary needed)
labeled_data <- gtsummary::trial
attr(labeled_data$age, "label") <- "Patient Age (years)"
attr(labeled_data$marker, "label") <- "Marker Level (ng/mL)"

labeled_data |>
  gtsummary::tbl_summary(include = c(age, marker)) |>
  add_auto_labels()  # Reads from label attributes

# Manual overrides always win
gtsummary::trial |>
  gtsummary::tbl_summary(
    by = trt,
    include = c(age, grade),
    label = list(age ~ "Custom Age Label")  # Manual override
  ) |>
  add_auto_labels(dictionary = my_dict)  # grade gets dict label, age keeps manual

# Control priority with options
options(sumExtras.preferDictionary = TRUE)  # Dictionary over attributes

# Data has both dictionary and attributes
labeled_trial <- gtsummary::trial
attr(labeled_trial$age, "label") <- "Age from Attribute"
dictionary <- tibble::tribble(
  ~Variable, ~Description,
  "age", "Age from Dictionary"
)

labeled_trial |>
  gtsummary::tbl_summary(include = age) |>
  add_auto_labels()  # Uses "Age from Dictionary" (option = TRUE)
```

`add_group_colors` *Add background colors to group headers with automatic gt conversion*

Description

Convenience function that adds background colors to variable group headers and converts the table to gt. This is a terminal operation that combines `get_group_rows()`, `gtsummary::as_gt()`, and `gt::tab_style()` into a single pipeable function.

For text formatting (bold/italic), use `add_group_styling()` before calling this function. This composable design keeps each function focused on doing one thing well.

Usage

```
add_group_colors(tbl, color = "#E8E8E8")
```

Arguments

`tbl`	A gtsummary table object with variable group headers created by `gtsummary::add_variable_group_he`
`color`	Background color for group headers. Default `"#E8E8E8"` (light gray). Can be any valid CSS color (hex code, color name, rgb(), etc.).

Details

This function:

1. Identifies group header rows with `get_group_rows()`
2. Converts the table to gt with `gtsummary::as_gt()`
3. Applies background color using `gt::tab_style()`

Since this function converts to gt, it should be used as the final styling step in your pipeline. Apply all gtsummary functions (like `modify_caption()`, `modify_footnote()`, etc.) and text formatting with `add_group_styling()` before calling `add_group_colors()`.

Value

A gt table object with colored group headers. **Note:** This is a terminal operation that converts to gt. You cannot pipe to additional gtsummary functions after calling this function.

See Also

- `add_group_styling()` for text formatting only (stays gtsummary)
- `get_group_rows()` for identifying group header rows
- `gtsummary::add_variable_group_header()` for creating variable groups
- `gt::tab_style()` for additional gt-specific styling

Examples

```
# Basic usage - text formatting then color
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras() |>
  gtsummary::add_variable_group_header(
    header = "Patient Characteristics",
    variables = age:stage
  ) |>
  add_group_styling() |>
  add_group_colors()

# Custom color - light blue
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras() |>
  gtsummary::add_variable_group_header(
    header = "Baseline Characteristics",
    variables = age:marker
  ) |>
  add_group_styling() |>
  add_group_colors(color = "#E3F2FD")

# Bold only formatting with custom color
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras() |>
  gtsummary::add_variable_group_header(
    header = "Clinical Measures",
    variables = marker:stage
  ) |>
  add_group_styling(format = "bold") |>
  add_group_colors(color = "#FFF9E6")

# Multiple group headers
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras() |>
  gtsummary::add_variable_group_header(
    header = "Demographics",
    variables = age
  ) |>
  gtsummary::add_variable_group_header(
    header = "Disease Measures",
    variables = marker:response
  ) |>
  add_group_styling() |>
  add_group_colors(color = "#E8E8E8")
```

`add_group_styling` *Apply styling to variable group headers in gtsummary tables*

Description

Adds customizable formatting to variable group headers in gtsummary tables. Variable groups are created using `gtsummary::add_variable_group_header()` to organize variables into sections. This function enhances table readability by making group headers visually distinct from individual variable labels.

Usage

```
add_group_styling(tbl, format = c("bold", "italic"), indent_labels = 0L)
```

Arguments

`tbl`	A gtsummary table object (e.g., from `tbl_summary()`, `tbl_regression()`)
`format`	Character vector specifying text formatting. Options include `"bold"`, `"italic"`, or both. Default is `c("bold", "italic")`.
`indent_labels`	Integer specifying indentation level (in spaces) for variable labels under group headers. Default is `0L` (left-aligned). Set to `4L` to preserve gtsummary's default group indentation, or use any non-negative integer for custom spacing.

Details

The function targets rows where `row_type == 'variable_group'` and applies the specified text formatting to the label column. This is particularly useful for tables with multiple sections or stratified analyses where clear visual hierarchy improves interpretation.

By default, variable labels are left-aligned (`indent_labels = 0L`) to distinguish them from categorical levels and statistics. Use `indent_labels = 4L` to preserve the default gtsummary behavior where grouped variables are indented under their group headers.

Value

A gtsummary table object with specified formatting applied to variable group headers

See Also

- `gtsummary::modify_table_styling()` for general table styling options
- `gtsummary::add_variable_group_header()` for creating variable group headers

Examples

```
# Default formatting (bold and italic)
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker, grade)) |>
  gtsummary::add_variable_group_header(
    header = "Patient Characteristics",
    variables = age:grade
  ) |>
  add_group_styling()

# Bold only
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker)) |>
  gtsummary::add_variable_group_header(
    header = "Demographics",
    variables = age:marker
  ) |>
  add_group_styling(format = "bold")

# Multiple group headers
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  gtsummary::add_variable_group_header(
    header = "Demographics",
    variables = age
  ) |>
  gtsummary::add_variable_group_header(
    header = "Clinical Measures",
    variables = marker:response
  ) |>
  add_group_styling()

# Custom indentation for grouped variables
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker)) |>
  gtsummary::add_variable_group_header(
    header = "Patient Measures",
    variables = age:marker
  ) |>
  add_group_styling(indent_labels = 4L)  # Variables indented under header
```

`apply_labels_from_dictionary` *Apply variable labels from dictionary to data as attributes*

Description

Sets variable label attributes on data columns using a dictionary. This enables cross-package integration with tools that read label attributes, including ggplot2 4.0+ (automatic axis labels), gt (label support), and Hmisc. Labels are stored as the `'label'` attribute on each column, following the informal convention used across the R ecosystem.

This function is designed for workflows where you need labels to persist with your data for use in plots, descriptive tables, or other visualizations beyond gtsummary tables.

Usage

```
apply_labels_from_dictionary(data, dictionary, overwrite = TRUE)
```

Arguments

`data`	A data frame or tibble to add label attributes to
`dictionary`	A data frame or tibble with `Variable` and `Description` columns matching the format used by `add_auto_labels()`
`overwrite`	Logical. If `TRUE` (default), overwrites existing label attributes. If `FALSE`, preserves existing labels and only adds new ones.

Details

This function provides a bridge from sumExtras' dictionary-based labeling system to the broader R ecosystem. Key use cases:

- **ggplot2 4.0+**: Automatic axis and legend labels from attributes
- **Cross-package workflows**: One dictionary for tables (gtsummary) and plots (ggplot2)
- **Documentation**: Labels visible in RStudio data viewer
- **Interoperability**: Compatible with gt, Hmisc, and other label-aware packages

Only variables present in both the data and dictionary will receive label attributes. Dictionary entries for non-existent variables are silently ignored.

Implementation: The R Ecosystem Label Convention:

This function uses **native R's `attr()` function** to store labels in the `"label"` attribute, following the same approach as haven, Hmisc, and ggplot2 4.0+. This standardized convention enables seamless integration across the R ecosystem.

Because labels are stored as simple base R attributes (not in a special package-specific format), they work transparently with any package that respects the `"label"` attribute:

- **Dictionary-to-Attribute Bridge**: Converts your dictionary's `Description` column into standard R label attributes
- **Zero Dependencies**: Uses only base R, no special packages required
- **Transparent & Simple**: Users can inspect labels with `attr(data$var, "label")`
- **Ecosystem Compatible**: Works with ggplot2, gt, gtsummary, Hmisc, and beyond

The benefits of this approach are that labels remain portable with your data, work across multiple R packages without version constraints, and integrate naturally with the broader R ecosystem's labeling conventions.

Value

The input data with label attributes attached to matching columns. Original data is returned unmodified except for added/updated attributes.

See Also

- `add_auto_labels()` for applying labels to gtsummary tables
- `labelled::var_label()` for an alternative way to set label attributes
- `ggplot2::labs()` for manual plot labeling

Other labeling functions: `add_auto_labels()`

Examples

```
# Create a dictionary
my_dict <- tibble::tribble(
  ~Variable, ~Description,
  "age", "Age at Enrollment (years)",
  "marker", "Marker Level (ng/mL)",
  "trt", "Treatment Group",
  "grade", "Tumor Grade"
)

# Apply labels to data
trial_labeled <- gtsummary::trial |>
  apply_labels_from_dictionary(my_dict)

# Now labels work automatically in gtsummary
trial_labeled |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker, grade))

# And in ggplot2 4.0+ (automatic axis labels!)
if (requireNamespace("ggplot2", quietly = TRUE) &&
    utils::packageVersion("ggplot2") >= "4.0.0") {
  library(ggplot2)
  trial_labeled |>
    ggplot(aes(x = age, y = marker, color = factor(trt))) +
    geom_point()  # Axes and legend automatically labeled!
}

# Check that labels were applied
attr(trial_labeled$age, "label")  # "Age at Enrollment (years)"

# Preserve existing labels
trial_partial <- gtsummary::trial
attr(trial_partial$age, "label") <- "Existing Age Label"

trial_partial |>
  apply_labels_from_dictionary(my_dict, overwrite = FALSE)

attr(trial_partial$age, "label")  # Still "Existing Age Label"
attr(trial_partial$marker, "label")  # "Marker Level (ng/mL)" (was added)
```

`clean_table` *Standardize missing value display across all gtsummary table types*

Description

Improves table readability by replacing various missing value representations with a consistent "—" symbol. This makes it easier to distinguish between actual data and missing/undefined values in summary tables, creating a cleaner and more professional appearance.

Works seamlessly with all gtsummary table types, including stacked tables (`tbl_strata`) and survey-weighted summaries (`tbl_svysummary`). Automatically handles tables with or without the standard `var_type` column.

Usage

```
clean_table(tbl)
```

Arguments

`tbl`	A gtsummary table object (e.g., from `tbl_summary()`, `tbl_svysummary()`, `tbl_regression()`, or `tbl_strata()`)

Details

The function uses `gtsummary::modify_table_body()` to transform character columns and replace common missing value patterns with "—":

- `"0 (NA%)"` - No events occurred and percentages cannot be calculated
- `"NA (NA)"` - Completely missing data for both count and percentage
- `"0 (0%)"` - Zero counts with zero percentage
- `"0% (0.000)"` - Zero percentage with decimal precision
- `"NA (NA, NA)"` - Missing data with confidence intervals
- `"NA, NA"` - Missing paired values (e.g., median and IQR)

This standardization makes tables more scannable and reduces visual clutter from various "empty" data representations.

Note: The function checks for the presence of `var_type` column before applying `modify_missing_symbol()`. This allows it to work seamlessly with `tbl_strata` objects which use `var_type_1`, `var_type_2`, etc. instead of `var_type`.

Value

A gtsummary table object with standardized missing value display

See Also

- `gtsummary::modify_table_body()` for general table body modifications
- `extras()` which includes `clean_table()` in its styling pipeline

Examples

```
# Basic usage - clean missing values in summary table
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  clean_table()

# Often used as part of a styling pipeline
# Create a test dictionary for add_auto_labels():
dictionary <- tibble::tribble(
  ~Variable, ~Description,
  'age', 'Age at enrollment',
  'stage', 'T Stage',
  'grade', 'Grade',
  'response', 'Tumor Response'
)
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  add_auto_labels() |>
  extras() |>
  clean_table()

# Works with regression tables too
lm(age ~ trt + grade, data = gtsummary::trial) |>
  gtsummary::tbl_regression() |>
  clean_table()
```

`extras` *Add standard styling and formatting to gtsummary tables*

Description

Applies a consistent set of formatting options to gtsummary tables including overall column, bold labels, clean headers, and optional p-values. Streamlines the common workflow of adding multiple formatting functions. The function always succeeds by applying what works and warning about unsupported features.

Usage

```
extras(
  tbl,
  pval = TRUE,
  overall = TRUE,
  last = FALSE,
  .args = NULL,
  .add_p_args = NULL
)
```

Arguments

`tbl`	A gtsummary table object (e.g., from `tbl_summary()`, `tbl_regression()`)
`pval`	Logical indicating whether to add p-values. Default is `TRUE`. When `TRUE`, uses gtsummary's default statistical tests (Kruskal-Wallis for continuous variables with 3+ groups, chi-square for categorical variables).
`overall`	Logical indicating whether to add overall column
`last`	Logical indicating if Overall column should be last. Aligns with default from `gtsummary::add_overall()`.
`.args`	Optional list of arguments to use instead of individual parameters. When provided, overrides `pval`, `overall`, and `last` arguments.
`.add_p_args`	Optional named list of arguments to pass to `gtsummary::add_p()`. Allows customization of statistical tests and p-value formatting. User-provided arguments override the default arguments (`pvalue_fun` and `test.args`). See `gtsummary::add_p()` documentation for available arguments.

Details

The function applies the following modifications:

- Bolds variable labels for emphasis (all table types)
- Removes the "Characteristic" header label (all table types)
- Adds an "Overall" column (only stratified summary tables)
- Optionally adds p-values (only stratified summary tables)
- Applies `clean_table()` styling (all table types)

The function automatically detects whether the input table is stratified (has a `by` argument) and what type of table it is (tbl_summary, tbl_regression, tbl_strata, etc.).

For tables that don't support overall columns or p-values (non-stratified tables, regression tables, or stacked tables), the function will issue a warning and continue by applying only the universally supported features (bold_labels and modify_header). This ensures the function always succeeds rather than failing midway through the pipeline.

If any individual formatting step fails (e.g., due to unexpected table structure), the function will issue a warning and continue without that feature. This provides robustness while keeping you informed of what was skipped.

Value

A gtsummary table object with standard formatting applied

Table Type Support

The function applies features based on table type and stratification:

- **bold_labels()** and **modify_header()**: Work on all table types
- **add_overall()**: Only works on stratified summary tables (tbl_summary with by)
- **add_p()**: Only works on stratified summary tables (tbl_summary with by)

Full feature support: tbl_summary and tbl_svysummary with by argument

Partial support (basic formatting only): tbl_regression, tbl_strata, and non-stratified tables. When applied to these table types and overall/pval = TRUE, the function warns about unsupported features but applies the formatting that works.

See Also

- `gtsummary::add_overall()` for adding overall columns
- `gtsummary::add_p()` for adding p-values
- `clean_table()` for additional table styling

Examples

```
# With p-values (default)
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras()

# Using .args list
extra_args <- list(pval = TRUE, overall = TRUE, last = FALSE)
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras(.args = extra_args)

# Without p-values
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras(pval = FALSE)

# Customize add_p() behavior
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  extras(.add_p_args = list(
    test = list(all_continuous() ~ "t.test"),
    pvalue_fun = ~ gtsummary::style_pvalue(.x, digits = 2)
  ))

# Chain with other functions
# Create required dictionary first
dictionary <- tibble::tribble(
  ~Variable, ~Description,
  'record_id', 'Participant ID',
  'age', 'Age at enrollment',
  'sex', 'Biological sex'
)
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt) |>
  add_auto_labels() |>
  extras(pval = TRUE) |>
  add_group_styling()
```

`get_group_rows` *Get row numbers of variable group headers for gt styling*

Description

Extracts the row indices of variable group headers from a gtsummary table. This is useful for applying background colors or other gt-specific styling after converting a gtsummary table to gt with `as_gt()`.

Usage

```
get_group_rows(tbl)
```

Arguments

`tbl`	A gtsummary table object with variable group headers created by `gtsummary::add_variable_group_he`

Details

Variable group headers are identified by `row_type == 'variable_group'` in the table body. The returned row numbers can be used with `gt::tab_style()` to apply styling like background colors after converting to a gt table.

This function should be called BEFORE converting the table with `as_gt()`, as the row type information is only available in gtsummary table objects.

Value

An integer vector of row numbers where variable_group headers are located

See Also

- `add_group_styling()` for applying text formatting to group headers
- `gtsummary::add_variable_group_header()` for creating variable groups
- `gt::tab_style()` for applying gt-specific styling

Examples

```
# Create table with variable groups
my_tbl <- gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker, grade, stage)) |>
  gtsummary::add_variable_group_header(
    header = "Demographics",
    variables = age
  ) |>
  gtsummary::add_variable_group_header(
    header = "Clinical",
    variables = marker:stage
  ) |>
  add_group_styling()

# Get group row numbers before conversion
group_rows <- get_group_rows(my_tbl)

# Convert to gt and apply gray background
my_tbl |>
  gtsummary::as_gt() |>
  gt::tab_style(
    style = gt::cell_fill(color = "#E8E8E8"),
    locations = gt::cells_body(rows = group_rows)
  )
```

`theme_gt_compact` *Apply compact JAMA-style theme to gt tables*

Description

Applies a compact table theme to gt tables that matches the 'jama' theme from gtsummary. This ensures visual consistency when mixing gtsummary tables (using `theme_gtsummary_compact("jama")`) with regular gt tables in the same document. The theme reduces padding, adjusts font sizes, and applies JAMA journal styling conventions.

Usage

```
theme_gt_compact(tbl)
```

Arguments

`tbl`	A gt table object created with `gt::gt()`

Details

This function replicates the visual appearance of `gtsummary::theme_gtsummary_compact("jama")` for use with regular gt tables. Key styling includes:

- Reduced font size (13px) for compact appearance
- Minimal padding (1px) on all row types
- Bold column headers and table titles
- Hidden top and bottom table borders
- Consistent spacing that matches JAMA journal standards

Value

A gt table object with compact JAMA-style formatting applied

See Also

- `gtsummary::theme_gtsummary_compact()` for gtsummary table themes
- `gtsummary::set_gtsummary_theme()` for setting global gtsummary themes
- `gt::tab_options()` for additional gt table styling options

Examples

```
# Basic usage with a data frame
mtcars |>
  head() |>
  gt::gt() |>
  theme_gt_compact()

# Combine with other gt functions
mtcars |>
  head() |>
  gt::gt() |>
  gt::tab_header(title = "Vehicle Data") |>
  theme_gt_compact()

# Use alongside gtsummary tables for consistency
# Set gtsummary theme first
gtsummary::set_gtsummary_theme(gtsummary::theme_gtsummary_compact("jama"))

# Then both tables will have matching appearance
summary_table <- gtsummary::trial |>
  gtsummary::tbl_summary()

data_table <- gtsummary::trial |>
  head() |>
  gt::gt() |>
  theme_gt_compact()
```

`use_jama_theme` *Apply JAMA Compact Theme to gtsummary Tables*

Description

Sets the global gtsummary theme to the JAMA (Journal of the American Medical Association) compact style. This is the recommended theme for use with sumExtras functions, providing professional medical journal formatting with reduced padding and consistent styling. The theme remains active for the entire R session or until changed with another theme.

Usage

```
use_jama_theme()
```

Details

The JAMA compact theme implements formatting standards used by the Journal of the American Medical Association, making it ideal for:

- Medical research manuscripts and reports
- Clinical trial summaries
- Academic publications requiring AMA style
- Professional presentations with clean, compact tables

Key formatting features include:

- Reduced font size (13px) for compact appearance
- Minimal cell padding (1px) to maximize information density
- Bold column headers and variable labels
- Clean borders following JAMA style guidelines
- Consistent alignment and spacing

The function checks for the gtsummary package and will stop with an informative error if it is not installed. The theme is applied globally and will affect all gtsummary tables created after calling this function, including `tbl_summary()`, `tbl_regression()`, `tbl_cross()`, `tbl_strata()`, and related functions.

For visual consistency with regular gt tables, use `theme_gt_compact()` which replicates the same styling for non-gtsummary tables.

Value

Invisibly returns the theme list object from `gtsummary::theme_gtsummary_compact("jama")`. The theme is applied globally via `gtsummary::set_gtsummary_theme()`, affecting all subsequent gtsummary tables created in the session. A message is printed confirming the theme application.

See Also

- `theme_gt_compact` for applying JAMA-style formatting to regular gt tables
- `extras` for standard sumExtras table formatting
- `gtsummary::theme_gtsummary_compact()` for other compact theme options
- `gtsummary::set_gtsummary_theme()` for setting custom themes
- `gtsummary::reset_gtsummary_theme()` for resetting to default theme

Examples

```
# Apply theme at the start of your analysis
use_jama_theme()

# All subsequent gtsummary tables will use JAMA formatting
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt)

# Works with all gtsummary table types
lm(age ~ trt + grade, data = gtsummary::trial) |>
  gtsummary::tbl_regression()

# Combine with sumExtras styling functions
use_jama_theme()
gtsummary::trial |>
  gtsummary::tbl_summary(by = trt, include = c(age, marker, stage)) |>
  extras() |>
  add_group_styling()

# Reset to default theme if needed
gtsummary::reset_gtsummary_theme()
```

Index