Package 'summata'

October 19, 2026

Type Package

Title Publication-Ready Summary Tables and Forest Plots

Version 0.13.0

Description A comprehensive framework for descriptive statistics and regression analysis that produces publication-ready tables and forest plots. Provides a unified interface from descriptive statistics through multivariable modeling, with support for linear models, generalized linear models, Cox proportional hazards, and mixed-effects models. Also includes univariable screening, multivariate regression, model comparison, and export to multiple formats including PDF, DOCX, PPTX, 'LaTeX', HTML, and RTF. Built on 'data.table' for computational efficiency.

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://phmcc.codefloe.page/summata/, https://codefloe.com/phmcc/summata, https://github.com/phmcc/summata

BugReports https://github.com/phmcc/summata/issues

Depends R (>= 4.2.0)

Imports data.table, survival, ggplot2, stats, grDevices, parallel, tools, utils

Suggests MASS, coxme, flextable, grid, knitr, lme4, MuMIn, officer, ragg, ResourceSelection, reformulas, rmarkdown, systemfonts, testthat, tinytex, withr, xtable

SystemRequirements All optional. A 'LaTeX' distribution providing pdflatex or xelatex, and kpsewhich, for PDF output from table2pdf(); pdfcrop for auto-sized PDF output; Ghostscript for the optional font embedding in forestsave(). Each feature reports the absence of its tool and the remainder of the package is unaffected.

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Paul Hsin-ti McClelland [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-3119-6531>)

Maintainer Paul Hsin-ti McClelland <PaulHMcClelland@protonmail.com>

Repository CRAN

Date/Publication 2026-10-19 02:30:02 UTC

Contents

autoforest . . . 3
clintrial . . . 6
clintrial_labels . . . 8
compfit . . . 9
coxforest . . . 14
desctable . . . 21
fit . . . 29
forestsave . . . 43
fullfit . . . 46
glmforest . . . 56
lmforest . . . 63
m2dt . . . 70
multifit . . . 74
multiforest . . . 88
recdims . . . 93
survtable . . . 94
table2docx . . . 102
table2html . . . 109
table2pdf . . . 115
table2pptx . . . 124
table2rtf . . . 131
table2tex . . . 138
tablesave . . . 145
uniforest . . . 148
uniscreen . . . 153

Index **167**

`autoforest` *Create Forest Plot with Automatic Model Detection*

Description

A convenience wrapper function that automatically detects the input type and routes to the appropriate specialized forest plot function. This eliminates the need to remember which forest function to call for different model types or analysis objects, making it ideal for exploratory analysis and rapid prototyping.

Usage

```
autoforest(x, data = NULL, title = NULL, ...)
```

Arguments

`x` One of the following:

- A fitted model object: `glm`, `lm`, `coxph`, *etc.*
- A `fit_result` object from `fit()`
- A `fullfit_result` object from `fullfit()`
- A `uniscreen_result` object from `uniscreen()`
- A `multifit_result` object from `multifit()`

`data` Data frame or data.table containing the original data. Required when `x` is a raw model object. Ignored when `x` is a result object from `fit()`, `fullfit()`, `uniscreen()`, or `multifit()` since these contain embedded data.

`title` Character string for plot title. If `NULL` (default), an appropriate title is generated based on the detected model type:

- Cox models: "Cox Proportional Hazards Model"
- Logistic regression: "Logistic Regression Model"
- Poisson regression: "Poisson Regression Model"
- Linear regression: "Linear Regression Model"
- Uniscreen results: "Univariable [Type] Screening"
- Multifit results: "Multivariate [Type] Analysis"

`...` Additional arguments passed to the specific forest plot function. Common arguments include:

labels Named character vector for variable labels

digits Number of decimal places for estimates (default 2)

p_digits Number of decimal places for *p*-values (default 3)

conf_level Confidence level for intervals (default 0.95)

show_n Logical, show sample sizes (default `TRUE`)

show_events Logical, show event counts (default `TRUE` for survival/binomial)

qc_footnote Logical, show model QC stats in footer (default `TRUE`)

zebra_stripes Logical, alternating row shading (default `TRUE`)

indent_groups Logical, indent factor levels (default FALSE)

color Color for point estimates

table_width Proportion of width for table (default 0.6)

plot_width, plot_height Explicit dimensions

units Dimension units: "in", "cm", or "mm"

See the documentation for the specific forest function for all available options.

Details

This function provides a convenient wrapper around the specialized forest plot functions, automatically routing to the appropriate function based on the model class or result type. All parameters are passed through to the underlying function, so the full range of options remains available.

For model-specific advanced features, individual forest functions may be called directly.

Automatic Detection Logic:

The function uses the following priority order for detection:

1. **uniscreen results**: Detected by class "uniscreen_result" or presence of attributes outcome, predictors, model_type, and model_scope = "Univariable". Routes to uniforest().
2. **multifit results**: Detected by presence of attributes predictor, outcomes, model_type, and raw_data. Routes to multiforest().
3. **Cox models**: Classes coxph or clogit. Routes to coxforest().
4. **GLM models**: Class glm. Routes to glmforest().
5. **Linear models**: Class lm (but not glm). Routes to lmforest().

Value

A ggplot object containing the complete forest plot. The plot can be:

- Displayed directly: print(plot)
- Saved to file: forestsave(plot, "forest.pdf")
- Further customized with **ggplot2** functions

The returned object includes an attribute "rec_dims" accessible via attr(plot, "rec_dims"), which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if plot_width or plot_height are not specified.

See Also

glmforest for GLM forest plots, coxforest for Cox model forest plots, lmforest for linear model forest plots, uniforest for univariable screening forest plots, multiforest for multi-outcome forest plots, fit for single-model regression, fullfit for combined univariable/multivariable regression, uniscreen for univariable screening, multifit for multi-outcome analysis, forestsave for saving with recommended dimensions

Other visualization functions: coxforest(), forestsave(), glmforest(), lmforest(), multiforest(), recdims(), uniforest()

Examples

```
data(clintrial)
data(clintrial_labels)
library(survival)

# Create example model
glm_model <- glm(surgery ~ age + sex + bmi + smoking,
                 family = binomial, data = clintrial)

# Example 1: Logistic regression model
p <- autoforest(glm_model, data = clintrial)
# Automatically detects GLM and routes to glmforest()



# Example 2: Cox proportional hazards model
cox_model <- coxph(Surv(os_months, os_status) ~ age + sex + treatment + stage,
                   data = clintrial)

plot2 <- autoforest(cox_model, data = clintrial)
# Automatically detects coxph and routes to coxforest()

# Example 3: Linear regression model
lm_model <- lm(biomarker_x ~ age + sex + bmi + treatment, data = clintrial)

plot3 <- autoforest(lm_model, data = clintrial)
# Automatically detects lm and routes to lmforest()

# Example 4: With custom labels and formatting options
plot4 <- autoforest(
    cox_model,
    data = clintrial,
    labels = clintrial_labels,
    title = "Prognostic Factors for Overall Survival",
    zebra_stripes = TRUE,
    indent_groups = TRUE
)

# Example 5: From fit() result - data and labels extracted automatically
fit_result <- fit(
    data = clintrial,
```

```
    outcome = "surgery",
    predictors = c("age", "sex", "bmi", "treatment"),
    labels = clintrial_labels
)

plot5 <- autoforest(fit_result)

# Save with recommended dimensions
forestsave(plot5, file.path(tempdir(), "forest.pdf"))
```

`clintrial` *Simulated Clinical Trial Dataset*

Description

A simulated dataset from a hypothetical multi-center oncology clinical trial comparing two experimental drugs against control. Designed to demonstrate the full capabilities of descriptive and regression analysis functions.

Usage

```
clintrial
```

Format

A data frame with 850 observations and 32 variables:

patient_id Unique patient identifier (character)

age Age at enrollment in years (numeric: 18-90)

sex Biological sex (factor: Female, Male)

race Self-reported race (factor: White, Black, Asian, Other)

ethnicity Hispanic ethnicity (factor: Non-Hispanic, Hispanic)

bmi Body mass index in kg/m^2 (numeric)

smoking Smoking history (factor: Never, Former, Current)

hypertension Hypertension diagnosis (factor: No, Yes)

diabetes Diabetes diagnosis (factor: No, Yes)

ecog ECOG performance status (factor: 0, 1, 2, 3)

creatinine Baseline creatinine in mg/dL (numeric)

hemoglobin Baseline hemoglobin in g/dL (numeric)

biomarker_x Serum biomarker A in ng/mL (numeric)

biomarker_y Serum biomarker B in U/L (numeric)

site Enrolling site (factor: Site Alpha through Site Kappa)

grade Tumor grade (factor: Well/Moderately/Poorly differentiated)

stage Disease stage at diagnosis (factor: I, II, III, IV)

treatment Randomized treatment (factor: Control, Drug A, Drug B)

surgery Surgical resection (factor: No, Yes)

any_complication Any post-operative complication (factor: No, Yes)

wound_infection Post-operative wound infection (factor: No, Yes)

icu_admission ICU admission required (factor: No, Yes)

readmission_30d Hospital readmission within 30 days (factor: No, Yes)

pain_score Pain score at discharge (numeric: 0-10)

recovery_days Days to functional recovery (numeric)

los_days Hospital length of stay in days (numeric)

ae_count Adverse event count (integer). Overdispersed count suitable for negative binomial or quasipoisson regression.

fu_count Follow-up visit count (integer). Equidispersed count suitable for standard Poisson regression.

pfs_months Progression-Free Survival Time (months)

pfs_status Progression or Death Event

os_months Overall survival time in months (numeric)

os_status Death indicator (numeric: 0=censored, 1=death)

Details

This dataset includes realistic correlations between variables: - Survival is worse with higher stage, ECOG, age, and biomarker_x - Treatment effects show Drug B > Drug A > Control - `ae_count` is overdispersed (variance > mean) for negative binomial demos - `fu_count` is equidispersed (variance $\approx$ mean) for Poisson demos - Approximately 2% of values are missing at random - Median follow-up is approximately 30 months

Source

Simulated data for demonstration purposes

See Also

Other sample data: `clintrial_labels`

Examples

```
data(clintrial)
data(clintrial_labels)

# Descriptive statistics by treatment arm
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex", "stage", "ecog",
```

```
                    "biomarker_x", "Surv(os_months, os_status)"),
        labels = clintrial_labels)


# Poisson regression for equidispersed counts
fit(clintrial,
    outcome = "fu_count",
    predictors = c("age", "stage", "treatment"),
    model_type = "glm",
    family = "poisson",
    labels = clintrial_labels)

# Negative binomial for overdispersed counts
fit(clintrial,
    outcome = "ae_count",
    predictors = c("age", "treatment", "diabetes"),
    model_type = "negbin",
    labels = clintrial_labels)

# Complete analysis pipeline
fullfit(clintrial,
        outcome = "Surv(os_months, os_status)",
        predictors = c("age", "sex", "stage", "grade", "ecog",
                       "smoking", "biomarker_x", "biomarker_y", "treatment"),
        method = "screen",
        p_threshold = 0.20,
        model_type = "coxph",
        labels = clintrial_labels)
```

`clintrial_labels` *Variable Labels for Clinical Trial Dataset*

Description

A named character vector providing descriptive labels for all variables in the clinical_trial dataset. Use with labels parameter in functions.

Usage

```
clintrial_labels
```

Format

Named character vector with 24 elements

See Also

clintrial

Other sample data: clintrial

---

compfit *Compare Multiple Regression Models*

Description

Fits multiple regression models and provides a comprehensive comparison table with model quality metrics, convergence diagnostics, and selection guidance. Computes a composite score combining multiple quality metrics to facilitate rapid model comparison and selection.

Usage

```
compfit(
  data,
  outcome,
  model_list,
  model_names = NULL,
  interactions_list = NULL,
  random = NULL,
  model_type = "auto",
  family = "binomial",
  conf_level = 0.95,
  p_digits = 3,
  include_coefficients = FALSE,
  scoring_weights = NULL,
  labels = NULL,
  number_format = NULL,
  verbose = NULL,
  ...
)
```

Arguments

data	Data frame or data.table containing the dataset.
outcome	Character string specifying the outcome variable. For survival analysis, use Surv() syntax (*e.g.,* "Surv(time, status)").
model_list	List of character vectors, each containing predictor names for one model. Can also be a single character vector to auto-generate nested models.
model_names	Character vector of names for each model. If NULL, uses "Model 1", "Model 2", *etc.* Default is NULL.

`interactions_list`
: List of character vectors specifying interaction terms for each model. Each element corresponds to one model in model_list. Use `NULL` for models without interactions. Use colon notation for interactions (*e.g.,* `c("age:treatment")`). If `NULL`, no interactions are added to any model. Default is `NULL`.

`random`
: Character string specifying the random-effects formula for mixed-effects models (`glmer`, `lmer`, `coxme`). Use standard `lme4`/`coxme` syntax, *e.g.,* `"(1|site)"` for random intercepts by site. This random effects formula is applied to all models in the comparison. Alternatively, random effects can be included directly in the predictor vectors within `model_list` using the same syntax, which allows different random effects structures across models. Default is `NULL`.

`model_type`
: Character string specifying model type. If `"auto"`, detects based on outcome. Options include:

  - `"auto"` - Automatically detect based on outcome type (default)
  - `"glm"` - Generalized linear model. Supports logistic, Poisson, Gamma, Gaussian via `family` parameter.
  - `"lm"` - Linear regression for continuous outcomes
  - `"coxph"` - Cox proportional hazards for survival analysis
  - `"negbin"` - Negative binomial regression for overdispersed counts (requires MASS package)
  - `"lmer"` - Mixed-effects linear regression for clustered continuous outcomes
  - `"glmer"` - Mixed-effects logistic regression for clustered categorical outcomes
  - `"coxme"` - Mixed-effects Cox regression for clustered time-to-event outcomes

`family`
: For GLM and GLMER models, specifies the error distribution and link function. Common options include:

  - `"binomial"` - Logistic regression for binary outcomes (default)
  - `"poisson"` - Poisson regression for count data
  - `"quasibinomial"` - Logistic with overdispersion
  - `"quasipoisson"` - Poisson with overdispersion
  - `"gaussian"` - Normal distribution (linear regression via GLM)
  - `"Gamma"` - Gamma for positive continuous data
  - `"inverse.gaussian"` - For positive, highly skewed data

  For negative binomial, use `model_type = "negbin"` instead. See `family` for all options.

`conf_level`
: Numeric confidence level for intervals. Default is 0.95.

`p_digits`
: Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

`include_coefficients`
: Logical. If TRUE, includes a second table with coefficient estimates. Default is FALSE.

`scoring_weights`
: Named list of scoring weights. Each weight should be between 0 and 1, and they should sum to 1. Available metrics depend on model type. If `NULL`, uses sensible defaults. See Details for available metrics.

`labels`
: Named character vector providing custom display labels for variables. Default is `NULL`.

`number_format`
: Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

  - `"us"` - Comma thousands, period decimal: `1,234.56` [default]
  - `"eu"` - Period thousands, comma decimal: `1.234,56`
  - `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
  - `"none"` - No thousands separator: `1234.56`

  Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

  When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

  ```
  options(summata.number_format = "eu")
  ```

`verbose`
: Logical. If `TRUE`, displays model fitting warnings (*e.g.,* singular fit, convergence issues). If `FALSE` (default), routine fitting messages are suppressed while unexpected warnings are preserved. When `NULL`, uses `getOption("summata.verbose", FALSE)`.

`...`
: Additional arguments passed to model fitting functions.

Details

This function fits all specified models and computes comprehensive quality metrics for comparison. It generates a Composite Model Score (CMS) that combines multiple metrics: lower AIC/BIC (information criteria), higher concordance (discrimination), and model convergence status.

For GLMs, McFadden's pseudo-R-squared is calculated as 1 - (logLik/logLik_null). For Cox models, the global *p*-value is that of the likelihood-ratio test of all coefficients. For linear mixed-effects models fitted by REML, AIC, BIC, and the global likelihood-ratio test are computed from maximum likelihood refits (`lme4::refitML()`), because REML likelihoods are not comparable between models with different fixed effects.

Models that fail to converge are flagged and penalized in the composite score.

Interaction Terms:

When `interactions_list` is provided, each element specifies the interaction terms for the corresponding model in `model_list`. This is particularly useful for testing whether adding interactions improves model fit:

- Use `NULL` for models without interactions
- Specify interactions using colon notation: `c("age:treatment", "sex:stage")`
- Main effects for all variables in interactions must be in the predictor list
- Common pattern: Compare main effects model vs model with interactions

Scoring weights can be customized based on model type:

- GLM: `"convergence"`, `"aic"`, `"concordance"`, `"pseudo_r2"`, `"brier"`
- Cox: `"convergence"`, `"aic"`, `"concordance"`, `"global_p"`
- Linear: `"convergence"`, `"aic"`, `"pseudo_r2"`, `"rmse"`

Default weights emphasize discrimination (concordance) and model fit (AIC).

The composite score is designed as a tool to quickly rank models by their quality metrics. It should be used alongside traditional model selection criteria rather than as a definitive model selection method.

Value

A data.table with class "compfit_result" containing:

Model Model name/identifier

CMS Composite Model Score for model selection (higher is better)

N Sample size

Events Number of events (for survival/logistic)

Predictors Number of predictors

Converged Whether model converged properly

AIC Akaike Information Criterion

BIC Bayesian Information Criterion

R^2 / Pseudo-R^2 McFadden pseudo-R-squared (GLM)

Concordance C-statistic (logistic/survival)

Brier Score Brier accuracy score (logistic)

Global *p* Overall model p-value

Attributes include:

models List of fitted model objects

coefficients Coefficient comparison table (if requested)

best_model Name of recommended model

See Also

`fit` for individual model fitting, `fullfit` for automated variable selection, `tablesave` for exporting tables

Other regression functions: `fit()`, `fullfit()`, `multifit()`, `print.compfit_result()`, `print.fit_result()`, `print.fullfit_result()`, `print.multifit_result()`, `print.uniscreen_result()`, `uniscreen()`

Examples

```
# Load example data
data(clintrial)
data(clintrial_labels)

# Information criteria assume a common sample, so the candidate predictors
# are restricted to complete cases before any comparison
comparison_data <- na.omit(
    clintrial[, c("readmission_30d", "os_months", "os_status", "age", "sex",
                  "smoking", "diabetes", "stage", "ecog", "grade",
                  "treatment")]
)

# Example 1: Compare nested logistic regression models
models <- list(
    base = c("age", "sex"),
    clinical = c("age", "sex", "smoking", "diabetes"),
    full = c("age", "sex", "smoking", "diabetes", "stage", "ecog")
)

comparison <- compfit(
    data = comparison_data,
    outcome = "readmission_30d",
    model_list = models,
    model_names = c("Base", "Clinical", "Full")
)
comparison



# Example 2: Compare Cox survival models
surv_models <- list(
    simple = c("age", "sex"),
    clinical = c("age", "sex", "stage", "grade")
)

surv_comparison <- compfit(
    data = comparison_data,
    outcome = "Surv(os_months, os_status)",
    model_list = surv_models,
    model_type = "coxph"
)
surv_comparison

# Example 3: Test the effect of adding an interaction term.
# Treatment efficacy differs by disease stage in these data, so the
# interaction is expected to improve the fit.
interaction_models <- list(
    main = c("age", "treatment", "stage"),
    interact = c("age", "treatment", "stage")
)
```

```
interaction_comp <- compfit(
    data = comparison_data,
    outcome = "Surv(os_months, os_status)",
    model_list = interaction_models,
    model_type = "coxph",
    model_names = c("Main Effects", "With Interaction"),
    interactions_list = list(
        NULL,
        c("treatment:stage")
    )
)
interaction_comp

# Example 4: Include coefficient comparison table
detailed <- compfit(
    data = comparison_data,
    outcome = "readmission_30d",
    model_list = models,
    include_coefficients = TRUE,
    labels = clintrial_labels
)

# Access coefficient table
coef_table <- attr(detailed, "coefficients")
coef_table

# Example 5: Access fitted model objects
fitted_models <- attr(comparison, "models")
names(fitted_models)

# Example 6: Get best model recommendation
best <- attr(comparison, "best_model")
cat("Recommended model:", best, "\n")
```

`coxforest` *Create Forest Plot for Cox Proportional Hazards Models*

Description

Generates a publication-ready forest plot that combines a formatted data table with a graphical representation of hazard ratios from a Cox proportional hazards survival model. The plot integrates variable names, group levels, sample sizes, event counts, hazard ratios with confidence intervals, *p*-values, and model diagnostics in a single comprehensive visualization designed for manuscripts and presentations.

Usage

```
coxforest(
  x,
  data = NULL,
  title = "Cox Proportional Hazards Model",
  effect_label = "Hazard Ratio",
  digits = 2,
  p_digits = 3,
  conf_level = 0.95,
  font_size = 1,
  annot_size = 3.88,
  header_size = 5.82,
  title_size = 23.28,
  plot_width = NULL,
  plot_height = NULL,
  table_width = 0.6,
  show_n = TRUE,
  show_events = TRUE,
  indent_groups = FALSE,
  condense_table = FALSE,
  bold_variables = FALSE,
  center_padding = 4,
  zebra_stripes = TRUE,
  ref_label = "reference",
  labels = NULL,
  color = "#8A61D8",
  qc_footer = TRUE,
  units = "in",
  number_format = NULL
)
```

Arguments

x	Either a fitted Cox model object (class coxph or coxme), a fit_result object from fit(), or a fullfit_result object from fullfit(). When a fit_result or fullfit_result is provided, the model, data, and labels are automatically extracted.
data	Data frame or data.table containing the original data used to fit the model. If NULL (default) and x is a model, the function attempts to extract data from the model object. If x is a fit_result, data is extracted automatically. Providing data explicitly is recommended when passing a model directly.
title	Character string specifying the plot title displayed at the top. Default is "Cox Proportional Hazards Model". Use descriptive titles for publication.
effect_label	Character string for the effect measure label on the forest plot axis. Default is "Hazard Ratio". Alternatives might include "HR" for space-constrained plots.
digits	Integer specifying the number of decimal places for hazard ratios and confidence intervals. Default is 2.

`p_digits` Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

`conf_level` Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals). The CI percentage is automatically displayed in column headers (*e.g.,* "90% CI" when `conf_level = 0.90`).

`font_size` Numeric multiplier controlling the base font size for all text elements. Default is 1.0.

`annot_size` Numeric value controlling the relative font size for data annotations. Default is 3.88.

`header_size` Numeric value controlling the relative font size for column headers. Default is 5.82.

`title_size` Numeric value controlling the relative font size for the main plot title. Default is 23.28.

`plot_width` Numeric value specifying the intended output width in specified `units`. Used for optimizing layout. Default is `NULL` (automatic). Recommended: 10-16 inches.

`plot_height` Numeric value specifying the intended output height in specified `units`. Default is `NULL` (automatic based on number of rows).

`table_width` Numeric value between 0 and 1 specifying the proportion of total plot width allocated to the data table. Default is 0.6 (60% table, 40% forest plot).

`show_n` Logical. If `TRUE`, includes a column showing group-specific sample sizes. Counts describe the observations used in fitting rather than every row supplied. Default is `TRUE`.

`show_events` Logical. If `TRUE`, includes a column showing the number of events (deaths, failures) for each group. Critical for survival analysis interpretation. Default is `TRUE`.

`indent_groups` Logical. If `TRUE`, indents factor levels under their parent variable, creating hierarchical structure. The "Group" column is hidden when `TRUE`. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses binary variables to show only the non-reference level (*e.g.,* "Yes" for Yes/No variables), with the variable name indicating the displayed level. This creates a more compact table for models with many binary predictors. Default is `FALSE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. If `FALSE` (default), variable names are displayed in plain text.

`center_padding` Numeric value specifying horizontal spacing between table and forest plot. Default is 4.

`zebra_stripes` Logical. If `TRUE`, applies alternating gray background shading to different variables for improved readability. Default is `TRUE`.

`ref_label` Character string to display for reference categories. Default is `"reference"`.

`labels` Named character vector providing custom display labels for variables. Example: `c(age = "Age (years)", stage = "Disease Stage")`. Default is `NULL`, in which case the labels attached by the function that produced `x` are used, where `x` is a **summata** result rather than a model object. Original variable names are used where neither is available.

`color` Character string specifying the color for hazard ratio point estimates in the forest plot. Default is `"#8A61D8"` (purple). Use hex codes or R color names.

`qc_footer` Logical. If `TRUE`, displays model quality control statistics in the footer (events analyzed, global log-rank *p*-value, concordance index, AIC). Default is `TRUE`.

`units` Character string specifying units for plot dimensions: `"in"` (inches), `"cm"`, or `"mm"`. Default is `"in"`.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

Details

Survival-Specific Features:

The Cox forest plot includes several survival analysis-specific components:

- **Event counts**: Number of events (deaths, failures) shown for each predictor category, critical for assessing statistical power
- **Hazard ratios**: Always exponentiated coefficients (never raw), interpreted as the multiplicative change in hazard
- **Log scale**: Forest plot uses log scale for HR (reference line at 1)
- **Model diagnostics**: Includes concordance (C-index), global log-rank test *p*-value, and AIC

Plot Components:

1. **Title**: Centered at top
2. **Data Table** (left): Contains:
   - Variable and Group columns
   - n: Sample sizes by group
   - Events: Event counts by group (critical for survival)
   - aHR (95% CI); *p*-value: Adjusted hazard ratios with CIs and *p*-values
3. **Forest Plot** (right):
   - Point estimates (squares sized by sample size)
   - 95% confidence intervals
   - Reference line at HR = 1
   - Log scale for hazard ratios

4. **Model Statistics** (footer):
    - Events analyzed (with total events and percentage)
    - Global log-rank test *p*-value
    - Concordance (C-index) with standard error
    - AIC

Interpreting Hazard Ratios:

- **HR = 1**: No effect on hazard (reference)
- **HR > 1**: Increased hazard (worse survival)
- **HR < 1**: Decreased hazard (better survival)
- Example: HR = 2.0 means twice the hazard of the event at any time

Event Counts:

The "Events" column is particularly important in survival analysis:

- Indicates the number of actual events (not censored observations) in each group
- Essential for assessing statistical power
- Categories with very few events may have unreliable HR estimates
- The footer shows events analyzed, total events, and percentage in the original data

Concordance (C-index):

The concordance statistic displayed in the footer indicates discrimination:

- Range: 0.5 to 1.0
- 0.5 = random prediction (coin flip)
- 0.7-0.8 = acceptable discrimination
- > 0.8 = excellent discrimination
- Standard error provided for confidence interval calculation

Global Log-Rank Test:

The global *p*-value tests the null hypothesis that all coefficients are zero:

- Significant *p*-value (< 0.05) indicates the model as a whole predicts survival
- Non-significant global test doesn't preclude significant individual predictors
- Based on the score (log-rank) test

Stratification and Clustering:

If the model includes stratification (`strata()`) or clustering (`cluster()`):

- Stratified variables are not shown in the forest plot (they don't have HRs)
- Clustering affects standard errors but not point estimates
- Both are handled automatically by the function

Aliased Coefficients:

Coefficients that the model could not estimate because their columns of the model matrix are linear combinations of others are omitted, as in the tables of `fit()`, together with any factor none of whose levels was estimated.

Proportional Hazards Assumption:

The forest plot assumes proportional hazards (constant HR over time). Users should verify this assumption using:

- `cox.zph(model)` for testing
- Stratification for variables violating the assumption
- Time-dependent coefficients if needed

Value

A `ggplot` object containing the complete forest plot. The plot can be:

- Displayed directly: `print(plot)`
- Saved to file: `forestsave(plot, "forest.pdf")`
- Further customized with **ggplot2** functions

The returned object includes an attribute `"rec_dims"` accessible via `attr(plot, "rec_dims")`, which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if `plot_width` or `plot_height` are not specified. The list also carries a `units` element recording the units the dimensions are expressed in, matching the `units` argument. `forestsave()` reads all three and requires no further handling.

The returned object also includes an attribute `"table_data"` accessible via `attr(plot, "table_data")`, a data.table holding the values drawn in the plot: one row per term in model order, with the variable, factor level, sample size, event count, estimate, confidence bounds and p-value. Sample sizes and event counts describe the observations used in fitting rather than every row supplied, so they sum to the model sample size within each variable.

See Also

`autoforest` for automatic model detection, `glmforest` for logistic/GLM forest plots, `lmforest` for linear model forest plots, `uniforest` for univariable screening forest plots, `multiforest` for multi-outcome forest plots, `coxph` for fitting Cox models, `fit` for regression modeling, `forestsave` for saving with recommended dimensions

Other visualization functions: `autoforest()`, `forestsave()`, `glmforest()`, `lmforest()`, `multiforest()`, `recdims()`, `uniforest()`

Examples

```
data(clintrial)
data(clintrial_labels)
library(survival)

# Create example model
model1 <- coxph(
    survival::Surv(os_months, os_status) ~ age + sex + treatment,
    data = clintrial)

# Example 1: Basic Cox model forest plot
p <- coxforest(model1, data = clintrial)



old_width <- options(width = 180)

# Example 2: With custom labels and title
plot2 <- coxforest(
    x = model1,
    data = clintrial,
    title = "Prognostic Factors for Overall Survival",
    labels = clintrial_labels
)

# Example 3: Comprehensive model with indented layout
model3 <- coxph(
    Surv(os_months, os_status) ~ age + sex + bmi + smoking +
        treatment + stage + grade,
    data = clintrial
)

plot3 <- coxforest(
    x = model3,
    data = clintrial,
    labels = clintrial_labels,
    indent_groups = TRUE,
    zebra_stripes = TRUE
)

# Example 4: Condensed layout for many binary predictors
model4 <- coxph(
    Surv(os_months, os_status) ~ age + sex + smoking +
        hypertension + diabetes + surgery,
    data = clintrial
)

plot4 <- coxforest(
    x = model4,
    data = clintrial,
    condense_table = TRUE,
    labels = clintrial_labels
```

```
)

# Example 5: Stratified Cox model
model5 <- coxph(
    Surv(os_months, os_status) ~ age + sex + treatment + strata(site),
    data = clintrial
)

plot5 <- coxforest(
    x = model5,
    data = clintrial,
    title = "Stratified by Study Site",
    labels = clintrial_labels
)

# Example 6: Save with recommended dimensions
forestsave(plot5, file.path(tempdir(), "survival_forest.pdf"))

options(old_width)
```

`desctable` *Create Publication-Ready Descriptive Statistics Tables*

Description

Generates comprehensive descriptive statistics tables with automatic variable type detection, group comparisons, and appropriate statistical testing. This function is designed to create "Table 1"-style summaries commonly used in clinical and epidemiological research, with full support for continuous, categorical, and time-to-event variables.

Usage

```
desctable(
  data,
  by = NULL,
  variables,
  stats_continuous = c("median_iqr"),
  stats_categorical = "n_percent",
  digits = 1,
  p_digits = 3,
  conf_level = 0.95,
  p_per_stat = FALSE,
  na_include = FALSE,
  na_label = "Unknown",
  na_percent = FALSE,
  test = TRUE,
  test_continuous = "auto",
```

```
  test_categorical = "auto",
  total = TRUE,
  total_label = "Total",
  labels = NULL,
  number_format = NULL,
  ...
)
```

Arguments

data	Data frame or data.table containing the dataset to summarize. Automatically converted to a data.table for efficient processing.
by	Character string specifying the column name of the grouping variable for stratified analysis (*e.g.*, treatment arm, exposure status). When NULL (default), produces overall summaries only without group comparisons or statistical tests.
variables	Character vector of variable names to summarize. Can include standard column names for continuous or categorical variables, and survival expressions using Surv() syntax (*e.g.*, "Surv(os_months, os_status)"). Variables are processed in the order provided.
stats_continuous	Character vector specifying which statistics to compute for continuous variables. Multiple values create separate rows for each variable. Options: • "mean_sd" - Mean ± standard deviation • "median_iqr" - Median [interquartile range] • "median_range" - Median (minimum-maximum) • "range" - Minimum-maximum only Default is "median_iqr".
stats_categorical	Character string specifying the format for categorical variable summaries: • "n" - Count only • "percent" - Percentage only • "n_percent" - Count (percentage) [default]
digits	Integer specifying the number of decimal places for continuous statistics. Default is 1.
p_digits	Integer specifying the number of decimal places for *p*-values. Values smaller than 10^(-p_digits) are displayed as "< 0.001" (for p_digits = 3), "< 0.0001" (for p_digits = 4), etc. Default is 3.
conf_level	Numeric confidence level for confidence intervals in survival variable summaries (median survival time with CI). Must be between 0 and 1. Default is 0.95 (95% confidence intervals).
p_per_stat	Logical. If TRUE, displays *p*-values on each row (per statistic) rather than only on the first row of each variable. Useful when different statistics within a variable warrant separate significance testing. Default is FALSE.
na_include	Logical. If TRUE, missing values (NAs) are displayed as a separate category/row for each variable. If FALSE, missing values are silently excluded from calculations. Default is FALSE.

`na_label` Character string used to label the missing values row when `na_include = TRUE`. Default is `"Unknown"`.

`na_percent` Logical. Controls how percentages are calculated for categorical variables when `na_include = TRUE`:

- If `TRUE`, percentages include NAs in the denominator (all categories sum to 100%)
- If `FALSE`, percentages exclude NAs from the denominator (non-missing categories sum to 100%, missing shown separately)

Only affects categorical variables. Default is `FALSE`.

`test` Logical. If `TRUE`, performs appropriate statistical tests for group comparisons and adds a *p*-value column. Requires by to be specified. Tests are automatically selected based on variable type and test parameters. Default is `TRUE`.

`test_continuous`

Character string specifying the statistical test for continuous variables:

- `"auto"` - Automatic selection: *t*-test/ANOVA for means, Wilcoxon/Kruskal-Wallis for medians [default]
- `"t"` - Independent samples *t*-test (2 groups only)
- `"aov"` - One-way ANOVA (2+ groups)
- `"wrs"` - Wilcoxon rank-sum test (2 groups only)
- `"kwt"` - Kruskal-Wallis test (2+ groups)

`test_categorical`

Character string specifying the statistical test for categorical variables:

- `"auto"` - Automatic selection: Fisher exact test if any expected cell frequency < 5, otherwise χ^2 test [default]
- `"fisher"` - Fisher exact test
- `"chisq"` - χ^2 test

`total` Logical or character string controlling the total column:

- `TRUE` or `"first"` - Include total column as first column after Variable/Group [default]
- `"last"` - Include total column as last column before *p*-value
- `FALSE` - Exclude total column

`total_label` Character string for the total column header. Default is `"Total"`.

`labels` Named character vector or list providing custom display labels for variables. Names should match variable names (or `Surv()` expressions), values are the display labels. Variables not in `labels` use their original names. Can also label the grouping variable specified in `by`. Default is `NULL`.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

`...` Additional arguments passed to the underlying statistical test functions (*e.g.*, `var.equal = TRUE` for *t*-tests, `simulate.p.value = TRUE` for Fisher test, `rho = 1` for the Fleming-Harrington G^1 test of survival curves). Each argument is passed only to the tests that accept it, so arguments for different tests may be combined in one call. Arguments that no test accepts are ignored with a warning.

Details

Variable Type Detection:

The function automatically detects variable types and applies appropriate summaries:

- **Continuous**: Numeric variables (integer or double) receive statistics specified in `stats_continuous`
- **Categorical**: Character, factor, or logical variables receive frequency counts and percentages
- **Time-to-Event**: Variables specified as `Surv(time, event)` display median survival with confidence intervals (level controlled by `conf_level`)

Statistical Testing:

When `test = TRUE` and `by` is specified:

- **Continuous with "auto"**: Parametric tests (*t*-test, ANOVA) for mean-based statistics; non-parametric tests (Wilcoxon, Kruskal-Wallis) for median-based statistics
- **Categorical with "auto"**: Fisher exact test when any expected cell frequency < 5; χ^2 test otherwise. Levels and groups without observations, and observations with a missing group, are excluded from the test. When `stats::fisher.test()` cannot complete the exact computation within its workspace, as is common for tables larger than 2×2 with a few hundred observations or more, the *p*-value is a Monte Carlo estimate from 10,000 simulated tables and a warning is issued
- **Survival**: Log-rank test for comparing survival curves, or another test of the G^ρ family of `survival::survdiff()` when `rho` is supplied
- **Range statistics**: No *p*-value computed (ranges are descriptive)

Missing Data Handling:

Missing values are handled differently by variable type:

- **Continuous**: NAs excluded from calculations; optionally shown as count when `na_include = TRUE`
- **Categorical**: NAs can be included as a category when `na_include = TRUE`. The `na_percent` parameter controls whether percentages are calculated with or without NAs in the denominator
- **Survival**: NAs in time or event excluded from analysis

Order of Rows and Columns:

Rows and columns follow the order of the data rather than a sorted order:

- Variables appear in the order of `variables`, and the statistics of each continuous variable in the order of `stats_continuous`
- The categories of a factor appear in the order of its levels, including levels without observations; the categories of a character or logical variable appear in order of first appearance in the data. The category of missing values, when shown, appears last
- The group columns follow the same rule for the grouping variable named in `by`

Regression models code a character variable as a factor with sorted levels, so the tables of `fit()` list its categories in sorted order. Converting the variable to a factor with its levels in the intended order gives the same order in both tables.

Formatting Conventions:

All numeric output respects the `number_format` parameter. Separators within ranges and confidence intervals adapt automatically to avoid ambiguity:

- Mean ± SD: `"45.2 ± 12.3"` (US) or `"45,2 ± 12,3"` (EU)
- Median [IQR]: `"38.0 [28.0–52.0]"` (US) or `"38,0 [28,0–52,0]"` (EU, en-dash separator)
- Range: `"18.0–75.0"` (positive, US), `"-5.0 to 10.0"` (when bounds are negative)
- Survival: `"24.5 (21.2–28.9)"` (US) or `"24,5 (21,2–28,9)"` (EU)
- Counts $\geq$ 1000: `"1,234"` (US) or `"1.234"` (EU)
- *p*-values: `"< 0.001"` (US) or `"< 0,001"` (EU)

Value

A data.table with S3 class `"desctable"` containing formatted descriptive statistics. The table structure includes:

Variable Variable name or label (from `labels`)

Group For continuous variables: statistic type (*e.g.*, "Mean ± SD", "Median [IQR]"). For categorical variables: category level. Empty for variable name rows.

Total Statistics for the total sample (if `total = TRUE`)

Group columns Statistics for each group level (when `by` is specified). Column names match group levels.

***p*-value** Formatted *p*-values from statistical tests (when `test = TRUE` and `by` is specified)

The first row always shows sample sizes for each column. All numeric output (counts, statistics, *p*-values) respects the `number_format` setting for locale-appropriate formatting.

The returned object includes the following attributes accessible via `attr()`:

raw_data A data.table containing unformatted numeric values suitable for further statistical analysis or custom formatting. Includes additional columns for standard deviations, quartiles, etc.

by_variable The grouping variable name used (value of `by`)

variables The variables analyzed (value of `variables`)

See Also

survtable for detailed survival summary tables, fit for regression modeling, tablesave for export in any supported format, table2pdf for PDF export, table2docx for Word export, table2html for HTML export

Other descriptive functions: print.survtable(), survtable()

Examples

```
# Load example clinical trial data
data(clintrial)

# Example 1: Basic descriptive table without grouping
desctable(clintrial,
        variables = c("age", "sex", "bmi"),
        labels = clintrial_labels)



# Example 2: Grouped comparison with default tests
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex", "race", "bmi"),
        labels = clintrial_labels)

# Example 3: Customize continuous statistics
desctable(clintrial,
        by = "treatment",
        variables = c("age", "bmi", "creatinine"),
        stats_continuous = c("median_iqr", "range"),
        labels = clintrial_labels)

# Example 4: Change categorical display format
desctable(clintrial,
        by = "treatment",
        variables = c("sex", "race", "smoking"),
        stats_categorical = "n",  # Show counts only
        labels = clintrial_labels)

# Example 5: Include missing values
desctable(clintrial,
        by = "treatment",
        variables = c("age", "smoking", "hypertension"),
        na_include = TRUE,
        na_label = "Missing",
        labels = clintrial_labels)

# Example 6: Disable statistical testing
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex", "bmi"),
        test = FALSE,
```

```
        labels = clintrial_labels)

# Example 7: Force specific tests
desctable(clintrial,
        by = "surgery",
        variables = c("age", "sex"),
        test_continuous = "t",        # t-test instead of auto
        test_categorical = "fisher",  # Fisher test instead of auto
        labels = clintrial_labels)

# Example 8: Adjust decimal places
desctable(clintrial,
        by = "treatment",
        variables = c("age", "bmi"),
        digits = 2,    # 2 decimals for continuous
        p_digits = 4,  # 4 decimals for p-values
        labels = clintrial_labels)

# Example 9: Custom variable labels
labels <- c(
    age = "Age (years)",
    sex = "Sex",
    bmi = "Body Mass Index (kg/m\u00b2)",
    treatment = "Treatment Arm"
)

desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex", "bmi"),
        labels = labels)

# Example 10: Position total column last
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex"),
        total = "last",
        labels = clintrial_labels)

# Example 11: Exclude total column
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex"),
        total = FALSE,
        labels = clintrial_labels)

# Example 12: Survival analysis
desctable(clintrial,
        by = "treatment",
        variables = "Surv(os_months, os_status)",
        labels = clintrial_labels)

# Example 13: Multiple survival endpoints
desctable(clintrial,
```

```
        by = "treatment",
        variables = c(
            "Surv(pfs_months, pfs_status)",
            "Surv(os_months, os_status)"
        ),
        labels = c(
            "Surv(pfs_months, pfs_status)" = "Progression-Free Survival",
            "Surv(os_months, os_status)" = "Overall Survival"
        ))

# Example 14: Mixed variable types
desctable(clintrial,
        by = "treatment",
        variables = c(
            "age", "sex", "race",           # Demographics
            "bmi", "creatinine",            # Labs
            "smoking", "hypertension",      # Risk factors
            "Surv(os_months, os_status)"    # Survival
        ),
        labels = clintrial_labels)

# Example 15: Three or more groups
desctable(clintrial,
        by = "stage",  # Assuming stage has 3+ levels
        variables = c("age", "sex", "bmi"),
        labels = clintrial_labels)
# Automatically uses ANOVA/Kruskal-Wallis and chi-squared

# Example 16: Access raw unformatted data
result <- desctable(clintrial,
                    by = "treatment",
                    variables = c("age", "bmi"))
raw_data <- attr(result, "raw_data")
print(raw_data)
# Raw data includes unformatted numbers, SDs, quartiles, etc.

# Example 17: Check which grouping variable was used
result <- desctable(clintrial,
                    by = "treatment",
                    variables = c("age", "sex"))
attr(result, "by_variable")  # "treatment"

# Example 18: NA percentage calculation options
# Include NAs in percentage denominator (all sum to 100%)
desctable(clintrial,
        by = "treatment",
        variables = "smoking",
        na_include = TRUE,
        na_percent = TRUE,
        labels = clintrial_labels)

# Exclude NAs from denominator (non-missing sum to 100%)
desctable(clintrial,
```

```
        by = "treatment",
        variables = "smoking",
        na_include = TRUE,
        na_percent = FALSE,
        labels = clintrial_labels)

# Example 19: Passing additional test arguments
# Equal variance t-test
desctable(clintrial,
        by = "sex",
        variables = "age",
        test_continuous = "t",
        var.equal = TRUE,
        labels = clintrial_labels)

# Example 20: European number formatting
desctable(clintrial,
        by = "treatment",
        variables = c("age", "sex", "bmi"),
        number_format = "eu",
        labels = clintrial_labels)

# Example 21: Complete Table 1 for publication
table1 <- desctable(
    data = clintrial,
    by = "treatment",
    variables = c(
        "age", "sex", "race", "ethnicity", "bmi",
        "smoking", "hypertension", "diabetes",
        "ecog", "creatinine", "hemoglobin",
        "site", "stage", "grade",
        "Surv(os_months, os_status)"
    ),
    labels = clintrial_labels,
    stats_continuous = c("median_iqr", "range"),
    total = TRUE,
    na_include = FALSE
)
print(table1)
```

`fit` *Fit Regression Model with Publication-Ready Output*

Description

Provides a unified interface for fitting various types of regression models with automatic formatting of results for publication. Supports generalized linear models, linear models, survival models, and

mixed-effects models with consistent syntax and output formatting. Handles both univariable and multivariable models automatically.

Usage

```
fit(
  data = NULL,
  outcome = NULL,
  predictors = NULL,
  model = NULL,
  model_type = "glm",
  family = "binomial",
  random = NULL,
  interactions = NULL,
  strata = NULL,
  cluster = NULL,
  weights = NULL,
  conf_level = 0.95,
  reference_rows = TRUE,
  show_n = TRUE,
  show_events = TRUE,
  digits = 2,
  p_digits = 3,
  labels = NULL,
  keep_qc_stats = TRUE,
  exponentiate = NULL,
  conf_method = NULL,
  number_format = NULL,
  verbose = NULL,
  ...
)
```

Arguments

`data`	Data frame or data.table containing the analysis dataset. Required for formula-based workflow; optional for model-based workflow (extracted from model if not provided).
`outcome`	Character string specifying the outcome variable name. For survival analysis, use `Surv()` syntax from the **survival** package (*e.g.,* `"Surv(time, status)"` or `"Surv(os_months, os_status)"`). Required for formula-based workflow; ignored if `model` is provided.
`predictors`	Character vector of predictor variable names to include in the model. All predictors are included simultaneously (multivariable model). For univariable models, provide a single predictor. Can include continuous, categorical (factor), or binary variables. Required for formula-based workflow; ignored if `model` is provided. Non-syntactic names (*e.g.,* `"tumor stage"`) are quoted in the model formula and need no backticks.

`model` Optional pre-fitted model object to format. When provided, `outcome` and `predictors` are ignored and the model is formatted directly. Supported model classes include:

- `glm` - Generalized linear models
- `negbin` - Negative binomial models from `MASS::glm.nb()`
- `lm` - Linear models
- `coxph` - Cox proportional hazards models
- `lmerMod`, `glmerMod` - Mixed-effects models from **lme4**
- `coxme` - Mixed-effects Cox models

`model_type` Character string specifying the type of regression model. Ignored if `model` is provided. Options include:

- `"glm"` - Generalized linear model (default). Supports multiple distributions via the `family` parameter including logistic, Poisson, Gamma, Gaussian, and quasi-likelihood models.
- `"lm"` - Linear regression for continuous outcomes with normally distributed errors. Equivalent to `glm` with `family = "gaussian"`.
- `"coxph"` - Cox proportional hazards model for time-to-event survival analysis. Requires `Surv()` outcome syntax.
- `"clogit"` - Conditional logistic regression for matched case-control studies or stratified analyses.
- `"negbin"` - Negative binomial regression for overdispersed count data (requires **MASS** package). Estimates an additional dispersion parameter compared to Poisson regression.
- `"lmer"` - Linear mixed-effects model for hierarchical or clustered data with continuous outcomes (requires **lme4** package).
- `"glmer"` - Generalized linear mixed-effects model for hierarchical or clustered data with non-normal outcomes (requires **lme4** package).
- `"coxme"` - Cox mixed-effects model for clustered survival data (requires coxme package).

`family` For GLM and GLMER models, specifies the error distribution and link function. Can be a character string, a family function, or a family object. Ignored for non-GLM/GLMER models.

Binary/Binomial outcomes:

- `"binomial"` or `binomial()` - Logistic regression for binary outcomes (0/1, TRUE/FALSE). Returns odds ratios (OR). Default.
- `"quasibinomial"` or `quasibinomial()` - Logistic regression with overdispersion. Use when residual deviance » residual df.
- `binomial(link = "probit")` - Probit regression (normal CDF link).
- `binomial(link = "cloglog")` - Complementary log-log link for asymmetric binary outcomes.

Count outcomes:

- `"poisson"` or `poisson()` - Poisson regression for count data. Returns rate ratios (RR). Assumes mean = variance.

- "quasipoisson" or quasipoisson() - Poisson regression with overdispersion. Use when variance > mean.

Continuous outcomes:

- "gaussian" or gaussian() - Normal/Gaussian distribution for continuous outcomes. Equivalent to linear regression.
- gaussian(link = "log") - Log-linear model for positive continuous outcomes. Returns multiplicative effects.
- gaussian(link = "inverse") - Inverse link for specific applications.

Positive continuous outcomes:

- "Gamma" or Gamma() - Gamma distribution for positive, right-skewed continuous data (*e.g.,* costs, lengths of stay). When passed as a string, resolves to log link for interpretable multiplicative effects. The canonical inverse link can be specified explicitly with Gamma(link = "inverse").
- Gamma(link = "log") - Gamma with log link (equivalent to "Gamma" string shorthand).
- Gamma(link = "inverse") - Gamma with inverse (canonical) link.
- Gamma(link = "identity") - Gamma with identity link for additive effects on positive outcomes.
- "inverse.gaussian" or inverse.gaussian() - Inverse Gaussian for positive, highly right-skewed data.

For negative binomial regression (overdispersed counts), use model_type = "negbin" instead of the family parameter.

See family for additional details and options.

`random` Character string specifying the random-effects formula for mixed-effects models (glmer, lmer, coxme). Use standard **lme4/coxme** syntax, *e.g.,* "(1|site)" for random intercepts by site, "(1|site/patient)" for nested random effects, or "(1|site) + (1|doctor)" for crossed random effects. Required when model_type is a mixed-effects model type. Alternatively, random effects can be included directly in the predictors vector using the same syntax. Default is NULL.

`interactions` Character vector of interaction terms using colon notation (*e.g.,* c("age:sex", "treatment:stage")). Interaction terms are added to the model in addition to main effects. Default is NULL (no interactions).

`strata` For Cox or conditional logistic models, character string naming the stratification variable. Creates separate baseline hazards for each stratum level without estimating stratum effects. Default is NULL.

`cluster` For Cox models, character string naming the variable for robust clustered standard errors. Accounts for within-cluster correlation (*e.g.,* patients within hospitals). Default is NULL.

`weights` Character string naming the weights variable in data. The specified column should contain numeric values for observation weights. Used for weighted regression, survey data, or inverse probability weighting. Default is NULL.

`conf_level` Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals).

`reference_rows` Logical. If `TRUE`, adds rows for reference categories of factor variables with baseline values (OR/HR/RR = 1, coefficient = 0). Default is `TRUE`.

`show_n` Logical. If `TRUE`, includes the sample size column in the output. Default is `TRUE`.

`show_events` Logical. If `TRUE`, includes the events column in the output (for survival and logistic regression). Default is `TRUE`.

`digits` Integer specifying the number of decimal places for effect estimates (OR, HR, RR, coefficients). Default is 2.

`p_digits` Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

`labels` Named character vector or list providing custom display labels for variables. Names should match variable names, values are display labels. Default is `NULL`.

`keep_qc_stats` Logical. If `TRUE`, includes model quality statistics (AIC, BIC, R-squared, concordance, *etc.*) in the raw data attribute for model diagnostics and comparison. Default is `TRUE`.

`exponentiate` Logical. Whether to exponentiate coefficients. Default is `NULL`, which automatically exponentiates for logistic, Poisson, and Cox models, and displays raw coefficients for linear models. Set to `TRUE` to force exponentiation or `FALSE` to force coefficients.

`conf_method` Character string controlling the confidence interval method. If `NULL` (default), uses `getOption("summata.conf_method", "profile")`.

- `"profile"` - Profile likelihood intervals for GLM and negative binomial models (via `MASS::confint.glm()`), exact *t*-distribution intervals for linear models. Falls back to Wald on profiling failure. Quasi-likelihood families always use Wald (no true likelihood).
- `"wald"` - Wald intervals (coefficient $\pm z \times$ SE) for all model types. Faster but less accurate near boundary conditions or with small subgroups.

Cox and mixed-effects models use Wald intervals regardless of this setting.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

`verbose` Logical. If `TRUE`, displays model fitting warnings (*e.g.,* singular fit, convergence issues). If `FALSE` (default), routine fitting messages are suppressed while unexpected warnings are preserved. When `NULL`, uses `getOption("summata.verbose", FALSE)`.

... Additional arguments passed to the underlying model fitting function (glm, lm, coxph, glmer, *etc.*). Common options include subset, na.action, and model-specific control parameters.

Details

Model Scope Detection:

The function automatically detects whether the model is:

- **Univariable**: Single predictor (*e.g.,* predictors = "age"). Effect estimates are labeled as unadjusted ("OR", "HR", *etc.*), representing crude (unadjusted) association
- **Multivariable**: Multiple predictors (*e.g.,* predictors = c("age", "sex", "treatment")) Effect estimates are labeled as adjusted ("aOR", "aHR", *etc.*), representing associations adjusted for confounding

Interaction Terms:

Interactions are specified using colon notation and added to the model:

- interactions = c("age:treatment") creates interaction between age and treatment
- Main effects for both variables are automatically included
- Multiple interactions can be specified: c("age:sex", "treatment:stage")
- For interactions between categorical variables, separate terms are created for each combination of levels

Stratification (Cox/Conditional Logistic):

The strata parameter creates separate baseline hazards:

- Allows baseline hazard to vary across strata without estimating stratum effects
- Useful when proportional hazards assumption violated across strata
- Example: strata = "center" for multicenter studies
- Stratification variable is not included as a predictor

Clustering (Cox Models):

The cluster parameter computes robust standard errors:

- Accounts for within-cluster correlation (*e.g.,* multiple observations per patient)
- Uses sandwich variance estimator
- Does not change point estimates, only standard errors and p-values

Weighting:

The weights parameter enables weighted regression:

- For survey data with sampling weights
- Inverse probability weighting for causal inference
- Frequency weights for aggregated data
- Weights should be in a column of data

Mixed-Effects Models (lmer/glmer/coxme):

Mixed effects models handle hierarchical or clustered data:

- Use `model_type = "lmer"` for continuous/normal outcomes
- Use `model_type = "glmer"` with appropriate `family` for GLM outcomes
- Use `model_type = "coxme"` for survival outcomes with clustering
- Random effects are specified in predictors using lme4 syntax:
  - `"(1|site)"` - Random intercepts by site
  - `"(treatment|site)"` - Random slopes for treatment by site
  - `"(1 + treatment|site)"` - Both random intercepts and slopes
- Include random effects as part of the predictors vector
- Example: `predictors = c("age", "treatment", "(1|site)")`

Effect Measures by Model Type:

- **Logistic** (`family = "binomial"/"quasibinomial"`): Odds ratios (OR/aOR)
- **Cox** (`model_type = "coxph"`): Hazard ratios (HR/aHR)
- **Poisson/Count** (`family = "poisson"/"quasipoisson"`): Rate ratios (RR/aRR)
- **Negative binomial** (`model_type = "negbin"`): Rate ratios (RR/aRR)
- **Gamma/Log-link**: Ratios (multiplicative effects)
- **Linear/Gaussian**: Raw coefficient estimates (additive effects)

Confidence Intervals:

Confidence interval computation is tailored to each model class using the best available method:

- **GLM and negative binomial**: Profile likelihood intervals via `MASS::confint.glm()`, which invert the profile deviance and account for asymmetry in the likelihood surface. More accurate than the Wald approximation when subgroup sizes are small or estimates are near boundary values. Quasi-likelihood families (`quasibinomial`, `quasipoisson`) fall back to Wald intervals because they lack a true likelihood function.
- **Linear models**: Exact t-distribution intervals via `confint.lm()`, based on the known sampling distribution under normality.
- **Cox proportional hazards**: Wald intervals (*i.e.,* coefficient $\pm\, z \times$ SE), the standard approach in the survival analysis literature.
- **Mixed-effects models** (lmer, glmer, coxme): Wald intervals. Profile likelihood is available for **lme4** models via `confint(model, method = "profile")` but can be prohibitively slow for complex random-effects structures and is not used by default.

If profile likelihood computation fails for any reason (*e.g.,* non-convergence during profiling), the function falls back silently to Wald intervals.

Value

A data.table with S3 class `"fit_result"` containing formatted regression results. The table structure includes:

Variable Character. Predictor name or custom label

Group Character. For factor variables: category level. For interactions: interaction term. For continuous: typically empty

n Integer. Sample size used in fitting, that is complete cases across the model variables (if `show_n = TRUE`)

n_group Integer. Sample size for this factor level

events Integer. Total number of events (if `show_events = TRUE`)

events_group Integer. Events for this factor level

OR/HR/RR/Coefficient or aOR/aHR/aRR/Adj. Coefficient (95% CI) Character. Formatted effect estimate with confidence interval. Column name depends on model type and scope. Univariable models use: OR, HR, RR, Coefficient. Multivariable models use adjusted notation: aOR, aHR, aRR, Adj. Coefficient

***p*-value** Character. Formatted *p*-value from Wald test

The returned object includes the following attributes accessible via `attr()`:

model The fitted model object (glm, lm, coxph, *etc.*). Access for diagnostics, predictions, or further analysis

raw_data data.table. Unformatted numeric results with columns for coefficients, standard errors, confidence bounds, quality statistics, *etc.*

analysis_counts List. How much of the supplied data reached the model, with elements `n_supplied`, `n_analyzed`, `n_missing_outcome` and `n_missing_predictor`, plus `events_supplied` and `events_analyzed` where the model type carries an event count. The counts describe the single fitted model, and the `print()` method always reports the analyzed sample

outcome Character. The outcome variable name

predictors Character vector. The predictor variable names

formula_str Character. The complete model formula as a string

model_scope Character. "Univariable" (one predictor) or "Multivariable" (multiple predictors)

model_type Character. The regression model type used

interactions Character vector (if interactions specified). The interaction terms included

strata Character (if stratification used). The stratification variable

cluster Character (if clustering used). The cluster variable

weights Character (if weighting used). The weights variable

significant Character vector. Names of predictors with *p*-value below 0.05, suitable for downstream variable selection workflows

See Also

uniscreen for univariable screening of multiple predictors, fullfit for complete univariable-to-multivariable workflow, compfit for comparing multiple models, m2dt for model-to-table conversion, autoforest for forest plots of fitted models, forestsave for exporting forest plots, tablesave for exporting tables

Other regression functions: compfit(), fullfit(), multifit(), print.compfit_result(), print.fit_result(), print.fullfit_result(), print.multifit_result(), print.uniscreen_result(), uniscreen()

Examples

```
# Load example data
data(clintrial)
data(clintrial_labels)
library(survival)

# Example 1: Univariable logistic regression
uni_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = "age",
    labels = clintrial_labels
)
print(uni_model)



# Example 2: Multivariable logistic regression
multi_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "treatment"),
    labels = clintrial_labels
)
print(multi_model)

# Example 3: Cox proportional hazards model
cox_model <- fit(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "treatment", "stage"),
    model_type = "coxph",
    labels = clintrial_labels
)
print(cox_model)

# Example 4: Model with interaction terms
interact_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "treatment", "sex"),
    interactions = c("age:treatment"),
```

```
    labels = clintrial_labels
)
print(interact_model)

# Example 5: Cox model with stratification
strat_model <- fit(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "treatment"),
    model_type = "coxph",
    strata = "site",  # Separate baseline hazards by site
    labels = clintrial_labels
)
print(strat_model)

# Example 6: Cox model with clustering
cluster_model <- fit(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "treatment"),
    model_type = "coxph",
    cluster = "site",  # Robust SEs accounting for site clustering
    labels = clintrial_labels
)
print(cluster_model)

# Example 7: Linear regression
linear_model <- fit(
    data = clintrial,
    outcome = "bmi",
    predictors = c("age", "sex", "smoking"),
    model_type = "lm",
    labels = clintrial_labels
)
print(linear_model)

# Example 8: Poisson regression for equidispersed count data
# fu_count has variance ~= mean, appropriate for standard Poisson
poisson_model <- fit(
    data = clintrial,
    outcome = "fu_count",
    predictors = c("age", "stage", "treatment", "surgery"),
    model_type = "glm",
    family = "poisson",
    labels = clintrial_labels
)
print(poisson_model)
# Returns rate ratios (RR/aRR)

# Example 9: Negative binomial regression for overdispersed counts
# ae_count has variance > mean (overdispersed), use negbin or quasipoisson
if (requireNamespace("MASS", quietly = TRUE)) {
    nb_result <- fit(
```

```
        data = clintrial,
        outcome = "ae_count",
        predictors = c("age", "treatment", "diabetes", "surgery"),
        model_type = "negbin",
        labels = clintrial_labels
    )
    print(nb_result)
}

# Example 10: Gamma regression for positive continuous outcomes
gamma_model <- fit(
    data = clintrial,
    outcome = "los_days",
    predictors = c("age", "treatment", "surgery"),
    model_type = "glm",
    family = Gamma(link = "log"),
    labels = clintrial_labels
)
print(gamma_model)

# Example 11: Access the underlying fitted model
result <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi")
)

# Get the model object
model_obj <- attr(result, "model")
summary(model_obj)

# Model diagnostics
plot(model_obj)

# Predictions
preds <- predict(model_obj, type = "response")

# Example 12: Access raw numeric data
raw_data <- attr(result, "raw_data")
print(raw_data)
# Contains unformatted coefficients, SEs, CIs, AIC, BIC, etc.

# Example 13: Multiple interactions
complex_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment", "bmi"),
    interactions = c("age:treatment", "sex:bmi"),
    labels = clintrial_labels
)
print(complex_model)

# Example 14: Customize output columns
```

```
minimal <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment"),
    show_n = FALSE,
    show_events = FALSE,
    reference_rows = FALSE,
    labels = clintrial_labels
)
print(minimal)

# Example 15: Different confidence levels
ci90 <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "treatment"),
    conf_level = 0.90,  # 90% confidence intervals
    labels = clintrial_labels
)
print(ci90)

# Example 16: Force coefficient display instead of OR
coef_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "bmi"),
    exponentiate = FALSE,  # Show log odds instead of OR
    labels = clintrial_labels
)
print(coef_model)

# Example 17: Confidence interval method
# Default: profile likelihood CIs for GLM (more accurate)
profile_result <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "treatment"),
    p_digits = 4,
    conf_method = "profile",
    labels = clintrial_labels
)
print(profile_result)

# Wald CIs (faster, suitable for simulation or exploratory work)
wald_result <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "treatment"),
    p_digits = 4,
    conf_method = "wald",
    labels = clintrial_labels
)
print(wald_result)
```

```
# Example 18: Check model quality statistics
result <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment", "stage"),
    keep_qc_stats = TRUE
)

raw <- attr(result, "raw_data")
cat("AIC:", raw$AIC[1], "\n")
cat("BIC:", raw$BIC[1], "\n")
cat("C-statistic:", raw$c_statistic[1], "\n")

# Example 19: Interaction effects - treatment effect modified by stage
interaction_model <- fit(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "treatment", "stage"),
    interactions = c("treatment:stage"),
    model_type = "coxph",
    labels = clintrial_labels
)
print(interaction_model)
# Shows main effects plus all treatment×stage interaction terms

# Example 20: Multiple interactions in logistic regression
multi_interaction <- fit(
    data = clintrial,
    outcome = "readmission_30d",
    predictors = c("age", "sex", "surgery", "diabetes"),
    interactions = c("surgery:diabetes", "age:sex"),
    labels = clintrial_labels
)
print(multi_interaction)

# Example 21: Quasipoisson for overdispersed count data
# Alternative to negative binomial when MASS not available
quasi_model <- fit(
    data = clintrial,
    outcome = "ae_count",
    predictors = c("age", "treatment", "diabetes", "surgery"),
    model_type = "glm",
    family = "quasipoisson",
    labels = clintrial_labels
)
print(quasi_model)
# Adjusts standard errors for overdispersion

# Example 22: Quasibinomial for overdispersed binary data
quasi_logistic <- fit(
    data = clintrial,
    outcome = "any_complication",
```

```
    predictors = c("age", "bmi", "diabetes", "surgery"),
    model_type = "glm",
    family = "quasibinomial",
    labels = clintrial_labels
)
print(quasi_logistic)

# Example 23: Gamma regression with identity link for additive effects
gamma_identity <- fit(
    data = clintrial,
    outcome = "los_days",
    predictors = c("age", "treatment", "surgery", "any_complication"),
    model_type = "glm",
    family = Gamma(link = "identity"),
    labels = clintrial_labels
)
print(gamma_identity)
# Shows additive effects (coefficients) instead of multiplicative (ratios)

# Example 24: Inverse Gaussian regression for highly skewed data
inverse_gaussian <- fit(
    data = clintrial,
    outcome = "recovery_days",
    predictors = c("age", "surgery", "pain_score"),
    model_type = "glm",
    family = inverse.gaussian(link = "log"),
    labels = clintrial_labels
)
print(inverse_gaussian)

# Example 25: Linear mixed effects with random intercepts
# Accounts for clustering of patients within sites
if (requireNamespace("lme4", quietly = TRUE)) {
    lmer_model <- fit(
        data = clintrial,
        outcome = "los_days",
        predictors = c("age", "treatment", "stage", "(1|site)"),
        model_type = "lmer",
        labels = clintrial_labels
    )
    print(lmer_model)
}

# Example 26: Generalized linear mixed effects (logistic with random effects)
if (requireNamespace("lme4", quietly = TRUE)) {
    glmer_model <- fit(
        data = clintrial,
        outcome = "readmission_30d",
        predictors = c("age", "surgery", "los_days", "(1|site)"),
        model_type = "glmer",
        family = "binomial",
        labels = clintrial_labels
    )
```

```
    print(glmer_model)
}

# Example 27: Cox mixed effects for clustered survival data
if (requireNamespace("coxme", quietly = TRUE)) {
    coxme_model <- fit(
        data = clintrial,
        outcome = "Surv(os_months, os_status)",
        predictors = c("age", "treatment", "stage", "(1|site)"),
        model_type = "coxme",
        labels = clintrial_labels
    )
    print(coxme_model)
}

# Example 28: Random slopes - treatment effect varies by site
if (requireNamespace("lme4", quietly = TRUE)) {
    random_slopes <- fit(
        data = clintrial,
        outcome = "los_days",
        predictors = c("age", "treatment", "stage", "(treatment|site)"),
        model_type = "lmer",
        labels = clintrial_labels
    )
    print(random_slopes)
}

# Example 29: Format a pre-fitted model (model-based workflow)
# Useful for models fitted outside of fit()
pre_fitted <- glm(os_status ~ age + sex + treatment,
                  family = binomial, data = clintrial)
result <- fit(model = pre_fitted,
              data = clintrial,
              labels = clintrial_labels)
print(result)
```

`forestsave` *Save a Forest Plot to File*

Description

Writes a forest plot to file, sized by default from the recommendations the plotting function computed from the plot's own content, and rendered through a graphics device chosen to reproduce the fonts seen on screen.

Usage

```
forestsave(
  plot,
  file,
  width = NULL,
  height = NULL,
  units = NULL,
  dpi = 300,
  device = NULL,
  embed_fonts = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

plot	A forest plot produced by `lmforest()`, `glmforest()`, `coxforest()`, `uniforest()`, `multiforest()`, or `autoforest()`. Any **ggplot2** object is accepted, but one that carries no recommended dimensions requires `width` and `height` to be supplied.
file	Character string giving the output path. The file extension determines the format: `.pdf`, `.png`, `.tiff`, `.jpeg`, `.svg`, `.eps`, and the other formats `ggplot2::ggsave()` supports.
width, height	Numeric plot dimensions. Taken from the plot's recommended dimensions when not supplied, and converted where these differ from `units`.
units	Character string giving the units for `width` and `height`: `"in"` (inches), `"cm"`, or `"mm"`. Defaults to the units the recommended dimensions were expressed in, and to `"in"` where none were recorded.
dpi	Numeric resolution in dots per inch for raster formats. Ignored for vector formats. Default is `300`.
device	Graphics device. Selected automatically when `NULL`: see Details. Supply `"cairo"` to use the Cairo device for the output format, which embeds fonts, or any device function or name recognized by `ggplot2::ggsave()`.
embed_fonts	Logical. Embed the fonts in a PDF after it is written, using `grDevices::embedFonts()`. Requires Ghostscript, and applies only to PDF output. Default is `FALSE`.
quiet	Logical. Suppress the message reporting the dimensions used. Default is `FALSE`.
...	Additional arguments passed to `ggplot2::ggsave()`, such as `bg` or `limitsize`.

Details

Forest plots are sized by their content rather than by the page: a plot with more variables, longer labels, or more factor levels requires a larger canvas, and the plotting functions compute one and attach it to the returned object as a `"rec_dims"` attribute. `forestsave()` reads that attribute, so the following are equivalent:

```
dims <- attr(p, "rec_dims")
ggplot2::ggsave("forest.pdf", p, width = dims$width,
```

```
                height = dims$height, units = dims$units)

forestsave(p, "forest.pdf")
```

Device selection. Fonts are measured differently by different graphics devices, so a figure written through one device may not reproduce the spacing seen on screen or in the package documentation. Unless `device` is supplied, `forestsave()` selects:

PDF R's internal PDF device, `grDevices::pdf()`, reached by deferring to `ggplot2::ggsave()`. Cairo is not used by default: although it embeds fonts, its face selection can substitute an incorrect italic, which matters here because forest plot footers and statistical notation are set in italic. Pass `device = "cairo"` to use it where font embedding is required.

PNG, TIFF, JPEG The corresponding **ragg** device where that package is installed, whose text metrics match those used to render the package vignettes. Falls back to the standard raster devices otherwise.

Other formats Left to `ggplot2::ggsave()`.

Font embedding. R's PDF device draws on the standard fonts that every PDF reader is required to provide, so its output is portable without embedding. Embedding matters where the plot uses a font outside that set, and where a publisher requires it. Setting `embed_fonts = TRUE` embeds the fonts after the file is written, through `grDevices::embedFonts()`, which requires a Ghostscript installation. This is offered as an alternative to `device = "cairo"`, which embeds fonts as it draws but selects its own italic face.

`device = "cairo"` is a shorthand of this package's own, resolving to `grDevices::cairo_pdf()`, `grDevices::cairo_ps()`, or `grDevices::svg()` according to the output format:

```
forestsave(p, "forest.pdf", device = "cairo")
```

It applies to vector formats only, since raster output already uses the **ragg** devices, which supersede Cairo's raster backends. Cairo is absent from some installations even where `capabilities("cairo")` reports otherwise, so the device is opened once before use and its unavailability reported. Any other value of `device` is passed to `ggplot2::ggsave()` unchanged.

The dimensions used are reported through a `message()` unless `quiet = TRUE`, so that a figure written at an unexpected size is apparent at the point it is written.

Value

Invisibly returns `file`.

See Also

`autoforest`, `coxforest`, `glmforest`, `lmforest`, `uniforest`, `multiforest` for producing forest plots; `tablesave` for exporting formatted tables

Other visualization functions: `autoforest()`, `coxforest()`, `glmforest()`, `lmforest()`, `multiforest()`, `recdims()`, `uniforest()`

Examples

```
data(clintrial)
data(clintrial_labels)

model <- stats::glm(readmission_30d ~ age + sex + stage,
                    data = clintrial, family = stats::binomial)

p <- glmforest(model, data = clintrial, labels = clintrial_labels)

# Example 1: Save at the recommended dimensions
forestsave(p, file.path(tempdir(), "forest.pdf"))



# Example 2: Raster output at publication resolution
forestsave(p, file.path(tempdir(), "forest.png"), dpi = 600)

# Example 3: Dimensions given explicitly, in any supported units
forestsave(p, file.path(tempdir(), "forest_sized.pdf"),
           width = 24, height = 16, units = "cm")

# Example 4: Embed fonts after writing, where Ghostscript is available
forestsave(p, file.path(tempdir(), "forest_embedded.pdf"),
           embed_fonts = TRUE)
```

---

fullfit *Complete Regression Analysis Workflow*

Description

Executes a comprehensive regression analysis pipeline that combines univariable screening, automatic/manual variable selection, and multivariable modeling in a single function call. This function is designed to streamline the complete analytical workflow from initial exploration to final adjusted models, with publication-ready formatted output showing both univariable and multivariable results side-by-side if desired.

Usage

```
fullfit(
  data,
  outcome,
  predictors,
  method = "screen",
  multi_predictors = NULL,
  p_threshold = 0.05,
  columns = "both",
```

```
  model_type = "glm",
  family = "binomial",
  random = NULL,
  conf_level = 0.95,
  reference_rows = TRUE,
  show_n = TRUE,
  show_events = TRUE,
  digits = 2,
  p_digits = 3,
  labels = NULL,
  metrics = "both",
  return_type = "table",
  keep_models = FALSE,
  exponentiate = NULL,
  conf_method = NULL,
  parallel = TRUE,
  n_cores = NULL,
  number_format = NULL,
  verbose = NULL,
  ...
)
```

Arguments

`data`	Data frame or data.table containing the analysis dataset. The function automatically converts data frames to data.tables for efficient processing.
`outcome`	Character string specifying the outcome variable name. For time-to-event analysis, use `Surv()` syntax for the outcome variable (*e.g.,* `"Surv(os_months, os_status)"`).
`predictors`	Character vector of predictor variable names to analyze. All predictors are tested in univariable models. The subset included in the multivariable model depends on the `method` parameter. Non-syntactic names (*e.g.,* `"tumor stage"`) are quoted in the model formula and need no backticks.
`method`	Character string specifying the variable selection strategy: • `"screen"` - Automatic selection based on univariable *p*-value threshold. Only predictors with p $\leq$ `p_threshold` in univariable analysis are included in the multivariable model [default] • `"all"` - Include all predictors in both univariable and multivariable analyses (no selection) • `"custom"` - Manual selection. All predictors in univariable analysis, but only those specified in `multi_predictors` are included in multivariable model
`multi_predictors`	Character vector of predictors to include in the multivariable model when `method = "custom"`. Required when using custom selection. Ignored for other methods. Default is `NULL`.

`p_threshold` Numeric *p*-value threshold for automatic variable selection when `method = "screen"`. Predictors with univariable *p*-value less than or equal to the threshold are included in multivariable model. Common values: 0.05 (strict), 0.10 (moderate), 0.20 (liberal screening). Default is 0.05. Ignored for other methods.

`columns` Character string specifying which result columns to display:

- `"both"` - Show both univariable and multivariable results side-by-side [default]
- `"uni"` - Show only univariable results
- `"multi"` - Show only multivariable results

`model_type` Character string specifying the regression model type:

- `"glm"` - Generalized linear model (default). Supports multiple distributions via the `family` parameter including logistic, Poisson, Gamma, Gaussian, and quasi-likelihood models.
- `"lm"` - Linear regression for continuous outcomes with normally distributed errors.
- `"coxph"` - Cox proportional hazards model for time-to-event survival analysis. Requires `Surv()` outcome syntax.
- `"clogit"` - Conditional logistic regression for matched case-control studies.
- `"negbin"` - Negative binomial regression for overdispersed count data (requires MASS package). Estimates an additional dispersion parameter compared to Poisson regression.
- `"glmer"` - Generalized linear mixed-effects model for hierarchical or clustered data with non-normal outcomes (requires **lme4** package and `random` parameter).
- `"lmer"` - Linear mixed-effects model for hierarchical or clustered data with continuous outcomes (requires **lme4** package and `random` parameter).
- `"coxme"` - Cox mixed-effects model for clustered survival data (requires coxme package and `random` parameter).

`family` For GLM and GLMER models, specifies the error distribution and link function. Can be a character string, a family function, or a family object. Ignored for non-GLM/GLMER models.

Binary/Binomial outcomes:

- `"binomial"` or `binomial()` - Logistic regression for binary outcomes (0/1, TRUE/FALSE). Returns odds ratios (OR). Default.
- `"quasibinomial"` or `quasibinomial()` - Logistic regression with overdispersion. Use when residual deviance » residual df.
- `binomial(link = "probit")` - Probit regression (normal CDF link).
- `binomial(link = "cloglog")` - Complementary log-log link for asymmetric binary outcomes.

Count outcomes:

- `"poisson"` or `poisson()` - Poisson regression for count data. Returns rate ratios (RR). Assumes mean = variance.

- "quasipoisson" or quasipoisson() - Poisson regression with overdispersion. Use when variance > mean.

Continuous outcomes:

- "gaussian" or gaussian() - Normal/Gaussian distribution for continuous outcomes. Equivalent to linear regression.
- gaussian(link = "log") - Log-linear model for positive continuous outcomes. Returns multiplicative effects.

Positive continuous outcomes:

- "Gamma" or Gamma() - Gamma distribution for positive, right-skewed continuous data (*e.g.,* costs, lengths of stay). When passed as a string, resolves to log link for interpretable multiplicative effects.
- Gamma(link = "inverse") - Gamma with inverse (canonical) link.
- Gamma(link = "identity") - Gamma with identity link for additive effects on positive outcomes.
- "inverse.gaussian" or inverse.gaussian() - Inverse Gaussian for positive, highly right-skewed data.

For negative binomial regression (overdispersed counts), use model_type = "negbin" instead of the family parameter.

See family for additional details and options.

random — Character string specifying the random-effects formula for mixed-effects models (glmer, lmer, coxme). Use standard **lme4/coxme** syntax, *e.g.,* "(1|site)" for random intercepts by site, "(1|site/patient)" for nested random effects. Required when model_type is a mixed-effects model type unless random effects are included in the predictors vector. Alternatively, random effects can be included directly in the predictors vector using the same syntax (*e.g.,* predictors = c("age", "sex", "(1|site)")), though they will not be screened as predictors. Default is NULL.

conf_level — Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% CI).

reference_rows — Logical. If TRUE, adds rows for reference categories of factor variables with baseline values. Default is TRUE.

show_n — Logical. If TRUE, includes sample size columns. Default is TRUE.

show_events — Logical. If TRUE, includes events columns (survival and logistic models). Default is TRUE.

digits — Integer specifying decimal places for effect estimates. Default is 2.

p_digits — Integer specifying the number of decimal places for *p*-values. Values smaller than 10^(-p_digits) are displayed as "< 0.001" (for p_digits = 3), "< 0.0001" (for p_digits = 4), etc. Default is 3.

labels — Named character vector or list providing custom display labels for variables. Names should match variable names, values are display labels. Default is NULL.

metrics — Character specification for which statistics to display:

- "both" - Show effect estimates with CI and *p*-values [default]
- "effect" - Show only effect estimates with CI

- "p" - Show only *p*-values

Can also be a character vector: c("effect", "p") is equivalent to "both".

return_type Character string specifying what to return:

- "table" - Return formatted results table only [default]
- "model" - Return multivariable model object only
- "both" - Return list with both table and model

keep_models Logical. If TRUE, stores univariable model objects in the output. Can consume significant memory for many predictors. Default is FALSE.

exponentiate Logical. Whether to exponentiate coefficients. Default is NULL, which automatically exponentiates for logistic, Poisson, and Cox models, and displays raw coefficients for linear models.

conf_method Character string controlling the confidence interval method. If NULL (default), uses getOption("summata.conf_method", "profile").

- "profile" - Profile likelihood intervals for GLM and negative binomial models (via MASS::confint.glm()), exact *t*-distribution intervals for linear models. Falls back to Wald on profiling failure. Quasi-likelihood families always use Wald (no true likelihood).
- "wald" - Wald intervals (coefficient $\pm z \times$ SE) for all model types. Faster but less accurate near boundary conditions or with small subgroups.

Cox and mixed-effects models use Wald intervals regardless of this setting. Set globally with options(summata.conf_method = "wald") to use Wald throughout a session. Note: when method = "screen" and columns = "multi", the internal screening pass always uses Wald since only *p*-values are needed for variable selection.

parallel Logical. If TRUE (default), fits univariable models in parallel using multiple CPU cores for improved performance.

n_cores Integer specifying the number of CPU cores to use for parallel processing. Default is NULL (auto-detect: uses detectCores() - 1). Ignored when parallel = FALSE.

number_format Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- "us" - Comma thousands, period decimal: 1,234.56 [default]
- "eu" - Period thousands, comma decimal: 1.234,56
- "space" - Thin-space thousands, period decimal: 1 234.56 (SI/ISO 31-0)
- "none" - No thousands separator: 1234.56

Or provide a custom two-element vector c(big.mark, decimal.mark), *e.g.*, c("'", ".") for Swiss-style: 1'234.56.

When NULL (default), uses getOption("summata.number_format", "us"). Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

verbose	Logical. If TRUE, displays model fitting warnings (*e.g.,* singular fit, convergence issues). If FALSE (default), routine fitting messages are suppressed while unexpected warnings are preserved. When NULL, uses getOption("summata.verbose", FALSE).
...	Additional arguments passed to model fitting functions (*e.g.,* weights, subset, na.action).

Details

Analysis Workflow:

The function implements a complete regression analysis pipeline:

1. **Univariable screening**: Fits separate models for each predictor (outcome ~ predictor). Each predictor is tested independently to understand crude associations.
2. **Variable selection**: Based on the method parameter:
   - "screen": Automatically selects predictors with univariable p $\leq$ p_threshold
   - "all": Includes all predictors (no selection)
   - "custom": Uses predictors specified in multi_predictors
3. **Multivariable modeling**: Fits a single model with selected predictors (outcome ~ predictor1 + predictor2 + ...). Estimates are adjusted for all other variables in the model.
4. **Output formatting**: Combines results into publication-ready table with appropriate effect measures and formatting.

Variable Selection Strategies:

"Screen" Method (method = "screen"):

- Uses *p*-value threshold for automatic selection
- Liberal thresholds (*e.g.,* 0.20) cast a wide net to avoid missing important predictors
- Stricter thresholds (*e.g.,* 0.05) focus on strongly associated predictors
- Helps reduce overfitting and multicollinearity
- Common in exploratory analyses and when sample size is limited

"All" Method (method = "all"):

- No variable selection; includes all predictors
- Appropriate when all variables are theoretically important
- Risk of overfitting with many predictors relative to sample size
- Useful for confirmatory analyses with pre-specified models

"Custom" Method (method = "custom"):

- Manual selection based on subject matter knowledge
- Runs univariable analysis for all predictors (for comparison)
- Includes only specified predictors in multivariable model
- Ideal for theory-driven model building

- Allows comparison of unadjusted vs adjusted effects for all variables

Interpreting Results:

When `columns = "both"` (default), tables show:

- **Univariable columns**: Crude associations, unadjusted for other variables. Labeled as "OR/HR/RR/Coefficient (95% CI)" and "Uni *p*"
- **Multivariable columns**: Adjusted associations, accounting for all other predictors in the model. Labeled as "aOR/aHR/aRR/Adj. Coefficient (95% CI)" and "Multi *p*" ("a" = adjusted)
- Variables not meeting selection criteria show "-" in multivariable columns

Comparing univariable and multivariable results helps identify:

- **Confounding**: Large changes in effect estimates
- **Independent effects**: Similar univariable and multivariable estimates
- **Mediation**: Attenuated effects in multivariable model
- **Suppression**: Effects that emerge only after adjustment

Sample Size Considerations:

Rule of thumb for multivariable models:

- **Logistic regression**: $\geq$ 10 events per predictor variable
- **Cox regression**: $\geq$ 10 events per predictor variable
- **Linear regression**: $\geq$ 10-20 observations per predictor

Use screening methods to reduce predictor count when these ratios are not met.

Value

Depends on `return_type` parameter:

When `return_type = "table"` (default): A data.table with S3 class `"fullfit_result"` containing:

Variable Character. Predictor name or custom label

Group Character. Category level for factors, empty for continuous

n/n_group Integer. Sample sizes (if `show_n = TRUE`). For variables included in the multivariable model, reflects the complete-case sample size from the fitted model (listwise deletion across all included predictors). For variables not selected into the multivariable model, reflects the per-variable sample size from the univariable analysis. This follows STROBE guideline item 12, which recommends reporting the number of participants included at each stage of analysis.

events/events_group Integer. Event counts (if `show_events = TRUE`). Same complete-case convention as `n`: multivariable rows show events from the fitted model, univariable-only rows show per-variable counts.

OR/HR/RR/Coefficient (95% CI) Character. Unadjusted effect (if `columns` includes "uni" and `metrics` includes "effect")

Uni p Character. Univariable *p*-value (if `columns` includes "uni" and `metrics` includes "p")

aOR/aHR/aRR/Adj. Coefficient (95% CI) Character. Adjusted effect (if `columns` includes "multi" and `metrics` includes "effect")

Multi p Character. Multivariable *p*-value (if `columns` includes "multi" and `metrics` includes "p")

When `return_type = "model"`: The fitted multivariable model object (glm, lm, coxph, *etc.*).

When `return_type = "both"`: A list with two elements:

table The formatted results data.table

model The fitted multivariable model object

The table includes the following attributes:

outcome Character. The outcome variable name

model_type Character. The regression model type

method Character. The variable selection method used

columns Character. Which columns were displayed

model The multivariable model object (if fitted)

uni_results The complete univariable screening results

n_multi Integer. Number of predictors in multivariable model

screened Character vector. Names of predictors that passed univariable screening at the specified *p*-value threshold

significant Character vector. Names of variables with p < 0.05 in the multivariable model (or univariable if multivariable was not fitted)

analysis_counts List. How much of the supplied data reached the models, with elements `n_supplied`, `n_analyzed`, `n_missing_outcome` and `n_missing_predictor`, plus `events_supplied` and `events_analyzed` where the model type carries an event count. The screening models and the multivariable model rest on different samples whenever a predictor carries missing values, so `n_analyzed` holds one value per model and the reported sample may be a range. The `print()` method always reports the analyzed sample

See Also

`uniscreen` for univariable screening only, `fit` for fitting a single multivariable model, `compfit` for comparing multiple models, `desctable` for descriptive statistics, `autoforest` for forest plots of the multivariable models, `forestsave` for exporting forest plots, `tablesave` for exporting tables

Other regression functions: `compfit()`, `fit()`, `multifit()`, `print.compfit_result()`, `print.fit_result()`, `print.fullfit_result()`, `print.multifit_result()`, `print.uniscreen_result()`, `uniscreen()`

Examples

```
# Load example data
data(clintrial)
data(clintrial_labels)

# Example 1: Basic screening with p < 0.05 threshold
result1 <- fullfit(
    data = clintrial,
```

```
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking",
                   "hypertension", "diabetes",
                   "treatment", "stage"),
    method = "screen",
    p_threshold = 0.05,
    labels = clintrial_labels
)
print(result1)



# Example 2: Include all predictors (no selection)
result2 <- fullfit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment", "stage"),
    method = "all",
    labels = clintrial_labels
)
print(result2)

# Example 3: Custom variable selection
result3 <- fullfit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking", "treatment", "stage"),
    method = "custom",
    multi_predictors = c("age", "treatment", "stage"),
    labels = clintrial_labels
)
print(result3)

# Example 4: Cox regression with screening
library(survival)
cox_result <- fullfit(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "treatment", "stage"),
    model_type = "coxph",
    method = "screen",
    p_threshold = 0.10,
    labels = clintrial_labels
)
print(cox_result)

# Example 5: Linear regression without screening
linear_result <- fullfit(
    data = clintrial,
    outcome = "bmi",
    predictors = c("age", "sex", "smoking", "creatinine"),
    model_type = "lm",
    method = "all",
```

```
    labels = clintrial_labels
)
print(linear_result)

# Example 6: Poisson regression for count outcomes
poisson_result <- fullfit(
    data = clintrial,
    outcome = "fu_count",
    predictors = c("age", "stage", "treatment", "surgery"),
    model_type = "glm",
    family = "poisson",
    method = "all",
    labels = clintrial_labels
)
print(poisson_result)

# Example 7: Show only multivariable results
multi_only <- fullfit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment", "stage"),
    method = "all",
    columns = "multi",
    labels = clintrial_labels
)
print(multi_only)

# Example 8: Return both table and model object
both <- fullfit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment", "stage"),
    method = "all",
    return_type = "both",
    labels = clintrial_labels
)
print(both$table)
summary(both$model)

# Example 9: Keep univariable models for diagnostics
with_models <- fullfit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "bmi", "creatinine"),
    keep_models = TRUE
)
uni_results <- attr(with_models, "uni_results")
uni_models <- attr(uni_results, "models")
summary(uni_models[["age"]])

# Example 10: Linear mixed effects with site clustering
if (requireNamespace("lme4", quietly = TRUE)) {
    lmer_result <- fullfit(
```

```
        data = clintrial,
        outcome = "los_days",
        predictors = c("age", "treatment", "surgery", "stage"),
        random = "(1|site)",
        model_type = "lmer",
        method = "all",
        labels = clintrial_labels
    )
    print(lmer_result)
}
```

---

glmforest *Create Forest Plot for Generalized Linear Models*

Description

Generates a publication-ready forest plot that combines a formatted data table with a graphical representation of effect estimates (odds ratios, risk ratios, or coefficients) from a generalized linear model. The plot integrates variable names, group levels, sample sizes, effect estimates with confidence intervals, *p*-values, and model diagnostics in a single comprehensive visualization designed for manuscripts and presentations.

Usage

```
glmforest(
  x,
  data = NULL,
  title = "Generalized Linear Model",
  effect_label = NULL,
  digits = 2,
  p_digits = 3,
  conf_level = 0.95,
  font_size = 1,
  annot_size = 3.88,
  header_size = 5.82,
  title_size = 23.28,
  plot_width = NULL,
  plot_height = NULL,
  table_width = 0.6,
  show_n = TRUE,
  show_events = TRUE,
  indent_groups = FALSE,
  condense_table = FALSE,
  bold_variables = FALSE,
  center_padding = 4,
```

```
  zebra_stripes = TRUE,
  ref_label = "reference",
  labels = NULL,
  color = NULL,
  exponentiate = NULL,
  qc_footer = TRUE,
  units = "in",
  number_format = NULL
)
```

Arguments

x	Either a fitted GLM object (class `glm` or `glmerMod`), a `fit_result` object from `fit()`, or a `fullfit_result` object from `fullfit()`. When a `fit_result` or `fullfit_result` is provided, the model, data, and labels are automatically extracted.
data	Data frame or data.table containing the original data used to fit the model. If `NULL` (default) and `x` is a model, the function attempts to extract data from the model object. If `x` is a `fit_result`, data is extracted automatically. Providing data explicitly is recommended when passing a model directly.
title	Character string specifying the plot title displayed at the top. Default is `"Generalized Linear Model"`. Use descriptive titles like "Risk Factors for Disease Outcome" for publication.
effect_label	Character string for the effect measure label on the forest plot axis. If `NULL` (default), automatically determined based on model family and link function: "Odds Ratio" for logistic regression (`family = binomial, link = logit`), "Risk Ratio" for log-link models, "Exp(Coefficient)" for other exponential families, or "Coefficient" for identity link.
digits	Integer specifying the number of decimal places for effect estimates and confidence intervals in the data table. Default is 2.
p_digits	Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.
conf_level	Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals). The CI percentage is automatically displayed in column headers (*e.g.,* "90% CI" when `conf_level = 0.90`). Intervals are computed by the method that `getOption("summata.conf_method")` selects, as in `fit()`; those of a `fit()` result at the same level are reused.
font_size	Numeric multiplier controlling the base font size for all text elements. Values > 1 increase all fonts proportionally, values < 1 decrease them. Default is 1.0. Useful for adjusting readability across different output sizes.
annot_size	Numeric value controlling the relative font size for data annotations (variable names, values in table cells). Default is 3.88. Adjust relative to `font_size`.
header_size	Numeric value controlling the relative font size for column headers ("Variable", "Group", "n", *etc.*). Default is 5.82. Headers are typically larger than annotations for hierarchy.

`title_size` Numeric value controlling the relative font size for the main plot title. Default is 23.28. The title is typically the largest text element.

`plot_width` Numeric value specifying the intended output width in specified `units`. Used for optimizing layout and text sizing. Default is `NULL` (automatic). Recommended: 10-16 inches for standard publications.

`plot_height` Numeric value specifying the intended output height in specified `units`. Default is `NULL` (automatic based on number of rows). The function provides recommendations if not specified.

`table_width` Numeric value between 0 and 1 specifying the proportion of total plot width allocated to the data table (left side). The forest plot occupies `1 - table_width`. Default is 0.6 (60% table, 40% forest). Increase for longer variable names, decrease to emphasize the forest plot.

`show_n` Logical. If `TRUE`, includes a column showing group-specific sample sizes for categorical variables and the fitted sample size for continuous variables. Counts describe the observations used in fitting rather than every row supplied. Default is `TRUE`.

`show_events` Logical. If `TRUE`, includes a column showing the number of events for each group. Relevant for logistic regression (number of cases) and other binary outcomes. Default is `TRUE`.

`indent_groups` Logical. If `TRUE`, indents factor levels under their parent variable name, creating a hierarchical visual structure. When `TRUE`, the "Group" column is hidden. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses binary categorical variables into single rows by showing only the non-reference category. Automatically sets `indent_groups = TRUE`. Useful for tables with many binary variables. Default is `FALSE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. If `FALSE` (default), variable names are displayed in plain text.

`center_padding` Numeric value specifying the horizontal spacing (in character units) between the data table and forest plot. Increase for more separation, decrease to fit more content. Default is 4.

`zebra_stripes` Logical. If `TRUE`, applies alternating gray background shading to different variables (not rows) to improve visual grouping and readability. Default is `TRUE`.

`ref_label` Character string to display for reference categories of factor variables. Typically shown in place of effect estimates. Default is `"reference"`. Common alternatives: "ref", "1.00 (ref)".

`labels` Named character vector or list providing custom display labels for variables. Names should match variable names in the model, values are the labels to display. Example: `c(age = "Age (years)", bmi = "Body Mass Index")`. Default is `NULL`, in which case the labels attached by the function that produced `x` are used, where `x` is a **summata** result rather than a model object. Original variable names are used where neither is available.

`color` Character string specifying the color for effect estimate point markers in the forest plot. Use hex codes or R color names. Default is `NULL`, which auto-selects based on effect type: `"#4BA6B6"` (teal) for odds ratios (binomial/quasibinomial with logit link), `"#3F87EE"` (blue) for rate/risk ratios (Poisson, Gamma, inverse

Gaussian with log link), and "#5A8F5A" (green) for coefficients (Gaussian/identity link). This scheme matches uniforest() and multiforest(). Choose colors that contrast well with black error bars.

exponentiate — Logical. If TRUE, exponentiates coefficients to display odds ratios, risk ratios, *etc.* If FALSE, shows raw coefficients. Default is NULL, which automatically exponentiates for logit, log, and cloglog links, and shows raw coefficients for identity link.

qc_footer — Logical. If TRUE, displays model quality control statistics in the footer (observations analyzed, model family, deviance, pseudo-R^2, AIC). Default is TRUE.

units — Character string specifying the units for plot dimensions. Options: "in" (inches), "cm" (centimeters), "mm" (millimeters). Default is "in". Affects interpretation of plot_width and plot_height.

number_format — Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- "us" - Comma thousands, period decimal: 1,234.56 [default]
- "eu" - Period thousands, comma decimal: 1.234,56
- "space" - Thin-space thousands, period decimal: 1 234.56 (SI/ISO 31-0)
- "none" - No thousands separator: 1234.56

Or provide a custom two-element vector c(big.mark, decimal.mark), *e.g.*, c("'", ".") for Swiss-style: 1'234.56.

When NULL (default), uses getOption("summata.number_format", "us"). Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

Details

Plot Components:

The forest plot consists of several integrated components:

1. **Title**: Centered at top, describes the analysis
2. **Data Table** (left side): Contains columns for:
   - Variable: Predictor names (or custom labels)
   - Group: Factor levels (optional, hidden when indenting)
   - n: Sample sizes by group (optional)
   - Events: Event counts by group (optional)
   - Effect (95% CI); *p*-value: Formatted estimates with *p*-values
3. **Forest Plot** (right side): Graphical display with:
   - Point estimates (squares sized by sample size)
   - 95% confidence intervals (error bars)
   - Reference line (at OR/RR = 1 or coefficient = 0)
   - Log scale for odds/risk ratios
   - Labeled axis

4. **Model Statistics** (footer): Summary of:
    - Observations analyzed (with total observations and percentage)
    - Model family (Binomial, Poisson, *etc.*)
    - Deviance statistics
    - Pseudo-R^2 (McFadden)
    - AIC

Automatic Effect Measure Selection:

When `effect_label = NULL` and `exponentiate = NULL`, the function intelligently selects the appropriate effect measure:

- **Logistic regression** (`family = binomial(link = "logit")`): Odds Ratios (OR)
- **Log-link models** (`link = "log"`): Risk Ratios (RR) or Rate Ratios
- **Other exponential families**: exp(coefficient)
- **Identity link**: Raw coefficients

Reference Categories:

For factor variables, the first level (determined by factor ordering or alphabetically for character variables) serves as the reference category:

- Displayed with the `ref_label` instead of an estimate
- No confidence interval or *p*-value shown
- Visually aligned with other categories
- When `condense_table = TRUE`, reference-only variables may be omitted entirely

Aliased Coefficients:

Coefficients that the model could not estimate because their columns of the model matrix are linear combinations of others are omitted, as in the tables of `fit()`, together with any factor none of whose levels was estimated.

Layout Optimization:

The function automatically optimizes layout based on content:

- Calculates appropriate axis ranges to accommodate all confidence intervals
- Selects meaningful tick marks on log or linear scales
- Sizes point markers proportional to sample size (larger = more data)
- Adjusts table width based on variable name lengths when `table_width = NULL`
- Recommends overall dimensions based on number of rows

Visual Grouping Options:

Three display modes are available:

1. **Standard** (`indent_groups = FALSE`, `condense_table = FALSE`): Separate "Variable" and "Group" columns, all categories shown
2. **Indented** (`indent_groups = TRUE`, `condense_table = FALSE`): Hierarchical display with groups indented under variables

3. **Condensed** (`condense_table = TRUE`): Binary variables shown in single rows, automatically indented

Zebra Striping:

When `zebra_stripes = TRUE`, alternating variables (not individual rows) receive light gray backgrounds. This helps visually group all levels of a factor variable together, making the plot easier to read especially with many multi-level factors.

Model Statistics Display:

The footer shows key diagnostic information:

- **Observations analyzed**: Observations used in fitting, the total supplied, and the percentage retained (accounting for missing values)
- **Null/Residual Deviance**: Model fit improvement
- **Pseudo-R^2**: McFadden R^2 = 1 - (log L_1 / log L_2)
- **AIC**: For model comparison (lower is better)

For logistic regression, concordance (C-statistic/AUC) may also be displayed if available.

Saving Plots:

Use `forestsave()`, which applies the recommended dimensions and selects a graphics device suited to the format:

```
p <- glmforest(model, data)
forestsave(p, "forest.pdf")
```

Or specify custom dimensions:

```
forestsave(p, "forest.png", width = 12, height = 8, dpi = 300)
```

Value

A `ggplot` object containing the complete forest plot. The plot can be:

- Displayed directly: `print(plot)`
- Saved to file: `forestsave(plot, "forest.pdf")`
- Further customized with **ggplot2** functions

The returned object includes an attribute `"rec_dims"` accessible via `attr(plot, "rec_dims")`, which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if `plot_width` or `plot_height` are not specified. The list also carries a `units` element recording the units the dimensions are expressed in, matching the `units` argument. `forestsave()` reads all three and requires no further handling.

The returned object also includes an attribute `"table_data"` accessible via `attr(plot, "table_data")`, a data.table holding the values drawn in the plot: one row per term in model order, with the variable, factor level, sample size, event count, estimate, confidence bounds and p-value. Sample sizes and event counts describe the observations used in fitting rather than every row supplied, so they sum to the model sample size within each variable.

See Also

autoforest for automatic model detection, coxforest for Cox proportional hazards forest plots, lmforest for linear model forest plots, uniforest for univariable screening forest plots, multiforest for multi-outcome forest plots, glm for fitting GLMs, fit for regression modeling, forestsave for saving with recommended dimensions

Other visualization functions: autoforest(), coxforest(), forestsave(), lmforest(), multiforest(), recdims(), uniforest()

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example model
model1 <- glm(os_status ~ age + sex + bmi + treatment,
              data = clintrial, family = binomial)

# Example 1: Basic logistic regression forest plot
p <- glmforest(model1, data = clintrial)



old_width <- options(width = 180)

# Example 2: With custom variable labels
plot2 <- glmforest(
    x = model1,
    data = clintrial,
    title = "Risk Factors for Mortality",
    labels = clintrial_labels
)

# Example 3: Indented layout with formatting options
plot3 <- glmforest(
    x = model1,
    data = clintrial,
    indent_groups = TRUE,
    zebra_stripes = TRUE,
    color = "#D62728",
    labels = clintrial_labels
)

# Example 4: Condensed layout for many binary variables
model4 <- glm(os_status ~ age + sex + smoking + hypertension +
                  diabetes + surgery,
              data = clintrial,
              family = binomial)

plot4 <- glmforest(
    x = model4,
    data = clintrial,
```

```
    condense_table = TRUE,
    labels = clintrial_labels
)
# Binary variables shown in single rows

# Example 5: Poisson regression for count data
model5 <- glm(ae_count ~ age + treatment + diabetes + surgery,
              data = clintrial,
              family = poisson)

plot5 <- glmforest(
    x = model5,
    data = clintrial,
    title = "Rate Ratios for Adverse Events",
    labels = clintrial_labels
)

# Example 6: Save with recommended dimensions
forestsave(plot5, file.path(tempdir(), "forest.pdf"))

options(old_width)
```

`lmforest` *Create Forest Plot for Linear Models*

Description

Generates a publication-ready forest plot that combines a formatted data table with a graphical representation of regression coefficients from a linear model. The plot integrates variable names, group levels, sample sizes, coefficients with confidence intervals, p-values, and model diagnostics (R^2, F-statistic, AIC) in a single comprehensive visualization designed for manuscripts and presentations.

Usage

```
lmforest(
  x,
  data = NULL,
  title = "Linear Model",
  effect_label = "Coefficient",
  digits = 2,
  p_digits = 3,
  conf_level = 0.95,
  font_size = 1,
  annot_size = 3.88,
  header_size = 5.82,
  title_size = 23.28,
  plot_width = NULL,
```

```
  plot_height = NULL,
  table_width = 0.6,
  show_n = TRUE,
  indent_groups = FALSE,
  condense_table = FALSE,
  bold_variables = FALSE,
  center_padding = 4,
  zebra_stripes = TRUE,
  ref_label = "reference",
  labels = NULL,
  units = "in",
  color = "#5A8F5A",
  qc_footer = TRUE,
  number_format = NULL
)
```

Arguments

x	Either a fitted linear model object (class lm or lmerMod), a fit_result object from fit(), or a fullfit_result object from fullfit(). When a fit_result or fullfit_result is provided, the model, data, and labels are automatically extracted.
data	Data frame or data.table containing the original data used to fit the model. If NULL (default) and x is a model, the function attempts to extract data from the model object. If x is a fit_result, data is extracted automatically. Providing data explicitly is recommended when passing a model directly.
title	Character string specifying the plot title displayed at the top. Default is "Linear Model".
effect_label	Character string for the effect measure label on the forest plot axis. Default is "Coefficient".
digits	Integer specifying the number of decimal places for coefficients and confidence intervals. Default is 2.
p_digits	Integer specifying the number of decimal places for *p*-values. Values smaller than 10^(-p_digits) are displayed as "< 0.001" (for p_digits = 3), "< 0.0001" (for p_digits = 4), etc. Default is 3.
conf_level	Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals). The CI percentage is automatically displayed in column headers (*e.g.,* "90% CI" when conf_level = 0.90). Intervals are computed by the method that getOption("summata.conf_method") selects, as in fit(); those of a fit() result at the same level are reused.
font_size	Numeric multiplier controlling the base font size for all text elements. Default is 1.0.
annot_size	Numeric value controlling the relative font size for data annotations. Default is 3.88.
header_size	Numeric value controlling the relative font size for column headers. Default is 5.82.

`title_size` Numeric value controlling the relative font size for the main plot title. Default is 23.28.

`plot_width` Numeric value specifying the intended output width in specified `units`. Default is `NULL` (automatic).

`plot_height` Numeric value specifying the intended output height in specified `units`. Default is `NULL` (automatic).

`table_width` Numeric value between 0 and 1 specifying the proportion of total plot width allocated to the data table. Default is 0.6.

`show_n` Logical. If `TRUE`, includes a column showing group-specific sample sizes. Counts describe the observations used in fitting rather than every row supplied. Default is `TRUE`.

`indent_groups` Logical. If `TRUE`, indents factor levels under their parent variable name, creating hierarchical structure. The "Group" column is hidden when `TRUE`. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses binary categorical variables into single rows. Automatically sets `indent_groups = TRUE`. Default is `FALSE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. If `FALSE` (default), variable names are displayed in plain text.

`center_padding` Numeric value specifying horizontal spacing between table and forest plot. Default is 4.

`zebra_stripes` Logical. If `TRUE`, applies alternating gray background shading to different variables. Default is `TRUE`.

`ref_label` Character string to display for reference categories of factor variables. Default is `"reference"`.

`labels` Named character vector providing custom display labels for variables. Example: `c(age = "Age (years)", height = "Height (cm)")`. Default is `NULL`, in which case the labels attached by the function that produced `x` are used, where `x` is a **summata** result rather than a model object. Original variable names are used where neither is available.

`units` Character string specifying units for plot dimensions: `"in"` (inches), `"cm"`, or `"mm"`. Default is `"in"`.

`color` Character string specifying the color for coefficient point estimates in the forest plot. Default is `"#5A8F5A"` (green). Use hex codes or R color names.

`qc_footer` Logical. If `TRUE`, displays model quality control statistics in the footer (observations analyzed, R^2, adjusted R^2, F-statistic, AIC). Default is `TRUE`.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

Details

Linear Model-Specific Features:

The linear model forest plot differs from logistic and Cox plots in several ways:

- **Coefficients**: Raw regression coefficients shown (not exponentiated)
- **Reference line**: At coefficient = 0 (not at 1)
- **Linear scale**: Forest plot uses linear scale (not log scale)
- **No events column**: Only sample sizes shown (no event counts)
- **R^2 statistics**: Model fit assessed by R^2 and adjusted R^2
- ***F*-test**: Overall model significance from F-statistic

Plot Components:

1. **Title**: Centered at top
2. **Data Table** (left): Contains:
   - Variable: Predictor names
   - Group: Factor levels (if applicable)
   - n: Sample sizes by group
   - Coefficient (95% CI); p-value: Raw coefficients with CIs and p-values
3. **Forest Plot** (right):
   - Point estimates (squares sized by sample size)
   - 95% confidence intervals (error bars)
   - Reference line at coefficient = 0
   - Linear scale
4. **Model Statistics** (footer):
   - Observations analyzed (with total observations and percentage)
   - R^2 and adjusted R^2
   - F-statistic with degrees of freedom and p-value
   - AIC

Interpreting Coefficients:

Linear regression coefficients represent the change in the outcome variable for a one-unit change in the predictor:

- **Continuous predictors**: Coefficient = change in Y per unit of X
- **Binary predictors**: Coefficient = difference in Y between groups

- **Factor predictors**: Coefficients = differences from reference category
- **Sign matters**: Positive = increase in Y, Negative = decrease in Y
- **Zero crossing**: CI crossing zero suggests no significant effect

Example: If the coefficient for "age" is 0.50 when predicting BMI, BMI increases by 0.50 kg/m^2 for each additional year of age.

Model Fit Statistics:

The footer displays key diagnostics:

- R^2: Proportion of variance explained (0 to 1)
  - 0.0-0.3: Weak explanatory power
  - 0.3-0.5: Moderate
  - 0.5-0.7: Good
  - > 0.7: Strong (rare in social/biological sciences)
- **Adjusted R^2**: R^2 penalized for number of predictors
  - Always $\leq R^2$
  - Preferred for model comparison
  - Accounts for model complexity
- ***F*-statistic**: Tests null hypothesis that all coefficients = 0
  - Degrees of freedom: df1 = # predictors, df2 = # observations - # predictors - 1
  - Significant p-value indicates model explains variance better than intercept-only
- **AIC**: For model comparison (lower is better)

Assumptions:

Linear regression assumes:

1. Linearity of relationships
2. Independence of observations
3. Homoscedasticity (constant variance)
4. Normality of residuals
5. No multicollinearity

Check assumptions using:

- `plot(model)` for diagnostic plots
- `car::vif(model)` for multicollinearity
- `lmtest::bptest(model)` for heteroscedasticity
- `shapiro.test(residuals(model))` for normality

Reference Categories:

For factor variables:

- First level is the reference (coefficient = 0)
- Other levels show difference from reference

- Reference displayed with `ref_label`
- Relevel factors before modeling if needed: `factor(x, levels = c("desired_ref", ...))`

Aliased Coefficients:

Coefficients that the model could not estimate because their columns of the model matrix are linear combinations of others are omitted, as in the tables of `fit()`, together with any factor none of whose levels was estimated.

Sample Size Reporting:

The "*n*" column shows:

- For continuous variables: Total observations with non-missing data
- For factor variables: Number of observations in each category
- Footer shows observations used in fitting, the total supplied, and the percentage retained (accounting for missing values)

Value

A `ggplot` object containing the complete forest plot. The plot can be:

- Displayed directly: `print(plot)`
- Saved to file: `forestsave(plot, "forest.pdf")`
- Further customized with **ggplot2** functions

The returned object includes an attribute `"rec_dims"` accessible via `attr(plot, "rec_dims")`, which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if `plot_width` or `plot_height` are not specified. The list also carries a `units` element recording the units the dimensions are expressed in, matching the `units` argument. `forestsave()` reads all three and requires no further handling.

The returned object also includes an attribute `"table_data"` accessible via `attr(plot, "table_data")`, a data.table holding the values drawn in the plot: one row per term in model order, with the variable, factor level, sample size, estimate, confidence bounds and p-value. Sample sizes describe the observations used in fitting rather than every row supplied, so they sum to the model sample size within each variable.

See Also

`autoforest` for automatic model detection, `glmforest` for logistic/GLM forest plots, `coxforest` for Cox model forest plots, `uniforest` for univariable screening forest plots, `multiforest` for multi-outcome forest plots, `lm` for fitting linear models, `fit` for regression modeling, `forestsave` for saving with recommended dimensions

Other visualization functions: `autoforest()`, `coxforest()`, `forestsave()`, `glmforest()`, `multiforest()`, `recdims()`, `uniforest()`

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example model
model1 <- lm(bmi ~ age + sex + smoking, data = clintrial)

# Example 1: Basic linear model forest plot
p <- lmforest(model1, data = clintrial)



old_width <- options(width = 180)

# Example 2: With custom labels and title
plot2 <- lmforest(
    x = model1,
    data = clintrial,
    title = "Predictors of Body Mass Index",
    effect_label = "Change in BMI (kg/m^2)",
    labels = clintrial_labels
)

# Example 3: Comprehensive model with indented layout
model3 <- lm(
    bmi ~ age + sex + smoking + hypertension + diabetes + creatinine,
    data = clintrial
)

plot3 <- lmforest(
    x = model3,
    data = clintrial,
    labels = clintrial_labels,
    indent_groups = TRUE,
    zebra_stripes = TRUE
)

# Example 4: Condensed layout
plot4 <- lmforest(
    x = model3,
    data = clintrial,
    condense_table = TRUE,
    labels = clintrial_labels
)

# Example 5: Different outcome (hemoglobin)
model5 <- lm(
    hemoglobin ~ age + sex + bmi + smoking + creatinine,
    data = clintrial
)

plot5 <- lmforest(
```

```
    x = model5,
    data = clintrial,
    title = "Predictors of Baseline Hemoglobin",
    effect_label = "Change in Hemoglobin (g/dL)",
    labels = clintrial_labels
)

# Example 6: Save with recommended dimensions
forestsave(plot5, file.path(tempdir(), "linear_forest.pdf"))

options(old_width)
```

`m2dt` *Convert Model to Data Table*

Description

Extracts coefficients, confidence intervals, and comprehensive model statistics from fitted regression models and converts them to a standardized data.table format suitable for further analysis or publication. This is a core utility function frequently used internally by other **summata** regression functions, although it can be used as a standalone function as well.

Usage

```
m2dt(
  data,
  model,
  conf_level = 0.95,
  keep_qc_stats = TRUE,
  include_intercept = TRUE,
  terms_to_exclude = NULL,
  reference_rows = TRUE,
  reference_label = "reference",
  skip_counts = FALSE,
  conf_method = NULL
)
```

Arguments

`data`	Data frame or data.table containing the dataset used to fit the model. Required for computing group-level sample sizes and event counts. Where the fitted object retains no model frame, the supplied data is restricted to complete cases across the model variables so that group-level sample sizes describe the observations actually used in fitting. Group sample sizes therefore sum to the model-level `n`, and both exclude observations with a missing response.
`model`	Fitted model object. Supported classes include:

- `glm` - Generalized linear models (logistic, Poisson, *etc.*)
- `lm` - Linear models
- `coxph` - Cox proportional hazards models
- `clogit` - Conditional logistic regression
- `coxme` - Mixed effects Cox models
- `lmerMod` - Linear mixed effects models
- `glmerMod` - Generalized linear mixed effects models

`conf_level` Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% CI).

`keep_qc_stats` Logical. If `TRUE`, includes model quality statistics such as AIC, BIC, R^2, concordance, and model fit tests. These appear as additional columns in the output. Default is `TRUE`.

`include_intercept`
Logical. If `TRUE`, includes the model intercept in output. If `FALSE`, removes the intercept row from results. Useful for creating cleaner presentation tables. Default is `TRUE`.

`terms_to_exclude`
Character vector of term names to exclude from output. Useful for removing specific unwanted parameters (*e.g.,* nuisance variables, spline terms). Default is `NULL`. Note: If `include_intercept = FALSE`, "(Intercept)" is automatically added to this list.

`reference_rows` Logical. If `TRUE`, adds rows for reference categories of factor variables with appropriate labels and baseline values (OR/HR = 1, Coefficient = 0). This makes tables more complete and easier to interpret. Default is `TRUE`.

`reference_label`
Character string used to label reference category rows in the output. Appears in the `reference` column. Default is `"reference"`.

`skip_counts` Logical. If `TRUE`, skips computation of group-level sample sizes and event counts (faster but less informative). Default is `FALSE`.

`conf_method` Character string controlling the confidence interval method. If `NULL` (default), uses `getOption("summata.conf_method", "profile")`.

- `"profile"` - Profile likelihood intervals for GLM and negative binomial models (via `MASS::confint.glm()`), exact t-distribution intervals for linear models. Falls back to Wald on profiling failure. Quasi-likelihood families always use Wald (no true likelihood).
- `"wald"` - Wald intervals (coefficient $\pm z \times$ SE) for all model types. Faster but less accurate near boundary conditions or with small subgroups.

Cox and mixed-effects models use Wald intervals regardless of this setting. Set globally with `options(summata.conf_method = "wald")` to use Wald throughout a session.

Details

This function is the core extraction utility used by `fit()` and other regression functions. It handles the complexities of different model classes and provides a consistent output format suitable for tables and forest plots.

Model Type Detection: The function automatically detects model type and applies appropriate:

- Effect measure naming (OR, HR, RR, Coefficient)
- Confidence interval calculation (see below)
- Event counting for binary/survival outcomes

Confidence Interval Methods: The CI method is selected per model class using `stats::confint()` dispatch:

- **GLM/negative binomial**: Profile likelihood via `MASS::confint.glm()`, except quasi-families which use Wald
- **Linear models**: Exact t-distribution via `confint.lm()`
- **Cox PH**: Wald intervals (coefficient $\pm z \times$ SE)
- **Mixed-effects models**: Wald intervals

Falls back to Wald on profiling failure.

Mixed Effects Models: For **lme4** models (glmer, lmer), the function extracts fixed effects only. Random effects variance components are not included in the output table, as they represent clustering structure rather than predictor effects.

Aliased Coefficients: Coefficients that the model could not estimate because their columns of the model matrix are linear combinations of others (*e.g.,* a predictor entered twice under different names) are omitted for every model class, as is the reference row of a factor none of whose levels was estimated.

Value

A `data.table` containing extracted model information with the following standard columns:

model_scope Character. Either "Univariable" (unadjusted model with single predictor) or "Multivariable" (adjusted model with multiple predictors)

model_type Character. Type of regression (*e.g.,* "Logistic", "Linear", "Cox PH", "Poisson", *etc.*)

variable Character. Variable name (for factor variables, the base variable name without the level)

group Character. Group/level name for factor variables; empty string for continuous variables

n Integer. Number of observations used in fitting the model, that is complete cases across the model variables; for the levels of a factor, the number at that level, as in `n_group`

n_group Integer. Number of those observations at this variable level (factor variables only). Group sizes sum to `n` within each variable

events Integer. Number of events among the observations used in fitting (for survival and logistic models); for the levels of a factor, the number at that level, as in `events_group`

events_group Integer. Number of events at this variable level (for survival and logistic models with factor variables). Group counts sum to `events` within each variable

coefficient Numeric. Raw regression coefficient (log odds, log hazard, *etc.*)

se Numeric. Standard error of the coefficient

OR/HR/RR/Coefficient Numeric. Effect estimate - column name depends on model type:

- `OR` for logistic regression (odds ratio)

- HR for Cox models (hazard ratio)
- RR for Poisson regression (rate/risk ratio)
- Coefficient for linear models or other GLMs

ci_lower Numeric. Lower bound of confidence interval for effect estimate

ci_upper Numeric. Upper bound of confidence interval for effect estimate

statistic Numeric. Test statistic (z-value for GLM/Cox, t-value for LM)

p_value Numeric. p-value for coefficient test

sig Character. Significance markers: *** (p < 0.001), ** ($p < 0.01$), * ($p < 0.05$), . ($p < 0.10$).

sig_binary Logical. Binary indicator: TRUE if $p < 0.05$, FALSE otherwise

reference Character. Contains reference_label for reference category rows when reference_rows = TRUE, empty string otherwise

The model-level sample size and number of events are also recorded in the attributes "model_n" and "model_events", since the n and events columns of factor levels hold group counts.

See Also

fit for the main regression interface, glmforest, coxforest, lmforest for forest plot visualization

Examples

```
# Load example data
data(clintrial)

# Example 1: Extract from logistic regression
glm_model <- glm(os_status ~ age + sex + treatment,
                 data = clintrial, family = binomial)

glm_result <- m2dt(clintrial, glm_model)
glm_result



# Example 2: Extract from linear model
lm_model <- lm(los_days ~ age + sex + surgery, data = clintrial)

lm_result <- m2dt(clintrial, lm_model)
lm_result

# Example 3: Cox proportional hazards model
library(survival)
cox_model <- coxph(Surv(os_months, os_status) ~ age + sex + stage,
                   data = clintrial)

cox_result <- m2dt(clintrial, cox_model)
cox_result

# Example 4: Exclude intercept for cleaner tables
```

```
clean_result <- m2dt(clintrial, glm_model, include_intercept = FALSE)
clean_result

# Example 5: Change confidence level
result_90ci <- m2dt(clintrial, glm_model, conf_level = 0.90)
result_90ci
```

`multifit` *Multivariate Regression Analysis*

Description

Performs regression analyses of a single predictor (exposure) across multiple outcomes. This function is designed for studies where a single exposure variable is tested against multiple endpoints, such as complication screening, biomarker associations, or phenome-wide association studies. Returns publication-ready formatted results with optional covariate adjustment. Supports interactions, mixed-effects models, stratification, and clustered standard errors.

Usage

```
multifit(
  data,
  outcomes,
  predictor,
  covariates = NULL,
  interactions = NULL,
  random = NULL,
  strata = NULL,
  cluster = NULL,
  model_type = "glm",
  family = "binomial",
  columns = "adjusted",
  p_threshold = 1,
  conf_level = 0.95,
  show_n = TRUE,
  show_events = TRUE,
  digits = 2,
  p_digits = 3,
  labels = NULL,
  predictor_label = NULL,
  include_predictor = TRUE,
  keep_models = FALSE,
  exponentiate = NULL,
  conf_method = NULL,
  parallel = TRUE,
```

```
  n_cores = NULL,
  number_format = NULL,
  verbose = NULL,
  ...
)
```

Arguments

data	Data frame or data.table containing the analysis dataset. The function automatically converts data frames to data.tables for efficient processing.
outcomes	Character vector of outcome variable names to analyze. Each outcome is tested in its own model with the predictor. For time-to-event analysis, use `Surv()` syntax for the outcome variable (*e.g.,* `c("Surv(time1, status1)", "Surv(time2, status2)")`).
predictor	Character string specifying the predictor (exposure) variable name. This variable is tested against each outcome. Can be continuous or categorical (factor). Non-syntactic names (*e.g.,* `"tumor stage"`) are quoted in the model formula and need no backticks.
covariates	Optional character vector of covariate variable names to include in adjusted models. When specified, models are fit as `outcome ~ predictor + covariate1 + covariate2 + ...`, and only the predictor effect is reported. Default is `NULL` (unadjusted models).
interactions	Optional character vector of interaction terms to include in adjusted models, using colon notation (*e.g.,* `c("predictor:sex")`). Interactions involving the predictor will have their effects extracted and reported. Default is `NULL`.
random	Optional character string specifying random effects formula for mixed effects models (*e.g.,* `"(1|hospital)"` or `"(1|site/patient)"`). Required when `model_type` is `"glmer"`, `"lmer"`, or `"coxme"` unless random effects are included in the `covariates` vector. Alternatively, random effects can be included directly in the `covariates` vector using the same syntax (*e.g.,* `covariates = c("age", "sex", "(1|site)")`). Default is `NULL`.
strata	Optional character string naming the stratification variable for Cox or conditional logistic models. Creates separate baseline hazards for each stratum. Default is `NULL`.
cluster	Optional character string naming the clustering variable for Cox models. Computes robust clustered standard errors. Default is `NULL`.
model_type	Character string specifying the type of regression model to fit. Options include: • `"glm"` - Generalized linear model (default). Supports multiple distributions via the `family` parameter including logistic, Poisson, Gamma, Gaussian, and quasi-likelihood models. • `"lm"` - Linear regression for continuous outcomes with normally distributed errors. • `"coxph"` - Cox proportional hazards model for time-to-event survival analysis. Requires `Surv()` outcome syntax. • `"clogit"` - Conditional logistic regression for matched case-control studies.

- "negbin" - Negative binomial regression for overdispersed count data (requires MASS package). Estimates an additional dispersion parameter compared to Poisson regression.
- "glmer" - Generalized linear mixed-effects model for hierarchical or clustered data with non-normal outcomes (requires **lme4** package and random parameter).
- "lmer" - Linear mixed-effects model for hierarchical or clustered data with continuous outcomes (requires **lme4** package and random parameter).
- "coxme" - Cox mixed-effects model for clustered survival data (requires coxme package and random parameter).

family For GLM and GLMER models, specifies the error distribution and link function. Can be a character string, a family function, or a family object. Ignored for non-GLM/GLMER models.

Binary/Binomial outcomes:

- "binomial" or binomial() - Logistic regression for binary outcomes (0/1, TRUE/FALSE). Returns odds ratios (OR). Default.
- "quasibinomial" or quasibinomial() - Logistic regression with overdispersion. Use when residual deviance » residual df.
- binomial(link = "probit") - Probit regression (normal CDF link).
- binomial(link = "cloglog") - Complementary log-log link for asymmetric binary outcomes.

Count outcomes:

- "poisson" or poisson() - Poisson regression for count data. Returns rate ratios (RR). Assumes mean = variance.
- "quasipoisson" or quasipoisson() - Poisson regression with overdispersion. Use when variance > mean.

Continuous outcomes:

- "gaussian" or gaussian() - Normal/Gaussian distribution for continuous outcomes. Equivalent to linear regression.
- gaussian(link = "log") - Log-linear model for positive continuous outcomes. Returns multiplicative effects.

Positive continuous outcomes:

- "Gamma" or Gamma() - Gamma distribution for positive, right-skewed continuous data (*e.g.,* costs, lengths of stay). When passed as a string, resolves to log link for interpretable multiplicative effects.
- Gamma(link = "inverse") - Gamma with inverse (canonical) link.
- Gamma(link = "identity") - Gamma with identity link for additive effects on positive outcomes.
- "inverse.gaussian" or inverse.gaussian() - Inverse Gaussian for positive, highly right-skewed data.

For negative binomial regression (overdispersed counts), use model_type = "negbin" instead of the family parameter.

See family for additional details and options.

columns
: Character string specifying which result columns to display when both unadjusted and adjusted models are fit (*i.e.,* when `covariates` is specified):

  - `"adjusted"` - Show only adjusted (covariate-controlled) results [default]
  - `"unadjusted"` - Show only unadjusted (crude) results
  - `"both"` - Show both unadjusted and adjusted results side-by-side

  Ignored when `covariates = NULL`.

p_threshold
: Numeric value between 0 and 1 specifying a *p*-value threshold for filtering results. Only outcomes with *p*-value less than or equal to the threshold are included in the output. Default is 1 (no filtering; all outcomes returned).

conf_level
: Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals).

show_n
: Logical. If `TRUE`, includes the sample size column in the output table. Default is `TRUE`.

show_events
: Logical. If `TRUE`, includes the events column in the output table (relevant for survival and logistic regression). Default is `TRUE`.

digits
: Integer specifying the number of decimal places for effect estimates (OR, HR, RR, coefficients). Default is 2.

p_digits
: Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

labels
: Named character vector or list providing custom display labels for variables. Can include labels for outcomes, predictors, and covariates. Names should match variable names, values are the display labels. Labels are applied to: (1) outcome names in the Outcome column, (2) predictor variable name when displayed, and (3) variable names in formatted interaction terms. Variables not in `labels` use their original names. Default is `NULL`.

predictor_label
: Optional character string providing a custom display label for the predictor variable. Takes precedence over `labels` for the predictor. Default is `NULL` (uses label from `labels` or original name).

include_predictor
: Logical. If `TRUE` (default), includes the predictor column in the output table showing which level of a factor predictor is being compared. If `FALSE`, omits the predictor column, which may be useful when the predictor information will be explained in a table caption or figure legend.

keep_models
: Logical. If `TRUE`, stores all fitted model objects in the output as an attribute. Models are accessible via `attr(result, "models")`. Default is `FALSE`.

exponentiate
: Logical. Whether to exponentiate coefficients (display OR/HR/RR instead of log odds/log hazards). Default is `NULL`, which automatically displays raw coefficients for linear models and exponentiates for logistic, Poisson, and Cox models.

conf_method
: Character string controlling the confidence interval method. If `NULL` (default), uses `getOption("summata.conf_method", "profile")`.

- "profile" - Profile likelihood intervals for GLM and negative binomial models (via MASS::confint.glm()), exact t-distribution intervals for linear models. Falls back to Wald on profiling failure. Quasi-likelihood families always use Wald (no true likelihood).
- "wald" - Wald intervals (coefficient $\pm z \times$ SE) for all model types. Faster but less accurate near boundary conditions or with small subgroups.

Cox and mixed-effects models use Wald intervals regardless of this setting. Set globally with options(summata.conf_method = "wald") to use Wald throughout a session.

parallel — Logical. If TRUE (default), fits models in parallel for improved performance with many outcomes.

n_cores — Integer specifying the number of CPU cores to use for parallel processing. Default is NULL (auto-detect: uses detectCores() - 1). Ignored when parallel = FALSE.

number_format — Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- "us" - Comma thousands, period decimal: 1,234.56 [default]
- "eu" - Period thousands, comma decimal: 1.234,56
- "space" - Thin-space thousands, period decimal: 1 234.56 (SI/ISO 31-0)
- "none" - No thousands separator: 1234.56

Or provide a custom two-element vector c(big.mark, decimal.mark), *e.g.*, c("'", ".") for Swiss-style: 1'234.56.

When NULL (default), uses getOption("summata.number_format", "us"). Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

verbose — Logical. If TRUE, displays model fitting warnings (*e.g.,* singular fit, convergence issues). If FALSE (default), routine fitting messages are suppressed while unexpected warnings are preserved. When NULL, uses getOption("summata.verbose", FALSE).

... — Additional arguments passed to the underlying model fitting functions.

Details

Analysis Approach:

The function implements a multivariate (multi-outcome) screening workflow that inverts the typical regression paradigm:

1. For each outcome in outcomes, fits a separate model with the predictor as the main exposure
2. If covariates specified, fits adjusted model: outcome ~ predictor + covariates + interactions
3. Extracts only the predictor effect(s) from each model, ignoring covariate coefficients
4. Combines results into a single table for comparison across outcomes
5. Optionally filters by p-value threshold

This is conceptually opposite to `uniscreen()`, which tests multiple predictors against a single outcome. `multifit()` applies where there is one exposure of interest and want to screen across multiple endpoints.

When to Use Multivariate Regression Analysis:

- **Complication screening**: Test one exposure (*e.g.,* operative time, BMI, biomarker level) against multiple postoperative complications
- **Treatment effects**: Test one treatment against multiple efficacy and safety endpoints simultaneously
- **Biomarker studies**: Test one biomarker against multiple clinical outcomes to understand its prognostic value
- **Phenome-wide association studies (PheWAS)**: Test genetic variants or exposures against many phenotypes
- **Risk factor profiling**: Understand how one risk factor relates to a spectrum of outcomes

Handling Categorical Predictors:

When the predictor is a factor variable with multiple levels:

- Each non-reference level gets its own row for each outcome
- Reference category is determined by factor level ordering
- The Predictor column shows "Variable (Level)" format (*e.g.,* "Treatment (Drug A)", "Treatment (Drug B)")
- For binary variables with affirmative non-reference levels (Yes, 1, True, Present, Positive, +), shows just "Variable" (*e.g.,* "Diabetes" instead of "Diabetes (Yes)")
- Effect estimates compare each level to the reference

Adjusted vs. Unadjusted Results:

When `covariates` is specified, the function fits both models but only extracts predictor effects:

- `columns = "adjusted"`: Reports only covariate-adjusted effects. Column labeled "aOR/aHR," *etc.*
- `columns = "unadjusted"`: Reports only crude effects. Column labeled "OR/HR," *etc.*
- `columns = "both"`: Reports both side-by-side. Useful for identifying confounding (large change in effect) or independent effects (similar estimates)

Interaction Terms:

When `interactions` includes terms involving the predictor:

- Main effect of predictor is always reported
- Interaction effects are extracted and displayed with formatted names
- Format: `Variable (Level) × Variable (Level)` using multiplication sign notation
- Useful for testing effect modification (*e.g.,* does treatment effect differ by sex?)

Mixed-Effects Models:

For clustered or hierarchical data (*e.g.,* patients within hospitals):

- Use `model_type = "glmer"` with `random = "(1|cluster)"` for random intercept models
- Nested random effects: `random = "(1|site/patient)"`
- Crossed random effects: `random = "(1|site) + (1|doctor)"`
- For survival outcomes, use `model_type = "coxme"`

Stratification and Clustering (Cox models):

For Cox proportional hazards models:

- `strata`: Creates separate baseline hazards for each stratum level. Use when hazards are non-proportional across strata but stratum effects do not need to be estimated
- `cluster`: Computes robust (sandwich) standard errors accounting for within-cluster correlation. Alternative to mixed effects when only robust SEs are needed

Filtering based on *p*-value:

The `p_threshold` parameter filters results after fitting all models:

- Only outcomes with p less than or equal to the threshold are retained in output
- For factor predictors, outcome is kept if any level is significant
- Useful for focusing on significant associations in exploratory analyses
- Default is 1 (no filtering) - recommended for confirmatory analyses

Outcome Homogeneity:

All outcomes in a single `multifit()` call should be of the same type (all binary, all continuous, or all survival). Mixing outcome types produces tables with incompatible effect measures (*e.g.,* odds ratios alongside regression coefficients), which can mislead readers. The function validates outcome compatibility and issues a warning when mixed types are detected.

For analyses involving multiple outcome types, run separate `multifit()` calls for each type:

```
# Binary outcomes
binary_results <- multifit(data, outcomes = c("death", "readmission"),
                           predictor = "treatment", model_type = "glm")

# Continuous outcomes
continuous_results <- multifit(data, outcomes = c("los_days", "cost"),
                               predictor = "treatment", model_type = "lm")
```

Effect Measures by Model Type:

- **Logistic** (`model_type = "glm"`, `family = "binomial"`): Odds ratios (OR/aOR)
- **Cox** (`model_type = "coxph"`): Hazard ratios (HR/aHR)
- **Poisson** (`model_type = "glm"`, `family = "poisson"`): Rate ratios (RR/aRR)
- **Linear** (`model_type = "lm"`): Coefficient estimates
- **Mixed effects**: Same as fixed-effects counterparts

Memory and Performance:

- `parallel = TRUE` (default) uses multiple cores for faster fitting
- `keep_models = FALSE` (default) discards model objects to save memory
- For many outcomes, parallel processing provides substantial speedup
- Set `keep_models = TRUE` only when model diagnostics are required

Value

A data.table with S3 class "multifit_result" containing formatted multivariate regression results. The table structure includes:

Outcome Character. Outcome variable name or custom label

Predictor Character. For factor predictors: formatted as "Variable (Level)" showing the level being compared to reference. For binary variables where the non-reference level is an affirmative value (Yes, 1, True, Present, Positive, +), shows just "Variable". For continuous predictors: the variable name. For interactions: the formatted interaction term (*e.g.,* "Treatment (Drug A) × Sex (Male)")

n Integer. Sample size used in fitting (if show_n = TRUE). When both unadjusted and adjusted estimates are shown, this is the adjusted model's sample, which is the one behind the adjusted estimate on the same row

Events Integer. Number of events (if show_events = TRUE)

OR/HR/RR/Coefficient (95% CI) Character. Unadjusted effect estimate with CI (if columns = "unadjusted" or "both")

aOR/aHR/aRR/Adj. Coefficient (95% CI) Character. Adjusted effect estimate with CI (if columns = "adjusted" or "both")

Uni *p* / Multi *p* / *p*-value Character. Formatted *p*-value(s). Column names depend on columns setting

The returned object includes the following attributes accessible via attr():

raw_data data.table. Unformatted numeric results with separate columns for effect estimates, standard errors, confidence intervals, and *p*-values. Suitable for custom analysis or visualization

models list (if keep_models = TRUE). Named list of fitted model objects, with outcome names as list names. Each element contains $unadjusted and/or $adjusted models depending on settings

predictor Character. The predictor variable name

outcomes Character vector. The outcome variable names

covariates Character vector or NULL. The covariate variable names

interactions Character vector or NULL. The interaction terms

random Character or NULL. The random effects formula

strata Character or NULL. The stratification variable

cluster Character or NULL. The clustering variable

model_type Character. The regression model type used

columns Character. Which columns were displayed

conf_level Numeric. The confidence level of the intervals, which multiforest() uses to draw the same intervals

analysis_type Character. "multi_outcome" to identify analysis type

significant Character vector. Names of outcomes with $p < 0.05$ for the predictor (uses adjusted *p*-values when available)

analysis_counts List. How much of the supplied data reached the models, with elements `n_supplied`, `n_analyzed`, `n_missing_outcome` and `n_missing_predictor`. One model is fitted per outcome, so `n_analyzed` holds one value per outcome and the reported sample may be a range. Event counts are not included: each outcome has its own event total, so a single reported figure could not describe them all. The `print()` method always reports the analyzed sample

See Also

`uniscreen` for screening multiple predictors against one outcome, `multiforest` for creating forest plots from multifit results, `fit` for single-outcome regression with full coefficient output, `fullfit` for complete univariable-to-multivariable workflow, `forestsave` for saving the forest plot, `tablesave` for exporting tables

Other regression functions: `compfit()`, `fit()`, `fullfit()`, `print.compfit_result()`, `print.fit_result()`, `print.fullfit_result()`, `print.multifit_result()`, `print.uniscreen_result()`, `uniscreen()`

Examples

```
# Load example data
data(clintrial)
data(clintrial_labels)

# Example 1: Basic multivariate analysis (unadjusted)
result1 <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    labels = clintrial_labels,
    parallel = FALSE
)
print(result1)



# Example 2: Adjusted analysis with covariates
result2 <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(result2)

# Example 3: Compare unadjusted and adjusted results
result3 <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    columns = "both",
```

```
    labels = clintrial_labels,
    parallel = FALSE
)
print(result3)

# Example 4: Continuous predictor across outcomes
result4 <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "age",
    covariates = c("sex", "treatment", "stage"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(result4)

# Example 5: Cox regression for survival outcomes
library(survival)
cox_result <- multifit(
    data = clintrial,
    outcomes = c("Surv(pfs_months, pfs_status)",
                 "Surv(os_months, os_status)"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    model_type = "coxph",
    labels = clintrial_labels,
    parallel = FALSE
)
print(cox_result)

# Example 6: Cox with stratification by site
cox_strat <- multifit(
    data = clintrial,
    outcomes = c("Surv(os_months, os_status)"),
    predictor = "treatment",
    covariates = c("age", "sex"),
    strata = "site",
    model_type = "coxph",
    labels = clintrial_labels,
    parallel = FALSE
)
print(cox_strat)

# Example 7: Cox with clustered standard errors
cox_cluster <- multifit(
    data = clintrial,
    outcomes = c("Surv(os_months, os_status)"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    cluster = "site",
    model_type = "coxph",
    labels = clintrial_labels,
    parallel = FALSE
```

```
)
print(cox_cluster)

# Example 8: Interaction between predictor and covariate
result_int <- multifit(
    data = clintrial,
    outcomes = c("surgery", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    interactions = c("treatment:sex"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(result_int)

# Example 9: Linear model for continuous outcomes
linear_result <- multifit(
    data = clintrial,
    outcomes = c("los_days", "biomarker_x"),
    predictor = "treatment",
    covariates = c("age", "sex"),
    model_type = "lm",
    labels = clintrial_labels,
    parallel = FALSE
)
print(linear_result)

# Example 10: Poisson regression for equidispersed count outcomes
# fu_count has variance ~= mean, appropriate for standard Poisson
poisson_result <- multifit(
    data = clintrial,
    outcomes = c("fu_count"),
    predictor = "treatment",
    covariates = c("age", "stage"),
    model_type = "glm",
    family = "poisson",
    labels = clintrial_labels,
    parallel = FALSE
)
print(poisson_result)

# Example 11: Filter to significant results only
sig_results <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "stage",
    p_threshold = 0.05,
    labels = clintrial_labels,
    parallel = FALSE
)
print(sig_results)

# Example 12: Custom outcome labels
```

```
result_labeled <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    labels = c(
        surgery = "Surgical Resection",
        pfs_status = "Disease Progression",
        os_status = "Death",
        treatment = "Treatment Group"
    ),
    parallel = FALSE
)
print(result_labeled)

# Example 13: Keep models for diagnostics
result_models <- multifit(
    data = clintrial,
    outcomes = c("surgery", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex"),
    keep_models = TRUE,
    parallel = FALSE
)

# Access stored models
models <- attr(result_models, "models")
names(models)

# Get adjusted model for surgery outcome
surgery_model <- models$surgery$adjusted
summary(surgery_model)

# Example 14: Access raw numeric data
result <- multifit(
    data = clintrial,
    outcomes = c("surgery", "os_status"),
    predictor = "age",
    parallel = FALSE
)

# Get unformatted results for custom analysis
raw_data <- attr(result, "raw_data")
print(raw_data)

# Example 15: Hide sample size and event columns
result_minimal <- multifit(
    data = clintrial,
    outcomes = c("surgery", "os_status"),
    predictor = "treatment",
    show_n = FALSE,
    show_events = FALSE,
    labels = clintrial_labels,
    parallel = FALSE
```

```
)
print(result_minimal)

# Example 16: Customize decimal places
result_digits <- multifit(
    data = clintrial,
    outcomes = c("surgery", "os_status"),
    predictor = "age",
    digits = 3,
    p_digits = 4,
    labels = clintrial_labels,
    parallel = FALSE
)
print(result_digits)

# Example 17: Force coefficient display (no exponentiation)
result_coef <- multifit(
    data = clintrial,
    outcomes = c("surgery"),
    predictor = "age",
    exponentiate = FALSE,
    labels = clintrial_labels,
    parallel = FALSE
)
print(result_coef)

# Example 18: Complete publication workflow
final_table <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage", "grade"),
    columns = "both",
    labels = clintrial_labels,
    digits = 2,
    p_digits = 3,
    parallel = FALSE
)
print(final_table)

# Example 19: Gamma regression for positive continuous outcomes
gamma_result <- multifit(
    data = clintrial,
    outcomes = c("los_days", "recovery_days"),
    predictor = "treatment",
    covariates = c("age", "surgery"),
    model_type = "glm",
    family = Gamma(link = "log"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(gamma_result)
```

```
# Example 20: Quasipoisson for overdispersed counts
quasi_result <- multifit(
    data = clintrial,
    outcomes = c("ae_count"),
    predictor = "treatment",
    covariates = c("age", "diabetes"),
    model_type = "glm",
    family = "quasipoisson",
    labels = clintrial_labels,
    parallel = FALSE
)
print(quasi_result)

# Example 21: Generalized linear mixed effects (GLMER)
# Test treatment across outcomes with site clustering
if (requireNamespace("lme4", quietly = TRUE)) {
    glmer_result <- suppressWarnings(multifit(
        data = clintrial,
        outcomes = c("surgery", "pfs_status", "os_status"),
        predictor = "treatment",
        covariates = c("age", "sex"),
        random = "(1|site)",
        model_type = "glmer",
        family = "binomial",
        labels = clintrial_labels,
        parallel = FALSE
    ))
    print(glmer_result)
}

# Example 22: Cox mixed effects with random site effects
if (requireNamespace("coxme", quietly = TRUE)) {
    coxme_result <- multifit(
        data = clintrial,
        outcomes = c("Surv(pfs_months, pfs_status)",
                     "Surv(os_months, os_status)"),
        predictor = "treatment",
        covariates = c("age", "sex", "stage"),
        random = "(1|site)",
        model_type = "coxme",
        labels = clintrial_labels,
        parallel = FALSE
    )
    print(coxme_result)
}

# Example 23: Multiple interactions across outcomes
multi_int <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    interactions = c("treatment:stage", "treatment:sex"),
```

```
    labels = clintrial_labels,
    parallel = FALSE
)
print(multi_int)
```

`multiforest` *Create Forest Plot for Multivariate Regression*

Description

Generates a publication-ready forest plot from a `multifit()` output object. The plot displays effect estimates (OR, HR, RR, or coefficients) with confidence intervals across multiple outcomes, organized by outcome with the predictor levels shown for each.

Usage

```
multiforest(
  x,
  title = "Multivariate Analysis",
  effect_label = NULL,
  column = "adjusted",
  digits = 2,
  p_digits = 3,
  conf_level = 0.95,
  font_size = 1,
  annot_size = 3.88,
  header_size = 5.82,
  title_size = 23.28,
  plot_width = NULL,
  plot_height = NULL,
  table_width = 0.6,
  show_n = TRUE,
  show_events = NULL,
  show_predictor = NULL,
  covariates_footer = TRUE,
  indent_predictor = FALSE,
  bold_variables = TRUE,
  center_padding = 4,
  zebra_stripes = TRUE,
  color = NULL,
  null_line = NULL,
  log_scale = NULL,
  labels = NULL,
  units = "in",
  number_format = NULL
)
```

Arguments

x	Multifit result object (data.table with class attributes from `multifit()`).
title	Character string specifying the plot title. Default is `"Multivariate Analysis"`. Use descriptive titles for publication.
effect_label	Character string for the effect measure label on the forest plot axis. Default is `NULL`, which auto-detects based on model type (*e.g.,* "Odds Ratio", "Hazard Ratio", "Rate Ratio", "Estimate").
column	Character string specifying which results to plot when `multifit()` was called with `columns = "both"`. Options are `"adjusted"` (default) or `"unadjusted"`. Ignored when the `multifit` result contains only one column type.
digits	Integer specifying the number of decimal places for effect estimates and confidence intervals. Default is 2.
p_digits	Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.
conf_level	Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals). The CI percentage is automatically displayed in column headers (*e.g.,* "90% CI" when `conf_level = 0.90`). At the level of the `multifit()` result, the intervals of its table are drawn; at another level, they are computed from the coefficients and their standard errors, with the *t* distribution on the residual degrees of freedom for linear models whose table holds exact *t* intervals, and as Wald intervals otherwise.
font_size	Numeric multiplier controlling the base font size for all text elements. Default is 1.0.
annot_size	Numeric value controlling the relative font size for data annotations. Default is 3.88.
header_size	Numeric value controlling the relative font size for column headers. Default is 5.82.
title_size	Numeric value controlling the relative font size for the main plot title. Default is 23.28.
plot_width	Numeric value specifying the intended output width in specified `units`. Used for optimizing layout. Default is `NULL` (automatic). Recommended: 10-16 inches.
plot_height	Numeric value specifying the intended output height in specified `units`. Default is `NULL` (automatic based on number of rows).
table_width	Numeric value between 0 and 1 specifying the proportion of total plot width allocated to the data table. Default is 0.6 (60% table, 40% forest plot).
show_n	Logical. If `TRUE`, includes a column showing sample sizes. Counts describe the observations used in fitting rather than every row supplied. Default is `TRUE`.
show_events	Logical. If `TRUE`, includes a column showing the number of events for each row. Default is `TRUE` for binomial and survival models, `FALSE` for linear models.
show_predictor	Logical. If `TRUE`, includes the "Predictor" column showing which level of a factor predictor is being compared. If `FALSE`, omits the column (useful when predictor info is in the caption). Default is `NULL`, which uses the `include_predictor` setting from `multifit()` if available, otherwise `TRUE`.

`covariates_footer`
: Logical. If `TRUE` (default), displays a footer listing the covariates used in adjusted models. Covariate names are formatted using the `labels` parameter if provided. Only shown when displaying adjusted results.

`indent_predictor`
: Logical. If `TRUE`, indents predictor levels under outcome names, creating a hierarchical display. If `FALSE` (default), shows outcome and predictor level in separate columns.

`bold_variables`
: Logical. If `TRUE`, variable names are displayed in bold. If `FALSE` (default), variable names are displayed in plain text.

`center_padding`
: Numeric value specifying horizontal spacing between table and forest plot. Default is 4.

`zebra_stripes`
: Logical. If `TRUE`, applies alternating gray background shading to different outcomes for improved readability. Default is `TRUE`.

`color`
: Character string specifying the color for point estimates in the forest plot. Default is `NULL`, which auto-selects based on effect type: purple (`"#8A61D8"`) for hazard ratios (Cox), teal (`"#4BA6B6"`) for odds ratios (logistic), blue (`"#3F87EE"`) for rate/risk ratios (Poisson, Gamma, *etc.*), and green (`"#5A8F5A"`) for coefficients (linear models). Use hex codes or R color names for custom colors.

`null_line`
: Numeric value for the reference line position. Default is `NULL`, which uses 1 for ratio measures (OR, HR, RR) and 0 for coefficients.

`log_scale`
: Logical. If `TRUE`, uses log scale for the x-axis. Default is `NULL`, which auto-detects (`TRUE` for OR/HR/RR, `FALSE` for coefficients).

`labels`
: Named character vector providing custom display labels for outcomes and variables. Applied to outcome names in the plot. Default is `NULL`, in which case the labels attached by `multifit()` are used. Original names are used where neither is available.

`units`
: Character string specifying units for plot dimensions: `"in"` (inches), `"cm"`, or `"mm"`. Default is `"in"`.

`number_format`
: Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

  - `"us"` - Comma thousands, period decimal: `1,234.56` [default]
  - `"eu"` - Period thousands, comma decimal: `1.234,56`
  - `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
  - `"none"` - No thousands separator: `1234.56`

  Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

  When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

  ```
  options(summata.number_format = "eu")
  ```

Details

Plot Layout:

The forest plot is organized with outcomes as grouping headers and predictor levels (or interaction terms) as rows within each outcome. This provides a clear visual comparison of how a single predictor affects multiple outcomes.

1. **Title**: Centered at top
2. **Data Table** (left): Contains:
   - Outcome column (or grouped headers)
   - Predictor/Group column
   - n: Sample sizes (optional)
   - Events: Event counts (optional, for applicable models)
   - Effect (95% CI); *p*-value
3. **Forest Plot** (right):
   - Point estimates (squares)
   - 95% confidence intervals
   - Reference line at null value (1 or 0)
   - Log scale for ratio measures

Data Source:

The function extracts effect estimates directly from the multifit output object's `raw_data` attribute, which contains the numeric values needed for plotting. This approach is efficient and ensures consistency with the formatted table output.

Value

A `ggplot` object containing the complete forest plot. The plot can be:

- Displayed directly: `print(plot)`
- Saved to file: `forestsave(plot, "forest.pdf")`
- Further customized with **ggplot2** functions

The returned object includes an attribute `"rec_dims"` accessible via `attr(plot, "rec_dims")`, which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

units Character. The units the dimensions are expressed in, matching the `units` argument

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if `plot_width` or `plot_height` are not specified. `forestsave()` reads all three and requires no further handling.

The returned object also includes an attribute `"table_data"` accessible via `attr(plot, "table_data")`, a data.table holding the values drawn in the plot, in the order of the model terms. Sample sizes and event counts describe the observations used in fitting rather than every row supplied.

See Also

autoforest for automatic model detection, multifit for multi-outcome regression analysis, glmforest for single GLM forest plots, coxforest for single Cox model forest plots, lmforest for single linear model forest plots, uniforest for univariable screening forest plots, forestsave for saving with recommended dimensions

Other visualization functions: autoforest(), coxforest(), forestsave(), glmforest(), lmforest(), recdims(), uniforest()

Examples

```
data(clintrial)
data(clintrial_labels)
library(survival)

# Create example multifit result
result <- multifit(
    data = clintrial,
    outcomes = c("surgery", "pfs_status", "os_status"),
    predictor = "treatment",
    covariates = c("age", "sex", "stage"),
    parallel = FALSE
)

# Example 1: Basic multivariate forest plot
p <- multiforest(result)



old_width <- options(width = 180)

# Example 2: With custom title and labels
plot2 <- multiforest(
    result,
    title = "Treatment Effects Across Clinical Outcomes",
    labels = clintrial_labels
)

# Example 3: Customize appearance
plot3 <- multiforest(
    result,
    color = "#E74C3C",
    zebra_stripes = TRUE,
    labels = clintrial_labels
)

# Example 4: Save with recommended dimensions
forestsave(plot3, file.path(tempdir(), "multioutcome_forest.pdf"))

options(old_width)
```

recdims *Recommended Figure Dimensions*

Description

Returns the figure dimensions recommended for a forest plot, computed from the plot's own content by the function that produced it.

Usage

```
recdims(x, units = NULL)
```

Arguments

x	A forest plot produced by `lmforest()`, `glmforest()`, `coxforest()`, `uniforest()`, `multiforest()`, or `autoforest()`.
units	Character string giving the units the dimensions are returned in: `"in"` (inches), `"cm"`, or `"mm"`. Defaults to the units the plot was created in, so that the dimensions pass through unconverted unless a change is requested.

Details

Forest plots are sized by their content rather than by the page: a plot with more variables, longer labels, or more factor levels requires a larger canvas. Each plotting function computes a suitable size and attaches it to the object it returns, and `recdims()` reports it.

The values returned are those `forestsave()` applies, and are reported exactly as computed rather than rounded, so that a figure written with `forestsave()` and one written with `ggplot2::ggsave()` at these dimensions are identical. Explicit use is therefore needed only where the dimensions themselves are wanted, as when sizing a **knitr** chunk or arranging several plots on one page.

Where `units` differs from the units the plot was created in, the dimensions are converted. A plot produced before the units were recorded carries none, and is treated as having been created in inches.

Value

A named numeric vector with elements `width` and `height`, expressed in `units`. The units are recorded on the result as a `"units"` attribute, so that a value carried between functions remains self-describing.

See Also

`forestsave` for saving at these dimensions, `autoforest` for producing the plot

Other visualization functions: `autoforest()`, `coxforest()`, `forestsave()`, `glmforest()`, `lmforest()`, `multiforest()`, `uniforest()`

Examples

```
data(clintrial)
data(clintrial_labels)

model <- stats::glm(readmission_30d ~ age + sex + stage,
                    data = clintrial, family = stats::binomial)

p <- glmforest(model, data = clintrial, labels = clintrial_labels)

# Example 1: Dimensions in the units the plot was created in
recdims(p)

# Example 2: Journals commonly specify figure widths in millimeters
recdims(p, units = "mm")

# Example 3: Sizing an output canvas from the individual elements
dims <- recdims(p)
dims[["width"]]
dims[["height"]]
attr(dims, "units")
```

---

survtable *Create Publication-Ready Survival Summary Tables*

Description

Generates comprehensive survival summary tables with survival probabilities at specified time points, median survival times, and optional group comparisons with statistical testing. Designed for creating survival summaries commonly used in clinical and epidemiological research publications.

Usage

```
survtable(
  data,
  outcome,
  by = NULL,
  times = NULL,
  probs = 0.5,
  stats = c("survival", "ci"),
  type = "survival",
  conf_level = 0.95,
  conf_type = "log",
  digits = 0,
  time_digits = 1,
  p_digits = 3,
  percent = TRUE,
```

```
  test = TRUE,
  test_type = "logrank",
  total = TRUE,
  total_label = "Total",
  time_unit = NULL,
  time_label = NULL,
  median_label = NULL,
  labels = NULL,
  by_label = NULL,
  na_rm = TRUE,
  number_format = NULL,
  ...
)
```

Arguments

`data`	Data frame or data.table containing the survival dataset. Automatically converted to a data.table for efficient processing.
`outcome`	Character string or character vector specifying one or more survival outcomes using `Surv()` syntax (*e.g.,* `"Surv(os_months, os_status)"`). When multiple outcomes are provided, results are stacked into a single table with outcome labels as row headers.
`by`	Character string specifying the column name of the stratifying variable for group comparisons (*e.g.,* treatment arm, risk group). When `NULL` (default), produces overall survival summaries only. Groups appear in the order of the levels of a factor, including empty levels, and otherwise in order of first appearance in the data, as in `desctable()`.
`times`	Numeric vector of time points at which to estimate survival probabilities. For example, `c(12, 24, 36)` for 1-, 2-, and 3-year survival when time is measured in months. Default is `NULL`.
`probs`	Numeric vector of survival probabilities for which to estimate corresponding survival times (quantiles). Values must be between 0 and 1. For example, `c(0.5)` returns median survival time, `c(0.25, 0.5, 0.75)` returns quartiles. Default is `0.5` (median only).
`stats`	Character vector specifying which statistics to display: • `"survival"` - Survival probability at specified times • `"ci"` - Confidence interval for survival probability • `"n_risk"` - Number at risk at each time point • `"n_event"` - Cumulative number of events by each time point Default is `c("survival", "ci")`.
`type`	Character string specifying the type of probability to report: • `"survival"` - Survival probability S(t) [default] • `"risk"` - Cumulative incidence/risk 1 - S(t) • `"cumhaz"` - Cumulative hazard -log(S(t))

`conf_level` Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals).

`conf_type` Character string specifying the confidence interval type for survival estimates:

- `"log"` - Log transformation (default, recommended)
- `"log-log"` - Log-log transformation
- `"plain"` - Linear/identity (can produce CIs outside [0, 1])
- `"logit"` - Logit transformation
- `"arcsin"` - Arcsin square root transformation

`digits` Integer specifying the number of decimal places for survival probabilities (as percentages). Default is 0 (whole percentages).

`time_digits` Integer specifying the number of decimal places for survival time estimates (median, quantiles). Default is 1.

`p_digits` Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

`percent` Logical. If `TRUE` (default), displays survival probabilities as percentages (*e.g.,* `"85%"`). If `FALSE`, displays as proportions (*e.g.,* `"0.85"`).

`test` Logical. If `TRUE` (default), performs a survival curve comparison test and adds a *p*-value column. Requires `by` to be specified.

`test_type` Character string specifying the statistical test for comparing survival curves:

- `"logrank"` - Log-rank test (default)
- `"wilcoxon"` - Gehan-Breslow generalized Wilcoxon test
- `"tarone"` - Tarone-Ware test
- `"petopeto"` - Peto-Peto test

See Details for the weights that define each test.

`total` Logical or character string controlling the total/overall column:

- `TRUE` or `"first"` - Include total column first [default]
- `"last"` - Include total column last (before *p*-value)
- `FALSE` - Exclude total column

`total_label` Character string for the total/overall row label. Default is `"Total"`.

`time_unit` Character string specifying the time unit for display in column headers and labels (*e.g.,* `"months"`, `"days"`, `"years"`). When specified, time column headers become "{time} {time_unit}" (*e.g.,* "12 months"). Default is `NULL` (no unit shown).

`time_label` Character string template for time column headers when `times` is specified. Use `"\{time\}"` as placeholder for the time value and `"\{unit\}"` for the time unit. Default is `"\{time\} \{unit\}"` when `time_unit` is specified, otherwise just `"\{time\}"`.

`median_label` Character string for the median survival row label. Default is `NULL`, which auto-constructs from `conf_level` (*e.g.,* `"Median (95% CI)"` for `conf_level = 0.95`).

`labels` Named character vector or list providing custom display labels. For stratified analyses, names should match levels of the by variable. For multiple outcomes, names should match the `Surv()` expressions. Default is `NULL`.

`by_label` Character string providing a custom label for the stratifying variable (used in output attributes and headers). Default is `NULL` (uses variable name).

`na_rm` Logical. If `TRUE` (default), observations with missing values in time, status, or the stratifying variable are excluded.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

`...` Additional arguments passed to `survfit`.

Details

Survival Probability Estimation:

Survival probabilities are estimated using the Kaplan-Meier method via `survfit`. At each specified time point, the function reports the estimated probability of surviving beyond that time.

Confidence Intervals:

The default `"log"` transformation for confidence intervals is recommended as it ensures intervals remain within [0, 1] and has good statistical properties. The `"log-log"` transformation is also commonly used and may perform better in the tails.

Statistical Testing:

The log-rank test (default) tests the null hypothesis that survival curves are identical across groups. It compares the observed and expected numbers of events in each group at every event time; the alternative tests weight these comparisons, with n_j subjects at risk and d_j events at event time t_j:

- Log-rank: Equal weights (best for proportional hazards)
- Gehan-Breslow (`"wilcoxon"`): Weights n_j, the number at risk, which emphasize early differences
- Tarone-Ware: Weights $\sqrt{n_j}$, between the log-rank and Gehan-Breslow weights
- Peto-Peto: Weights $\tilde{S}(t_j) = \prod_{t_i \leq t_j}(1 - d_i/(n_i + 1))$, an estimate of the pooled survival function, which also emphasize early differences but, unlike the number at risk, do not depend on the censoring distribution

The log-rank test is computed by `survival::survdiff()`. The weighted tests use the same statistic and variance with the weights above, the definitions of the SAS LIFETEST and Stata `sts test` procedures.

Formatting:

All numeric output respects the `number_format` parameter. Separators within confidence intervals adapt automatically to avoid ambiguity:

- Survival probabilities: `"85% (80%-89%)"` (US) or `"85% (80%-89%)"` (EU, en-dash separator)
- Median survival: `"24.5 (21.2-28.9)"` (US) or `"24,5 (21,2-28,9)"` (EU)
- Counts $\geq$ 1000: `"1,234"` (US) or `"1.234"` (EU)
- *p*-values: `"< 0.001"` (US) or `"< 0,001"` (EU)

Value

A data.table with S3 class `"survtable"` containing formatted survival statistics. The table structure depends on parameters:

When `times` is specified (survival at time points):

Variable/Group Row identifier – stratifying variable levels

Time columns Survival statistics at each requested time point

***p*-value** Test *p*-value (if `test = TRUE` and by specified)

When only `probs` is specified (survival quantiles):

Variable/Group Row identifier – stratifying variable levels

Quantile columns Time to reach each survival probability

***p*-value** Test *p*-value (if `test = TRUE` and by specified)

All numeric output (probabilities, times, counts, *p*-values) respects the `number_format` setting for locale-appropriate formatting.

The returned object includes the following attributes:

raw_data Data.table with unformatted numeric values

survfit_objects List of survfit objects for each stratum

by_variable The stratifying variable name

times The time points requested

probs The probability quantiles requested

test_result Full test result object (if test performed)

See Also

`desctable` for baseline characteristics tables, `fit` for regression analysis, `tablesave` for export in any supported format, `table2pdf` for PDF export, `table2docx` for Word export, `survfit` for underlying survival estimation, `survdiff` for survival curve comparison tests

Other descriptive functions: `desctable()`, `print.survtable()`

Examples

```
# Load example data
data(clintrial)

# Example 1: Survival at specific time points by treatment
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24, 36),
    time_unit = "months",
    labels = clintrial_labels
)



# Example 2: Median survival only
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = NULL,
    probs = 0.5,
    labels = clintrial_labels
)

# Example 3: Multiple quantiles (quartiles)
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "stage",
    times = NULL,
    probs = c(0.25, 0.5, 0.75),
    labels = clintrial_labels
)

# Example 4: Both time points and median
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    probs = 0.5,
    time_unit = "months",
    labels = clintrial_labels
)

# Example 5: Cumulative incidence (1 - survival)
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
```

```
    times = c(12, 24),
    type = "risk",
    labels = clintrial_labels
)

# Example 6: Include number at risk
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    stats = c("survival", "ci", "n_risk"),
    labels = clintrial_labels
)

# Example 7: Overall survival without stratification
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    times = c(12, 24, 36, 48),
    labels = clintrial_labels
)

# Example 8: Without total row
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    total = FALSE,
    labels = clintrial_labels
)

# Example 9: Custom labels
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    labels = c("Drug A" = "Treatment A", "Drug B" = "Treatment B"),
    time_unit = "months"
)

# Example 10: Different confidence interval type
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    conf_type = "log-log",
    labels = clintrial_labels
)
```

```
# Example 11: Gehan-Breslow generalized Wilcoxon test instead of log-rank
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    test_type = "wilcoxon",
    labels = clintrial_labels
)

# Example 12: Access raw data for custom analysis
result <- survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24)
)
raw <- attr(result, "raw_data")
print(raw)

# Example 13: Access survfit objects for plotting
fits <- attr(result, "survfit_objects")
plot(fits$overall)  # Plot overall survival curve

# Example 14: Multiple survival outcomes stacked
survtable(
    data = clintrial,
    outcome = c("Surv(pfs_months, pfs_status)", "Surv(os_months, os_status)"),
    by = "treatment",
    times = c(12, 24),
    probs = 0.5,
    time_unit = "months",
    total = FALSE,
    labels = c(
        "Surv(pfs_months, pfs_status)" = "Progression-Free Survival",
        "Surv(os_months, os_status)" = "Overall Survival"
    )
)

# Example 15: European number formatting
survtable(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    by = "treatment",
    times = c(12, 24),
    number_format = "eu",
    labels = clintrial_labels
)
```

table2docx *Export Table to Microsoft Word Format (DOCX)*

Description

Converts a data frame, data.table, or matrix to a fully editable Microsoft Word document (.docx) using the **flextable** and **officer** packages. Creates publication-ready tables with extensive formatting options including typography, alignment, colors, and page layout. Tables can be further edited in Microsoft Word after creation.

Usage

```
table2docx(
  table,
  file,
  caption = NULL,
  font_size = 8,
  font_family = "Arial",
  format_headers = TRUE,
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  zebra_stripes = FALSE,
  dark_header = FALSE,
  paper = "letter",
  orientation = "portrait",
  width = NULL,
  align = NULL,
  return_ft = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

table	Data frame, data.table, or matrix to export. Can be output from desctable(), survtable(), fit(), uniscreen(), fullfit(), compfit(), multifit(), or any tabular data.
file	Character string specifying the output DOCX filename. Must have .docx extension. Example: "results.docx", "Table1.docx".
caption	Character string. Optional caption displayed above the table in the Word document. Default is NULL.

`font_size` Numeric. Base font size in points for table content. Default is 8. Typical range: 8-12 points. Headers use slightly larger size.

`font_family` Character string. Font family name for the table. Must be a font installed on the system. Default is `"Arial"`. Common options: `"Times New Roman"`, `"Calibri"`, `"Helvetica"`.

`format_headers` Logical. If `TRUE`, formats column headers by italicizing statistical notation ("*n*", "*p*"), converting underscores to spaces, and improving readability. Default is `TRUE`.

`bold_significant` Logical. If `TRUE`, applies bold formatting to *p*-values below the significance threshold. Makes significant results stand out. Default is `TRUE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. Default is `FALSE`.

`p_threshold` Numeric. Threshold for bold *p*-value formatting. Only used when `bold_significant = TRUE`. Default is 0.05.

`indent_groups` Logical. If `TRUE`, indents factor levels under their parent variable using horizontal spacing, creating hierarchical display. Useful for categorical variables in regression tables. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses table by showing only essential rows (single row for continuous, non-reference for binary). Automatically sets `indent_groups = TRUE`. Significantly reduces table height. Default is `FALSE`.

`condense_quantitative` Logical. If `TRUE`, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from `desctable()`. Automatically sets `indent_groups = TRUE`. Unlike `condense_table`, this does not collapse binary categorical variables. Default is `FALSE`.

`zebra_stripes` Logical. If `TRUE`, applies alternating row shading to different variables (not individual rows) for visual grouping. Default is `FALSE`.

`dark_header` Logical. If `TRUE`, creates a dark background with light text for the header row, providing strong visual contrast. Default is `FALSE`.

`paper` Character string specifying paper size:

- `"letter"` - US Letter (8.5" × 11") [default]
- `"a4"` - A4 (210 mm × 297 mm)
- `"legal"` - US Legal (8.5" × 14")

`orientation` Character string specifying page orientation:

- `"portrait"` - Vertical [default]
- `"landscape"` - Horizontal (for wide tables)

`width` Numeric. Table width in inches. If `NULL` (default), automatically fits to content and page width. Specify to control exactly.

`align` Character vector specifying column alignment for each column. Options: `"left"`, `"center"`, or `"right"`. If `NULL` (default), automatically determines based on content (text left, numbers right). Example: `c("left", "left", "center", "right", "right")`.

return_ft	Logical. If TRUE, returns the **flextable** object directly for further customization. If FALSE (default), returns invisibly with **flextable** object as attribute. See Details for usage. Default is FALSE.
quiet	Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is FALSE.
...	Additional arguments passed to read_docx for document initialization.

Details

Package Requirements:

This function requires:

- **flextable** - For creating formatted tables
- **officer** - For Word document manipulation

Install if needed:

```
install.packages(c("flextable", "officer"))
```

Output Features:

The generated Word document contains:

- Fully editable table (native Word table, not image)
- Professional typography and spacing
- Proper page setup (size, orientation, margins)
- Caption (if provided) as separate paragraph above table
- All formatting preserved but editable
- Compatible with Word 2007 and later

Further Customization:

For programmatic customization beyond the built-in options, access the flextable object:

Method 1: Via attribute (default)

```
result <- table2docx(table, "output.docx")
ft <- attr(result, "flextable")

# Customize flextable
ft <- flextable::bold(ft, i = 1, j = 1, part = "body")
ft <- flextable::color(ft, i = 2, j = 3, color = "red")

# Re-save if needed
doc <- officer::read_docx()
doc <- flextable::body_add_flextable(doc, ft)
print(doc, target = "customized.docx")
```

Method 2: Direct return

```
ft <- table2docx(table, "output.docx", return_ft = TRUE)

# Customize immediately
ft <- flextable::bg(ft, bg = "yellow", part = "header")
ft <- flextable::autofit(ft)

# Save to new document
doc <- officer::read_docx()
doc <- flextable::body_add_flextable(doc, ft)
print(doc, target = "custom.docx")
```

Page Layout:

The function automatically sets up the Word document with:

- Specified paper size and orientation
- Standard margins (1 inch by default)
- Continuous section (no page breaks before table)
- Left-aligned table placement

For landscape orientation:

- Automatically swaps page width and height
- Applies landscape property to section
- Useful for wide tables with many columns

Table Width Management:

Width behavior:

- `width = NULL` - Auto-fits to content and page width
- `width = 6` - Exactly 6 inches wide
- Width distributed evenly across columns by default
- Can adjust individual column widths in Word after creation

For very wide tables:

1. Use `orientation = "landscape"`
2. Use `paper = "legal"` for extra width
3. Reduce `font_size`
4. Use `condense_table = TRUE`
5. Consider breaking across multiple tables

Typography:

The function applies professional typography:

- Column headers: Bold, slightly larger font
- Body text: Regular weight, specified font size

- Numbers: Right-aligned for easy comparison
- Text: Left-aligned for readability
- Consistent spacing: Adequate padding in cells

Font family must be installed on the system where Word opens the document. Common cross-platform choices:

- Arial - Sans-serif, highly readable
- Times New Roman - Serif, traditional
- Calibri - Microsoft default, modern
- Helvetica - Sans-serif, professional

Zebra Striping:

When `zebra_stripes = TRUE`:

- Alternating variables receive light gray background
- All rows of same variable share same shading
- Improves visual grouping
- Particularly useful for tables with many factor variables
- Color can be changed in Word after creation

Dark Header:

When `dark_header = TRUE`:

- Header row: Dark gray/black background
- Header text: White for high contrast
- Modern, professional appearance
- Draws attention to column names

Integration with R Markdown/Quarto:

For R Markdown/Quarto Word output:

```
# Create flextable for inline display
ft <- table2docx(results, "temp.docx", return_ft = TRUE)

# Display in R Markdown chunk
ft  # Renders in Word output
```

Or use `flextable` directly in chunks:

```
flextable::flextable(results)
```

Value

Behavior depends on `return_ft`:

`return_ft = FALSE` Invisibly returns a list with components:

- `file` - Path to created file
- `caption` - Caption text (if provided)

The **flextable** object is accessible via `attr(result, "flextable")`

`return_ft = TRUE` Directly returns the **flextable** object for immediate further customization

In both cases, creates a `.docx` file at the specified location.

See Also

`tablesave` for saving in the format given by the file extension, `table2pptx` for PowerPoint slides, `table2pdf` for PDF output, `table2html` for HTML tables, `table2rtf` for Rich Text Format, `table2tex` for LaTeX output, `flextable` for the underlying table object, `read_docx` for Word document manipulation

Other export functions: `table2html()`, `table2pdf()`, `table2pptx()`, `table2rtf()`, `table2tex()`, `tablesave()`

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example table
results <- fit(
   data = clintrial,
   outcome = "os_status",
   predictors = c("age", "sex", "treatment", "stage"),
   labels = clintrial_labels
)

# Example 1: Basic Word export
table2docx(results, file.path(tempdir(), "results.docx"))


old_width <- options(width = 180)
# Example 2: With caption
table2docx(results, file.path(tempdir(), "captioned.docx"),
        caption = "Table 1: Multivariable Logistic Regression Results")

# Example 3: Landscape orientation for wide tables
table2docx(results, file.path(tempdir(), "wide.docx"),
        orientation = "landscape")

# Example 4: Custom font and size
table2docx(results, file.path(tempdir(), "custom_font.docx"),
        font_family = "Times New Roman",
        font_size = 11)
```

```
# Example 5: Hierarchical display
table2docx(results, file.path(tempdir(), "indented.docx"),
        indent_groups = TRUE)

# Example 6: Condensed table
table2docx(results, file.path(tempdir(), "condensed.docx"),
        condense_table = TRUE)

# Example 7: With zebra stripes
table2docx(results, file.path(tempdir(), "striped.docx"),
        zebra_stripes = TRUE)

# Example 8: Dark header style
table2docx(results, file.path(tempdir(), "dark.docx"),
        dark_header = TRUE)

# Example 9: A4 paper for international journals
table2docx(results, file.path(tempdir(), "a4.docx"),
        paper = "a4")

# Example 10: Get flextable for customization
result <- table2docx(results, file.path(tempdir(), "base.docx"))
ft <- attr(result, "flextable")

# Customize the flextable
ft <- flextable::bold(ft, i = 1, part = "body")
ft <- flextable::color(ft, j = "p-value", color = "blue")

# Example 11: Direct flextable return
ft <- table2docx(results, file.path(tempdir(), "direct.docx"), return_ft = TRUE)
ft <- flextable::bg(ft, bg = "yellow", part = "header")

# Example 12: Publication-ready table
table2docx(results, file.path(tempdir(), "publication.docx"),
        caption = "Table 2: Adjusted Odds Ratios for Mortality",
        font_family = "Times New Roman",
        font_size = 10,
        indent_groups = TRUE,
        zebra_stripes = FALSE,
        bold_significant = TRUE)

# Example 13: Custom column alignment
table2docx(results, file.path(tempdir(), "aligned.docx"),
        align = c("left", "left", "center", "right", "right"))

# Example 14: Disable significance bolding
table2docx(results, file.path(tempdir(), "no_bold.docx"),
        bold_significant = FALSE)

# Example 15: Stricter significance threshold
table2docx(results, file.path(tempdir(), "strict.docx"),
        bold_significant = TRUE,
```

```
        p_threshold = 0.01)

options(old_width)
```

---

table2html *Export Table to HTML Format*

Description

Converts a data frame, data.table, or matrix to HTML format with optional CSS styling for web display, HTML documents, or embedding in web applications. Generates clean, standards-compliant HTML with professional styling options including responsive design support, color schemes, and interactive features. Requires **xtable** for export.

Usage

```
table2html(
  table,
  file,
  caption = NULL,
  format_headers = TRUE,
  variable_padding = FALSE,
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  zebra_stripes = FALSE,
  stripe_color = "#EEEEEE",
  dark_header = FALSE,
  include_css = TRUE,
  quiet = FALSE,
  ...
)
```

Arguments

table	Data frame, data.table, or matrix to export. Can be output from desctable(), survtable(), fit(), uniscreen(), fullfit(), compfit(), multifit(), or any tabular data.
file	Character string specifying the output HTML filename. Must have .html or .htm extension. Example: "results.html".
caption	Character string. Optional caption displayed below the table. Supports basic HTML formatting. Default is NULL.

`format_headers` Logical. If `TRUE`, formats column headers by converting underscores to spaces and applying proper casing. Wraps statistical notation ("*n*", "*p*") in `<i>` tags. Default is `TRUE`.

`variable_padding` Logical. If `TRUE`, adds vertical spacing around variable groups for improved readability. Particularly useful for multi-row factor variables. Default is `FALSE`.

`bold_significant` Logical. If `TRUE`, wraps significant *p*-values in `<b>` tags for bold display. Makes important results stand out visually. Default is `TRUE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. Default is `FALSE`.

`p_threshold` Numeric. Threshold for bold *p*-value formatting. Only used when `bold_significant = TRUE`. Default is 0.05.

`indent_groups` Logical. If `TRUE`, indents grouped rows using non-breaking spaces (` `) for hierarchical display. Useful for factor variables in regression output. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses table by showing only essential rows. Automatically sets `indent_groups = TRUE`. Default is `FALSE`.

`condense_quantitative` Logical. If `TRUE`, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from `desctable()`. Automatically sets `indent_groups = TRUE`. Unlike `condense_table`, this does not collapse binary categorical variables. Default is `FALSE`.

`zebra_stripes` Logical. If `TRUE`, applies alternating background shading to different variables (not individual rows) for visual grouping. Default is `FALSE`.

`stripe_color` Character string. HTML color specification for zebra stripes. Can use hex codes (`"#EEEEEE"`), RGB (`"rgb(238,238,238)"`), or color names (`"lightgray"`). Default is `"#EEEEEE"`.

`dark_header` Logical. If `TRUE`, creates black background with white text for the header row. Provides strong visual contrast. Default is `FALSE`.

`include_css` Logical. If `TRUE`, includes embedded CSS styling in the output file for standalone HTML. Set to `FALSE` when embedding in existing HTML with its own stylesheet. Default is `TRUE`.

`quiet` Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is `FALSE`.

`...` Additional arguments passed to `xtable`.

Details

Output Format:

The function generates standards-compliant HTML5 markup with:

- Semantic `<table>` structure
- Proper `<thead>` and `<tbody>` sections
- Accessible header cells (`<th>`)

- Clean, readable markup
- Optional embedded CSS styling

Standalone vs. Embedded:

Standalone HTML (`include_css = TRUE`*):*

- Can be opened directly in web browsers
- Includes all necessary styling
- Self-contained, portable
- Suitable for sharing via email or web hosting

Embedded HTML (`include_css = FALSE`*):*

- For inclusion in existing HTML documents
- No CSS included (use parent document's styles)
- Smaller file size
- Integrates with web frameworks (Shiny, R Markdown, Quarto)

CSS Styling:

When `include_css = TRUE`, the function applies professional styling:

- **Table:** Border-collapse, sans-serif font (Arial), 20px margin
- **Cells:** 8px vertical × 12px horizontal padding, left-aligned text
- **Borders:** 1px solid `#DDD` (light gray)
- **Headers:** Bold text, light gray background (`#F2F2F2`)
- **Numeric columns:** Center-aligned (auto-detected)
- **Caption:** Bold, 1.1em font, positioned below table

With `dark_header = TRUE`*:*

- Header background: Black (`#000000`)
- Header text: White (`#FFFFFF`)
- Creates high contrast, modern appearance

With `zebra_stripes = TRUE`*:*

- Alternating variable groups receive background color
- Default color: `#EEEEEE` (light gray)
- Applied via CSS class `.zebra-stripe`
- Groups entire variable (all factor levels together)

Hierarchical Display:

The `indent_groups` option creates visual hierarchy using HTML non-breaking spaces:

```
<td><b>Treatment</b></td>  <!-- Variable name -->
<td>    Control</td>  <!-- Indented level -->
<td>    Active</td>   <!-- Indented level -->
```

Integration with R Markdown/Quarto:

For R Markdown or Quarto documents:

```
# Generate HTML fragment (no CSS)
table2html(results, "table.html", include_css = FALSE)
```

Then include in the document chunk with `results='asis'`:

```
cat(readLines("table.html"), sep = "\n")
```

Or directly render without file:

```
# For inline display
htmltools::HTML(
  capture.output(
    print(xtable::xtable(results), type = "html")
  )
)
```

Integration with Shiny:

For Shiny applications:

```
# In server function
output$results_table <- renderUI({
  table2html(results_data(), "temp.html", include_css = FALSE)
  HTML(readLines("temp.html"))
})

# Or use directly with DT package for interactive tables
output$interactive_table <- DT::renderDT({
  results_data()
})
```

Accessibility:

The generated HTML follows accessibility best practices:

- Semantic table structure
- Proper header cells (<th>) with scope attributes
- Clear visual hierarchy
- Adequate color contrast (when using default styles)
- Screen reader friendly markup

Value

Invisibly returns the file path, matching `tablesave()` and `forestsave()`. Called for its side effect of creating an HTML file at the specified location, which can be opened in web browsers or embedded in HTML documents.

See Also

tablesave for saving in the format given by the file extension, table2pdf for PDF output, table2tex for LaTeX output, table2docx for Word documents, table2pptx for PowerPoint, table2rtf for Rich Text Format, fit for regression tables, desctable for descriptive tables

Other export functions: table2docx(), table2pdf(), table2pptx(), table2rtf(), table2tex(), tablesave()

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example table
results <- fit(
     data = clintrial,
     outcome = "os_status",
     predictors = c("age", "sex", "treatment", "stage"),
     labels = clintrial_labels
)

# Example 1: Basic HTML export (standalone)
if (requireNamespace("xtable", quietly = TRUE)) {
  table2html(results, file.path(tempdir(), "results.html"))
}


# Example 2: With caption
table2html(results, file.path(tempdir(), "captioned.html"),
          caption = "Table 1: Multivariable Logistic Regression Results")

# Example 3: For embedding (no CSS)
table2html(results, file.path(tempdir(), "embed.html"),
          include_css = FALSE)
# Include in the HTML document

# Example 4: Hierarchical display
table2html(results, file.path(tempdir(), "indented.html"),
          indent_groups = TRUE)

# Example 5: Condensed table
table2html(results, file.path(tempdir(), "condensed.html"),
          condense_table = TRUE)

# Example 6: With zebra stripes
table2html(results, file.path(tempdir(), "striped.html"),
          zebra_stripes = TRUE,
          stripe_color = "#F0F0F0")

# Example 7: Dark header style
table2html(results, file.path(tempdir(), "dark.html"),
          dark_header = TRUE)
```

```
# Example 8: Combination styling
table2html(results, file.path(tempdir(), "styled.html"),
          zebra_stripes = TRUE,
          dark_header = TRUE,
          bold_significant = TRUE)

# Example 9: Custom stripe color
table2html(results, file.path(tempdir(), "blue_stripes.html"),
          zebra_stripes = TRUE,
          stripe_color = "#E3F2FD")  # Light blue

# Example 10: Disable significance bolding
table2html(results, file.path(tempdir(), "no_bold.html"),
          bold_significant = FALSE)

# Example 11: Stricter significance threshold
table2html(results, file.path(tempdir(), "strict.html"),
          bold_significant = TRUE,
          p_threshold = 0.01)

# Example 12: No header formatting
table2html(results, file.path(tempdir(), "raw_headers.html"),
          format_headers = FALSE)

# Example 13: Descriptive statistics table
desc_table <- desctable(clintrial, by = "treatment",
     variables = c("age", "sex", "bmi"), labels = clintrial_labels)

table2html(desc_table, file.path(tempdir(), "baseline.html"),
          caption = "Table 1: Baseline Characteristics by Treatment Group")

# Example 14: For R Markdown (no CSS, for inline display)
table2html(results, file.path(tempdir(), "rmd_table.html"),
          include_css = FALSE,
          indent_groups = TRUE)

# Then in R Markdown, use a chunk with results='asis' to display inline:
cat(readLines(file.path(tempdir(), "rmd_table.html")), sep = "\n")

# Example 15: Email-friendly version
table2html(results, file.path(tempdir(), "email.html"),
          include_css = TRUE,  # Self-contained
          zebra_stripes = TRUE,
          caption = "Regression Results - See Attached")
# Can be directly included in HTML emails

# Example 16: Publication-ready web version
table2html(results, file.path(tempdir(), "publication.html"),
          caption = "Table 2: Multivariable Analysis of Risk Factors",
          indent_groups = TRUE,
          zebra_stripes = FALSE,  # Clean look
          bold_significant = TRUE,
```

```
          dark_header = FALSE)

# Example 17: Modern dark theme
table2html(results, file.path(tempdir(), "dark_theme.html"),
          dark_header = TRUE,
          stripe_color = "#2A2A2A",  # Dark gray stripes
          zebra_stripes = TRUE)

# Example 18: Minimal styling for custom CSS
table2html(results, file.path(tempdir(), "minimal.html"),
          include_css = FALSE,
          format_headers = FALSE,
          bold_significant = FALSE)
# Apply personal CSS classes and styling

# Example 19: Model comparison table
models <- list(
     base = c("age", "sex"),
     full = c("age", "sex", "treatment", "stage")
)

# Information criteria assume a common sample, so the candidate
# predictors are restricted to complete cases before comparison
comparison_data <- na.omit(clintrial[, c("os_status", "age", "sex", "treatment", "stage")])

comparison <- compfit(
     data = comparison_data,
     outcome = "os_status",
     model_list = models
)

table2html(comparison, file.path(tempdir(), "comparison.html"),
          caption = "Model Comparison Statistics")
```

`table2pdf` *Export Table to PDF Format*

Description

Converts a data frame, data.table, or matrix to a professionally formatted PDF document using LaTeX as an intermediate format. Provides extensive control over page layout, typography, and formatting for publication-ready output. Particularly well-suited for tables from regression analyses, descriptive statistics, and model comparisons. Requires **xtable** for export.

Usage

```
table2pdf(
  table,
```

```
  file,
  orientation = "portrait",
  paper = "letter",
  margins = NULL,
  fit_to_page = TRUE,
  font_size = 8,
  caption = NULL,
  caption_size = NULL,
  format_headers = TRUE,
  variable_padding = FALSE,
  cell_padding = "normal",
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  align = NULL,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  zebra_stripes = FALSE,
  stripe_color = "gray!20",
  dark_header = FALSE,
  show_logs = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

table	Data frame, data.table, or matrix to export. Can be output from desctable(), survtable(), fit(), uniscreen(), fullfit(), compfit(), or any tabular data structure.
file	Character string specifying the output PDF filename. Must have .pdf extension. Example: "results.pdf", "Table1.pdf".
orientation	Character string specifying page orientation: • "portrait" - Vertical orientation [default] • "landscape" - Horizontal orientation (recommended for wide tables)
paper	Character string specifying paper size: • "letter" - US Letter (8.5" x 11") [default] • "a4" - A4 (210 mm x 297 mm) • "auto" - Auto-size to content (no margins, crops to fit)
margins	Numeric vector of length 4 specifying margins in inches as c(top, right, bottom, left). Default is c(1, 1, 1, 1). Ignored when paper = "auto".
fit_to_page	Logical. If TRUE, scales table to fit within the text width (respects margins). Useful for wide tables that would otherwise overflow. Default is TRUE.
font_size	Numeric. Base font size in points. Default is 8. Smaller values accommodate more content; larger values improve readability. Typical range: 6-12 points.

`caption` Character string. Optional caption displayed below the table. Supports LaTeX formatting for multi-line captions, superscripts, italics, *etc.* See Details for formatting guidance. Default is `NULL`.

`caption_size` Numeric. Caption font size in points. If `NULL` (default), uses the base `font_size`. Set to a specific value (*e.g.,* 6, 7, 8) to control caption size independently of table font size. Useful for fitting captions on constrained page sizes. Typical range: 6-10 points.

`format_headers` Logical. If `TRUE`, applies automatic formatting to column headers: converts underscores to spaces, italicizes statistical notation ("*n*", "*p*"), and improves readability. Default is `TRUE`.

`variable_padding`
Logical. If `TRUE`, adds vertical spacing between different variables in the table, creating visual grouping. Particularly useful for regression tables with multiple predictors. Default is `FALSE`.

`cell_padding` Character string or numeric specifying vertical padding within table cells:

- `"none"` - No extra padding (most compact)
- `"normal"` - Standard padding [default]
- `"relaxed"` - Increased padding
- `"loose"` - Maximum padding
- Numeric value - Custom multiplier (*e.g.,* `1.5`)

Adjusts `\arraystretch` in LaTeX.

`bold_significant`
Logical. If `TRUE`, applies bold formatting to p-values below the significance threshold, making significant results stand out visually. Default is `TRUE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. Default is `FALSE`.

`p_threshold` Numeric. Threshold for bold *p*-value formatting. Only used when `bold_significant = TRUE`. Default is 0.05.

`align` Character string or vector specifying column alignment. Options:

- `"l"` - Left aligned
- `"c"` - Center aligned
- `"r"` - Right aligned

If `NULL` (default), automatically determines alignment based on content (text left, numbers right). Can specify per-column: `c("l", "l", "r", "r")`.

`indent_groups` Logical. If `TRUE`, indents factor levels/groups under their parent variable using horizontal space, creating a hierarchical display. Useful for factor variables in regression tables. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses the table by:

- Showing only one row per continuous variable (estimate + CI)
- Showing only non-reference categories for binary factors
- Automatically setting `indent_groups = TRUE`

Significantly reduces table height. Default is `FALSE`.

`condense_quantitative`
: Logical. If `TRUE`, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from `desctable()`. Automatically sets `indent_groups = TRUE`. Unlike `condense_table`, this does not collapse binary categorical variables. Default is `FALSE`.

`zebra_stripes`
: Logical. If `TRUE`, applies alternating gray background shading to different variables (not individual rows) for improved visual grouping and readability. Default is `FALSE`.

`stripe_color`
: Character string. LaTeX color specification for zebra stripes. Default is `"gray!20"` (20% gray). Can use other colors like `"blue!10"`, `"red!15"`. Requires `zebra_stripes = TRUE`.

`dark_header`
: Logical. If `TRUE`, creates a dark (black) background with white text for the header row. Provides strong visual contrast. Default is `FALSE`.

`show_logs`
: Logical. If `TRUE`, retains LaTeX log and auxiliary files after PDF compilation for troubleshooting. If `FALSE`, deletes these files. Default is `FALSE`.

`quiet`
: Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is `FALSE`.

`...`
: Additional arguments passed to `xtable` for advanced LaTeX table customization.

Details

LaTeX Requirements:

This function requires a working LaTeX installation. The function checks for LaTeX availability and provides installation guidance if missing.

Recommended LaTeX distributions:

- **TinyTeX** (lightweight, R-integrated): Install via `tinytex::install_tinytex()`
- **TeX Live** (comprehensive, cross-platform)
- **MiKTeX** (Windows)
- **MacTeX** (macOS)

Required LaTeX packages (auto-installed with most distributions):

- `fontenc`, `inputenc` - Character encoding
- `array`, `booktabs`, `longtable` - Table formatting
- `graphicx` - Scaling tables
- `geometry` - Page layout
- `pdflscape`, `lscape` - Landscape orientation
- `helvet` - Sans-serif fonts
- `standalone`, `varwidth` - Auto-sizing (for `paper = "auto"`)
- `float`, `caption` - Floats and captions
- `xcolor`, `colortable` - Colors (for `zebra_stripes` or `dark_header`)

Caption Formatting:

Captions support LaTeX commands for rich formatting:

```
# Multi-line caption with line breaks
caption = "Table 1: Multivariable Analysis\\
          OR = odds ratio; CI = confidence interval"

# With superscripts (using LaTeX syntax)
caption = "Table 1: Results\\
          Adjusted for age and sex\\
          p-values from Wald tests"

# With special characters (must escape percent signs)
caption = "Results for income (in thousands)"
```

Auto-Sizing (`paper = "auto"`):

When `paper = "auto"`, the function attempts to create a minimal PDF sized exactly to the table content:

1. Using the `standalone` LaTeX class (cleanest output)
2. Fallback to `pdfcrop` utility if standalone unavailable
3. Fallback to minimal margins if neither available

Table Width Management:

For wide tables that don't fit on the page:

1. Use `orientation = "landscape"`
2. Use `fit_to_page = TRUE` (default) to auto-scale
3. Reduce `font_size` (*e.g.,* 7 or 6)
4. Consider `paper = "auto"` for maximum flexibility

Troubleshooting:

If PDF compilation fails:

1. Check that LaTeX is installed: Run `Sys.which("pdflatex")`
2. Set `show_logs = TRUE` and examine the .log file
3. Common issues:
   - Missing LaTeX packages: Install via package manager
   - Special characters in text: Escape properly
   - Very wide tables: Use landscape or reduce font size
   - Caption formatting: Check LaTeX syntax

Value

Invisibly returns the file path, matching `tablesave()` and `forestsave()`. Called for its side effect of creating a PDF file at the specified location. If compilation fails, check the `.log` file (if `show_logs = TRUE`) for error details.

See Also

tablesave for saving in the format given by the file extension, table2tex for LaTeX source files, table2html for HTML output, table2docx for Microsoft Word, table2pptx for PowerPoint, table2rtf for Rich Text Format, desctable for descriptive tables, fit for regression tables

Other export functions: table2docx(), table2html(), table2pptx(), table2rtf(), table2tex(), tablesave()

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example table
results <- fit(
    data = clintrial,
    outcome = "readmission_30d",
    predictors = c("age", "sex", "treatment", "stage"),
    labels = clintrial_labels
)

# Test that LaTeX can compile (needed for all PDF examples)
has_latex <- local({
  if (!nzchar(Sys.which("pdflatex"))) return(FALSE)
  test_tex <- file.path(tempdir(), "summata_latex_test.tex")
  writeLines(c("\\documentclass{article}", "\\usepackage{booktabs}",
               "\\begin{document}", "test", "\\end{document}"), test_tex)
  tryCatch(
    system2("pdflatex", c("-interaction=nonstopmode",
            paste0("-output-directory=", tempdir()), test_tex),
            stdout = FALSE, stderr = FALSE),
    error = function(e) 1L) == 0L
})

# Example 1: Basic PDF export
if(has_latex){
  table2pdf(results, file.path(tempdir(), "basic_results.pdf"))
}



# Example 2: Landscape orientation for wide tables
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "wide_results.pdf"),
            orientation = "landscape")
}

# Example 3: With caption
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "captioned.pdf"),
            caption = "Table 1: Multivariable logistic regression results")
}
```

```
# Example 4: Multi-line caption with formatting
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "formatted_caption.pdf"),
           caption = "Table 1: Risk Factors for Mortality\\\\
                     aOR = adjusted odds ratio; CI = confidence interval")
}

# Example 5: Auto-sized PDF (no fixed page dimensions)
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "autosize.pdf"),
           paper = "auto")
}

# Example 6: A4 paper with custom margins
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "a4_custom.pdf"),
           paper = "a4",
           margins = c(0.75, 0.75, 0.75, 0.75))
}

# Example 7: Larger font for readability
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "large_font.pdf"),
           font_size = 11)
}

# Example 8: Indented hierarchical display
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "indented.pdf"),
           indent_groups = TRUE)
}

# Example 9: Condensed table (reduced height)
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "condensed.pdf"),
           condense_table = TRUE)
}

# Example 10: With zebra stripes
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "striped.pdf"),
           zebra_stripes = TRUE,
           stripe_color = "gray!15")
}

# Example 11: Dark header style
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "dark_header.pdf"),
           dark_header = TRUE)
}

# Example 12: Combination of formatting options
```

```
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "publication_ready.pdf"),
           orientation = "portrait",
           paper = "letter",
           font_size = 9,
           caption = "Table 2: Multivariable Analysis\\\\
                     Model adjusted for age, sex, and clinical factors",
           indent_groups = TRUE,
           zebra_stripes = TRUE,
           bold_significant = TRUE,
           p_threshold = 0.05)
}

# Example 13: Adjust cell padding
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "relaxed_padding.pdf"),
           cell_padding = "relaxed")  # More spacious
}

# Example 14: No scaling (natural table width)
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "no_scale.pdf"),
           fit_to_page = FALSE,
           font_size = 10)
}

# Example 15: Hide significance bolding
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "no_bold.pdf"),
           bold_significant = FALSE)
}

# Example 16: Custom column alignment
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "custom_align.pdf"),
           align = c("c", "c", "c", "c", "c", "c", "c"))
}

# Example 17: Descriptive statistics table
if (has_latex) {
  desc_table <- desctable(clintrial, by = "treatment",
       variables = c("age", "sex", "bmi", "stage"), labels = clintrial_labels)

  table2pdf(desc_table, file.path(tempdir(), "descriptive.pdf"),
           caption = "Table 1: Baseline Characteristics by Treatment Group",
           orientation = "landscape")
}

# Example 18: Model comparison table
if (has_latex) {
  models <- list(
       base = c("age", "sex"),
       full = c("age", "sex", "bmi", "treatment")
```

```
  )

  # Information criteria assume a common sample, so the candidate
  # predictors are restricted to complete cases before comparison
 comparison_data <- na.omit(clintrial[, c("os_status", "age", "sex", "bmi", "treatment")])

  comparison <- compfit(
       data = comparison_data,
       outcome = "os_status",
       model_list = models,
       verbose = FALSE
  )

  table2pdf(comparison, file.path(tempdir(), "model_comparison.pdf"),
            caption = "Table 3: Model Comparison Statistics")
}

# Example 19: Very wide table with aggressive fitting
if (has_latex) {
  wide_model <- fit(
       data = clintrial,
       outcome = "os_status",
       predictors = c("age", "sex", "race", "bmi", "smoking",
                      "hypertension", "diabetes", "treatment", "stage")
  )

  table2pdf(wide_model, file.path(tempdir(), "very_wide.pdf"),
            orientation = "landscape",
            font_size = 7,
            fit_to_page = TRUE,
            condense_table = TRUE)
}

# Example 20: With caption size control
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "caption_size.pdf"),
            font_size = 8,
            caption_size = 6,
            caption = "Table 4: Results with Compact Caption\\\\
                      Smaller caption fits better on constrained pages")
}

# Example 21: Troubleshooting - keep logs
if (has_latex) {
  table2pdf(results, file.path(tempdir(), "debug.pdf"),
            show_logs = TRUE)
  # If it fails, check debug.log for error messages
}
```

table2pptx *Export Table to Microsoft PowerPoint Format (PPTX)*

Description

Converts a data frame, data.table, or matrix to a Microsoft PowerPoint slide (`.pptx`) with a formatted table using the **flextable** and **officer** packages. Creates presentation-ready slides with extensive control over table formatting, positioning, and layout. Tables can be further edited in PowerPoint after creation. Ideal for creating data-driven presentations and conference talks.

Usage

```
table2pptx(
  table,
  file,
  caption = NULL,
  font_size = 10,
  font_family = "Arial",
  format_headers = TRUE,
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  zebra_stripes = FALSE,
  dark_header = FALSE,
  width = NULL,
  align = NULL,
  template = NULL,
  layout = "Title and Content",
  master = "Office Theme",
  left = 0.5,
  top = 1.5,
  return_ft = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

`table`	Data frame, data.table, or matrix to export. Can be output from `desctable()`, `survtable()`, `fit()`, `uniscreen()`, `fullfit()`, `compfit()`, `multifit()`, or any tabular data.
`file`	Character string specifying the output PPTX filename. Must have `.pptx` extension. Example: `"results.pptx"`, `"Slide1.pptx"`.

caption — Character string. Optional title displayed in the slide's title placeholder or as text box above the table. Default is NULL.

font_size — Numeric. Base font size in points for table content. Default is 10. Typical range for presentations: 10-14 points. Larger than print documents for visibility at distance.

font_family — Character string. Font family name for the table. Must be installed on the system. Default is "Arial". Common presentation fonts: "Calibri", "Helvetica", "Arial".

format_headers — Logical. If TRUE, formats column headers by italicizing statistical notation ("*n*", "*p*") and improving readability. Default is TRUE.

bold_significant — Logical. If TRUE, applies bold formatting to *p*-values below the significance threshold. Makes important results stand out in presentations. Default is TRUE.

bold_variables — Logical. If TRUE, variable names are displayed in bold. Default is FALSE.

p_threshold — Numeric. Threshold for bold *p*-value formatting. Only used when bold_significant = TRUE. Default is 0.05.

indent_groups — Logical. If TRUE, indents factor levels under their parent variable, creating hierarchical display. Useful for categorical variables. Default is FALSE.

condense_table — Logical. If TRUE, condenses table to essential rows only. Automatically sets indent_groups = TRUE. Crucial for fitting content on slides. Default is FALSE.

condense_quantitative — Logical. If TRUE, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from desctable(). Automatically sets indent_groups = TRUE. Unlike condense_table, this does not collapse binary categorical variables. Default is FALSE.

zebra_stripes — Logical. If TRUE, applies alternating row shading to different variables for visual grouping. Improves readability during presentations. Default is FALSE.

dark_header — Logical. If TRUE, creates dark background with light text for header row. Provides strong contrast visible from distance. Default is FALSE.

width — Numeric. Table width in inches. If NULL (default), auto-fits to slide width (approximately 9 inches for standard 10-inch slide with margins). Specify for exact control.

align — Character vector specifying column alignment. Options: "left", "center", or "right". If NULL (default), automatically determines based on content.

template — Character string. Path to custom PPTX template file. If NULL (default), uses officer's default blank template. Use to match corporate branding or conference themes.

layout — Character string. Name of slide layout to use from template. Default is "Title and Content". Common layouts:

- "Title and Content" - Standard layout [default]
- "Blank" - Empty slide for maximum control
- "Title Only" - Title area only

- `"Two Content"` - Title with two content areas

`master`	Character string. Name of slide master to use. Default is `"Office Theme"`. Varies by template. Check template for available masters.
`left`	Numeric. Horizontal position from left edge of slide in inches. Default is 0.5. Standard slide is 10 inches wide.
`top`	Numeric. Vertical position from top edge of slide in inches. Default is 1.5 (leaves room for title). Standard slide is 7.5 inches tall. Adjust based on table size and layout.
`return_ft`	Logical. If `TRUE`, returns the **flextable** object directly for further customization. If `FALSE` (default), returns invisibly with **flextable** object as attribute. See Details for usage. Default is `FALSE`.
`quiet`	Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is `FALSE`.
`...`	Additional arguments passed to `read_pptx`.

Details

Package Requirements:

Requires:

- **flextable** - Table creation and formatting
- **officer** - PowerPoint manipulation

Install: `install.packages(c("flextable", "officer"))`

Slide Dimensions:

Standard PowerPoint slide:

- Width: 10 inches (25.4 cm)
- Height: 7.5 inches (19.05 cm)
- Aspect ratio: 4:3 (standard) or 16:9 (widescreen)

Safe content area (with margins):

- Width: ~9 inches
- Height: ~6 inches (accounting for title)

Positioning:

The `left` and `top` parameters control table placement:

- (0, 0) = Top-left corner of slide
- Default (0.5, 1.5) = Standard position with title room
- Center: `left = (10 - table_width) / 2`

When caption is provided:

- Attempts to use title placeholder (if layout supports)
- Falls back to text box above table

- Automatically adjusts table position downward

Slide Layouts:

Different layouts serve different purposes:

Title and Content (default):

- Has title and content placeholders
- Caption goes in title area
- Table in content area
- Most common for data slides

Blank:

- No predefined areas
- Maximum flexibility
- Use absolute positioning (`left`, `top`)
- Good for custom layouts

Title-Only:

- Title area only
- Large space for table
- Good for data-heavy slides

Custom Templates:

Use organizational or conference templates:

```
table2pptx(table, "branded.pptx",
         template = "company_template.pptx",
         layout = "Content Layout",  # Name from template
         master = "Company Theme")   # Name from template
```

To find layout and master names in template:

```
pres <- officer::read_pptx("template.pptx")
officer::layout_summary(pres)
```

Multiple Slides:

Creating presentations with multiple tables:

```
# Each call creates new presentation - combine after
table2pptx(table1, "slide1.pptx", caption = "Results Part 1")
table2pptx(table2, "slide2.pptx", caption = "Results Part 2")

# Then manually combine in PowerPoint, or:
# Use officer to create multi-slide presentation
pres <- officer::read_pptx()
```

```
# Add first table
ft1 <- table2pptx(table1, "temp1.pptx", return_ft = TRUE)
pres <- officer::add_slide(pres)
pres <- officer::ph_with(pres, ft1,
                         location = officer::ph_location(left = 0.5, top = 1.5))

# Add second table
ft2 <- table2pptx(table2, "temp2.pptx", return_ft = TRUE)
pres <- officer::add_slide(pres)
pres <- officer::ph_with(pres, ft2,
                         location = officer::ph_location(left = 0.5, top = 1.5))

print(pres, target = "combined.pptx")
```

Further Customization:

Access the `flextable` object for advanced formatting:

```
ft <- table2pptx(table, "base.pptx", return_ft = TRUE)

# Customize
ft <- flextable::color(ft, j = "p-value", color = "red")
ft <- flextable::bg(ft, i = 1, bg = "yellow")
ft <- flextable::bold(ft, i = ~ estimate > 0, j = "estimate")

# Save to new slide
pres <- officer::read_pptx()
pres <- officer::add_slide(pres)
pres <- officer::ph_with(pres, ft,
                         location = officer::ph_location(left = 0.5, top = 1.5))
print(pres, target = "custom.pptx")
```

Value

Behavior depends on `return_ft`:

`return_ft = FALSE` Invisibly returns a list with:

- `file` - Path to created file
- `caption` - Caption/title text
- `layout` - Layout name used
- `master` - Master name used
- `template` - Template path (if provided)
- `position` - List with `left` and `top` coordinates

Flextable accessible via `attr(result, "flextable")`

`return_ft = TRUE` Directly returns the **flextable** object

Always creates a `.pptx` file at the specified location.

See Also

tablesave for saving in the format given by the file extension, table2docx for Word documents, table2pdf for PDF output, table2html for HTML tables, table2rtf for Rich Text Format, table2tex for LaTeX output, flextable for table customization, read_pptx for PowerPoint manipulation

Other export functions: table2docx(), table2html(), table2pdf(), table2rtf(), table2tex(), tablesave()

Examples

```
# Create example data
data(clintrial)
data(clintrial_labels)
tbl <- desctable(clintrial, by = "treatment",
    variables = c("age", "sex"), labels = clintrial_labels)

# Basic PowerPoint export
table2pptx(tbl, file.path(tempdir(), "example.pptx"))


old_width <- options(width = 180)
# Load data
data(clintrial)
data(clintrial_labels)

# Create regression table
results <- fit(
   data = clintrial,
   outcome = "os_status",
   predictors = c("age", "sex", "treatment"),
   labels = clintrial_labels
)

# Example 1: Basic PowerPoint slide
table2pptx(results, file.path(tempdir(), "results.pptx"))

# Example 2: With title
table2pptx(results, file.path(tempdir(), "titled.pptx"),
        caption = "Multivariable Regression Results")

# Example 3: Larger font for visibility
table2pptx(results, file.path(tempdir(), "large_font.pptx"),
        font_size = 12,
        caption = "Main Findings")

# Example 4: Condensed for slide space
table2pptx(results, file.path(tempdir(), "condensed.pptx"),
        condense_table = TRUE,
        caption = "Key Results")

# Example 5: Dark header for emphasis
```

```
table2pptx(results, file.path(tempdir(), "dark.pptx"),
        dark_header = TRUE,
        caption = "Risk Factors")

# Example 6: With zebra stripes
table2pptx(results, file.path(tempdir(), "striped.pptx"),
        zebra_stripes = TRUE)

# Example 7: Blank layout with custom positioning
table2pptx(results, file.path(tempdir(), "blank.pptx"),
        layout = "Blank",
        left = 1,
        top = 1.5,
        width = 8)

# Example 8: Get flextable for customization
ft <- table2pptx(results, file.path(tempdir(), "base.pptx"), return_ft = TRUE)

# Customize the returned flextable object
ft <- flextable::color(ft, j = "p-value", color = "darkred")

# Example 9: Presentation-optimized table
table2pptx(results, file.path(tempdir(), "presentation.pptx"),
        caption = "Main Analysis Results",
        font_size = 11,
        condense_table = TRUE,
        zebra_stripes = TRUE,
        dark_header = TRUE,
        bold_significant = TRUE)

# Example 10: Descriptive statistics slide
desc <- desctable(
   data = clintrial,
   by = "treatment",
   variables = c("age", "sex", "bmi"),
   labels = clintrial_labels
)

table2pptx(desc, file.path(tempdir(), "baseline.pptx"),
        caption = "Baseline Characteristics",
        font_size = 10)

# Example 11: Conference presentation style
table2pptx(results, file.path(tempdir(), "conference.pptx"),
        caption = "Study Outcomes",
        font_family = "Calibri",
        font_size = 14,  # Large for big rooms
        dark_header = TRUE,
        condense_table = TRUE)

options(old_width)
```

table2rtf *Export Table to Rich Text Format (RTF)*

Description

Converts a data frame, data.table, or matrix to a Rich Text Format (`.rtf`) document using the **flextable** and **officer** packages. Creates widely compatible tables with extensive formatting options. RTF files can be opened and edited in Microsoft Word, LibreOffice, WordPad, and many other word processors. Particularly useful for regulatory submissions, cross-platform compatibility, and when maximum editability is required.

Usage

```
table2rtf(
  table,
  file,
  caption = NULL,
  font_size = 8,
  font_family = "Arial",
  format_headers = TRUE,
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  zebra_stripes = FALSE,
  dark_header = FALSE,
  paper = "letter",
  orientation = "portrait",
  width = NULL,
  align = NULL,
  return_ft = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

`table`	Data frame, data.table, or matrix to export. Can be output from `desctable()`, `survtable()`, `fit()`, `uniscreen()`, `fullfit()`, `compfit()`, `multifit()`, or any tabular data.
`file`	Character string specifying the output RTF filename. Must have `.rtf` extension. Example: `"results.rtf"`, `"Table1.rtf"`.
`caption`	Character string. Optional caption displayed above the table in the RTF document. Default is `NULL`.

`font_size` Numeric. Base font size in points for table content. Default is 8. Typical range: 8-12 points. Headers use slightly larger size.

`font_family` Character string. Font family name for the table. Must be a font installed on the system. Default is `"Arial"`. Common options: `"Times New Roman"`, `"Calibri"`, `"Courier New"`.

`format_headers` Logical. If `TRUE`, formats column headers by italicizing statistical notation ("*n*", "*p*"), converting underscores to spaces, and improving readability. Default is `TRUE`.

`bold_significant` Logical. If `TRUE`, applies bold formatting to *p*-values below the significance threshold. Makes significant results stand out. Default is `TRUE`.

`bold_variables` Logical. If `TRUE`, variable names are displayed in bold. Default is `FALSE`.

`p_threshold` Numeric. Threshold for bold *p*-value formatting. Only used when `bold_significant = TRUE`. Default is 0.05.

`indent_groups` Logical. If `TRUE`, indents factor levels under their parent variable using horizontal spacing, creating hierarchical display. Useful for categorical variables in regression tables. Default is `FALSE`.

`condense_table` Logical. If `TRUE`, condenses table by showing only essential rows (single row for continuous, non-reference for binary). Automatically sets `indent_groups = TRUE`. Significantly reduces table height. Default is `FALSE`.

`condense_quantitative` Logical. If `TRUE`, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from `desctable()`. Automatically sets `indent_groups = TRUE`. Unlike `condense_table`, this does not collapse binary categorical variables. Default is `FALSE`.

`zebra_stripes` Logical. If `TRUE`, applies alternating row shading to different variables (not individual rows) for visual grouping. Default is `FALSE`.

`dark_header` Logical. If `TRUE`, creates a dark background with light text for the header row, providing strong visual contrast. Default is `FALSE`.

`paper` Character string specifying paper size:

- `"letter"` - US Letter (8.5" × 11") [default]
- `"a4"` - A4 (210 mm × 297 mm)
- `"legal"` - US Legal (8.5" × 14")

`orientation` Character string specifying page orientation:

- `"portrait"` - Vertical [default]
- `"landscape"` - Horizontal (for wide tables)

`width` Numeric. Table width in inches. If `NULL` (default), automatically fits to content and page width. Specify to control exactly.

`align` Character vector specifying column alignment for each column. Options: `"left"`, `"center"`, or `"right"`. If `NULL` (default), automatically determines based on content (text left, numbers right). Example: `c("left", "left", "center", "right", "right")`.

return_ft	Logical. If TRUE, returns the **flextable** object directly for further customization. If FALSE (default), returns invisibly with flextable as attribute. See Details for usage. Default is FALSE.
quiet	Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is FALSE.
...	Additional arguments (currently unused, reserved for future extensions).

Details

Package Requirements:

This function requires:

- **flextable** - For creating formatted tables
- **officer** - For RTF document generation

Install if needed:

```
install.packages(c("flextable", "officer"))
```

RTF Format Advantages:

RTF (Rich Text Format) is a universal document format with several advantages:

- **Maximum compatibility** - Opens in virtually all word processors
- **Cross-platform** - Works on Windows, Mac, Linux without conversion
- **Fully editable** - Native text format, not embedded objects
- **Lightweight** - Smaller file sizes than DOCX
- **Regulatory compliance** - Widely accepted for submissions (FDA, EMA)
- **Long-term accessibility** - Simple text-based format
- **Version control friendly** - Text-based, works with diff tools

Applications that can open RTF files:

- Microsoft Word (Windows, Mac)
- LibreOffice Writer
- Apache OpenOffice Writer
- WordPad (Windows built-in)
- TextEdit (Mac built-in)
- Google Docs (with import)
- Pages (Mac)
- Many other word processors

Output Features:

The generated RTF document contains:

- Fully editable table (native RTF table, not image)

- Professional typography and spacing
- Proper page setup (size, orientation, margins)
- Caption (if provided) as separate paragraph above table
- All formatting preserved but editable
- Compatible with RTF 1.5 specification

Further Customization:

For programmatic customization beyond the built-in options, access the `flextable` object:

Method 1: Via attribute (default)

```
result <- table2rtf(table, "output.rtf")
ft <- attr(result, "flextable")

# Customize flextable
ft <- flextable::bold(ft, i = 1, j = 1, part = "body")
ft <- flextable::color(ft, i = 2, j = 3, color = "red")

# Re-save if needed
flextable::save_as_rtf(ft, path = "customized.rtf")
```

Method 2: Direct return

```
ft <- table2rtf(table, "output.rtf", return_ft = TRUE)

# Customize immediately
ft <- flextable::bg(ft, bg = "yellow", part = "header")
ft <- flextable::autofit(ft)

# Save to new file
flextable::save_as_rtf(ft, path = "custom.rtf")
```

Page Layout:

The function automatically sets up the RTF document with:

- Specified paper size and orientation
- Standard margins (1 inch by default)
- Table positioned at document start
- Left-aligned table placement

For landscape orientation:

- Automatically swaps page dimensions
- Applies landscape property
- Useful for wide tables with many columns

Table Width Management:

Width behavior:

- `width = NULL` - Auto-fits to content and page width
- `width = 6` - Exactly 6 inches wide
- Width distributed evenly across columns by default
- Can adjust individual column widths in word processor after creation

For very wide tables:

1. Use `orientation = "landscape"`
2. Use `paper = "legal"` for extra width
3. Reduce `font_size`
4. Use `condense_table = TRUE`
5. Consider breaking across multiple tables

Typography:

The function applies professional typography:

- Column headers: Bold, slightly larger font
- Body text: Regular weight, specified font size
- Numbers: Right-aligned for easy comparison
- Text: Left-aligned for readability
- Consistent spacing: Adequate padding in cells
- Statistical notation: Italicized appropriately

Value

Behavior depends on `return_ft`:

`return_ft = FALSE` Invisibly returns a list with components:

- `file` - Path to created file
- `caption` - Caption text (if provided)

The **flextable** object is accessible via `attr(result, "flextable")`

`return_ft = TRUE` Directly returns the **flextable** object for immediate further customization

In both cases, creates a .rtf file at the specified location.

See Also

`tablesave` for saving in the format given by the file extension, `table2docx` for Word documents, `table2pptx` for PowerPoint slides, `table2pdf` for PDF output, `table2html` for HTML tables, `table2tex` for LaTeX output, `flextable` for the underlying table object, `save_as_rtf` for direct RTF export

Other export functions: `table2docx()`, `table2html()`, `table2pdf()`, `table2pptx()`, `table2tex()`, `tablesave()`

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example table
results <- fit(
   data = clintrial,
   outcome = "os_status",
   predictors = c("age", "sex", "treatment", "stage"),
   labels = clintrial_labels
)

# Example 1: Basic RTF export
table2rtf(results, file.path(tempdir(), "results.rtf"))


old_width <- options(width = 180)
# Example 2: With caption
table2rtf(results, file.path(tempdir(), "captioned.rtf"),
       caption = "Table 1: Multivariable Logistic Regression Results")

# Example 3: Landscape orientation for wide tables
table2rtf(results, file.path(tempdir(), "wide.rtf"),
       orientation = "landscape")

# Example 4: Custom font and size
table2rtf(results, file.path(tempdir(), "custom_font.rtf"),
       font_family = "Times New Roman",
       font_size = 11)

# Example 5: Hierarchical display
table2rtf(results, file.path(tempdir(), "indented.rtf"),
       indent_groups = TRUE)

# Example 6: Condensed table
table2rtf(results, file.path(tempdir(), "condensed.rtf"),
       condense_table = TRUE)

# Example 7: With zebra stripes
table2rtf(results, file.path(tempdir(), "striped.rtf"),
       zebra_stripes = TRUE)

# Example 8: Dark header style
table2rtf(results, file.path(tempdir(), "dark.rtf"),
       dark_header = TRUE)

# Example 9: A4 paper for international submissions
table2rtf(results, file.path(tempdir(), "a4.rtf"),
       paper = "a4")

# Example 10: Get flextable for customization
result <- table2rtf(results, file.path(tempdir(), "base.rtf"))
```

```
ft <- attr(result, "flextable")

# Customize the flextable
ft <- flextable::bold(ft, i = 1, part = "body")
ft <- flextable::color(ft, j = "p-value", color = "blue")

# Re-save
flextable::save_as_rtf(ft, path = file.path(tempdir(), "customized.rtf"))

# Example 11: Direct flextable return
ft <- table2rtf(results, file.path(tempdir(), "direct.rtf"), return_ft = TRUE)
ft <- flextable::bg(ft, bg = "yellow", part = "header")

# Example 12: Regulatory submission table
table2rtf(results, file.path(tempdir(), "submission.rtf"),
       caption = "Table 2: Adjusted Odds Ratios for Mortality",
       font_family = "Times New Roman",
       font_size = 10,
       indent_groups = TRUE,
       zebra_stripes = FALSE,
       bold_significant = TRUE)

# Example 13: Custom column alignment
table2rtf(results, file.path(tempdir(), "aligned.rtf"),
       align = c("left", "left", "center", "right", "right"))

# Example 14: Disable significance bolding
table2rtf(results, file.path(tempdir(), "no_bold.rtf"),
       bold_significant = FALSE)

# Example 15: Stricter significance threshold
table2rtf(results, file.path(tempdir(), "strict.rtf"),
       bold_significant = TRUE,
       p_threshold = 0.01)

# Example 16: Descriptive statistics for baseline characteristics
desc <- desctable(clintrial, by = "treatment",
   variables = c("age", "sex", "bmi", "stage"), labels = clintrial_labels)

table2rtf(desc, file.path(tempdir(), "baseline.rtf"),
       caption = "Table 1: Baseline Patient Characteristics",
       zebra_stripes = TRUE)

# Example 17: Clinical trial efficacy table
table2rtf(results, file.path(tempdir(), "efficacy.rtf"),
       caption = "Table 3: Primary Efficacy Analysis - Intent to Treat Population",
       font_family = "Courier New",  # Monospace for alignment
       paper = "letter",
       orientation = "landscape",
       condense_table = TRUE)

options(old_width)
```

table2tex *Export Table to LaTeX Format*

Description

Converts a data frame, data.table, or matrix to LaTeX source code suitable for inclusion in LaTeX documents. Generates publication-quality table markup with extensive formatting options including booktabs styling, color schemes, and hierarchical displays. Output can be directly `\input{}` or `\include{}` into LaTeX manuscripts. Requires **xtable** for export.

Usage

```
table2tex(
  table,
  file,
  format_headers = TRUE,
  variable_padding = FALSE,
  cell_padding = "normal",
  bold_significant = TRUE,
  bold_variables = FALSE,
  p_threshold = 0.05,
  align = NULL,
  indent_groups = FALSE,
  condense_table = FALSE,
  condense_quantitative = FALSE,
  booktabs = FALSE,
  zebra_stripes = FALSE,
  stripe_color = "gray!20",
  dark_header = FALSE,
  caption = NULL,
  caption_size = NULL,
  label = NULL,
  show_logs = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

table	Data frame, data.table, or matrix to export. Can be output from `desctable()`, `survtable()`, `fit()`, `uniscreen()`, `fullfit()`, `compfit()`, `multifit()`, or any tabular data.
file	Character string specifying the output `.tex` filename. Must have `.tex` extension. Example: `"results.tex"`, `"table1.tex"`.

format_headers Logical. If TRUE, formats column headers by converting underscores to spaces, italicizing statistical notation ("*n*", "*p*"), and applying title case. Default is TRUE.

variable_padding
Logical. If TRUE, adds vertical spacing around variable groups using \addlinespace for improved readability. Default is FALSE.

cell_padding Character string or numeric. Vertical padding within cells:

- "none" - No extra padding
- "normal" - Standard padding [default]
- "relaxed" - Increased padding
- "loose" - Maximum padding
- Numeric - Custom \arraystretch value

bold_significant
Logical. If TRUE, wraps significant *p*-values in textbf commands for bold display. Default is TRUE.

bold_variables Logical. If TRUE, variable names are displayed in bold. Default is FALSE.

p_threshold Numeric. Threshold for bold *p*-value formatting. Only used when bold_significant = TRUE. Default is 0.05.

align Character string or vector specifying column alignment:

- "l" - Left
- "c" - Center
- "r" - Right
- Paragraph column with specified width (p-type)

If NULL, automatically determines based on content. Can specify per-column as vector. Default is NULL.

indent_groups Logical. If TRUE, uses hspace to indent grouped rows, creating hierarchical display. Useful for factor variables in regression tables. Default is FALSE.

condense_table Logical. If TRUE, condenses table by showing only essential rows (single row for continuous, non-reference for binary). Automatically sets indent_groups = TRUE. Default is FALSE.

condense_quantitative
Logical. If TRUE, condenses continuous and survival variables into single rows while preserving all categorical variable rows (including binary). Only applies to descriptive tables from desctable(). Automatically sets indent_groups = TRUE. Unlike condense_table, this does not collapse binary categorical variables. Default is FALSE.

booktabs Logical. If TRUE, uses booktabs package commands (toprule, midrule, bottomrule) for professional table rules. Requires booktabs package in LaTeX preamble. Default is FALSE.

zebra_stripes Logical. If TRUE, adds alternating row colors for variable groups using rowcolor command. Requires xcolor package with table option in preamble. Default is FALSE.

stripe_color Character string. LaTeX color specification for zebra stripes (*e.g.,* "gray!20", "blue!10"). Only used when zebra_stripes = TRUE. Default is "gray!20".

dark_header	Logical. If TRUE, creates white text on black background for header row. Requires xcolor package with table option. Default is FALSE.
caption	Character string. Table caption for LaTeX caption command. Supports multiline captions using double backslash. Default is NULL.
caption_size	Numeric. Caption font size in points. If NULL (default), caption will use the document's default caption size (typically slightly smaller than body text). Set to a specific value (*e.g.,* 6, 7, 8, 9) to control caption size explicitly. This generates a LaTeX comment for use when wrapping the table. Typical range: 6-10 points.
label	Character string. LaTeX label for cross-references. Example: "tab:regression". Default is NULL.
show_logs	Logical. If TRUE, displays informational messages about required LaTeX packages and formatting options applied. If FALSE, suppresses these messages. Default is FALSE.
quiet	Logical. Suppress progress and confirmation messages, such as the notice reporting the file written. Errors and warnings are unaffected. Default is FALSE.
...	Additional arguments passed to xtable.

Details

Output Format:

The function generates a standalone LaTeX tabular environment that can be:

1. Included in documents with \input command
2. Embedded in table/figure environments
3. Used in manuscript classes (article, report, *etc.*)

The output includes:

- Complete tabular environment with proper alignment
- Horizontal rules (\hline or booktabs rules)
- Column headers with optional formatting
- Data rows with automatic escaping of special characters
- Optional caption and label commands

Required LaTeX Packages:

The following are required in the LaTeX document preamble:

Always required:

```
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{array}
\usepackage{graphicx}
```

Optional (based on parameters):

```
\usepackage{booktabs}
\usepackage[table]{xcolor}
```

Booktabs Style:

When `booktabs = TRUE`, the table uses publication-quality rules:

- `\toprule` - Heavy rule at top
- `\midrule` - Medium rule below headers
- `\bottomrule` - Heavy rule at bottom
- No vertical rules (`booktabs` style)
- Better spacing around rules

This is the preferred style for most academic journals.

Color Features:

Zebra Stripes: Creates alternating background colors for visual grouping:

```
zebra_stripes = TRUE
stripe_color = "gray!20"  # 20% gray
stripe_color = "blue!10"  # 10% blue
```

Dark Header: Creates high-contrast header row:

```
dark_header = TRUE  # Black background, white text
```

Both require the xcolor package with the table option in the document preamble.

Integration with LaTeX Documents:

Basic inclusion:

```
\begin{table}[htbp]
  \centering
  \caption{Regression Results}
  \label{tab:regression}
  \input{results.tex}
\end{table}
```

With resizing:

```
\begin{table}[htbp]
  \centering
  \caption{Results}
  \resizebox{\textwidth}{!}{\input{results.tex}}
\end{table}
```

Landscape orientation:

```
\usepackage{pdflscape}
\begin{landscape}
  \begin{table}[htbp]
    \centering
    \input{wide_results.tex}
  \end{table}
\end{landscape}
```

Caption Formatting:

Captions in the `caption` parameter are written as LaTeX comments in the output file for reference. For actual LaTeX captions, wrap the table in a table environment (see examples above).

Special Characters:

The function automatically escapes LaTeX special characters in the data:

- Ampersand, percent, dollar sign, hash, underscore
- Left and right braces
- Tilde and caret (using `textasciitilde` and `textasciicircum`)

Variable names and labels should not include these characters unless intentionally using LaTeX commands.

Value

Invisibly returns the file path, matching `tablesave()` and `forestsave()`. Called for its side effect of creating a `.tex` file at the specified location containing a LaTeX tabular environment.

See Also

`tablesave` for saving in the format given by the file extension, `table2pdf` for direct PDF output, `table2html` for HTML tables, `table2docx` for Word documents, `table2pptx` for PowerPoint, `table2rtf` for Rich Text Format, `fit` for regression tables, `desctable` for descriptive tables

Other export functions: `table2docx()`, `table2html()`, `table2pdf()`, `table2pptx()`, `table2rtf()`, `tablesave()`

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example table
results <- fit(
   data = clintrial,
   outcome = "os_status",
   predictors = c("age", "sex", "treatment", "stage"),
   labels = clintrial_labels
)

# Example 1: Basic LaTeX export
if (requireNamespace("xtable", quietly = TRUE)) {
```

```
  table2tex(results, file.path(tempdir(), "basic.tex"))
}


# Example 2: With booktabs for publication
table2tex(results, file.path(tempdir(), "publication.tex"),
       booktabs = TRUE,
       caption = "Multivariable logistic regression results",
       label = "tab:regression")

# Example 3: Multi-line caption with abbreviations
table2tex(results, file.path(tempdir(), "detailed.tex"),
       booktabs = TRUE,
       caption = "Table 1: Risk Factors for Mortality\\\\
                  aOR = adjusted odds ratio; CI = confidence interval\\\\
                  Model adjusted for age, sex, treatment, and disease stage",
       label = "tab:mortality")

# Example 4: Hierarchical display with indentation
table2tex(results, file.path(tempdir(), "indented.tex"),
       indent_groups = TRUE,
       booktabs = TRUE)

# Example 5: Condensed table (reduced height)
table2tex(results, file.path(tempdir(), "condensed.tex"),
       condense_table = TRUE,
       booktabs = TRUE)

# Example 6: With zebra stripes
table2tex(results, file.path(tempdir(), "striped.tex"),
       zebra_stripes = TRUE,
       stripe_color = "gray!15",
       booktabs = TRUE)
# Remember to add \usepackage[table]{xcolor} to the LaTeX document

# Example 7: Dark header style
table2tex(results, file.path(tempdir(), "dark_header.tex"),
       dark_header = TRUE,
       booktabs = TRUE)
# Requires \usepackage[table]{xcolor}

# Example 8: Custom cell padding
table2tex(results, file.path(tempdir(), "relaxed.tex"),
       cell_padding = "relaxed",
       booktabs = TRUE)

# Example 9: Custom column alignment (auto-detected by default)
table2tex(results, file.path(tempdir(), "custom_align.tex"),
       align = c("c", "c", "c", "c", "c", "c", "c"))

# Example 10: No header formatting (keep original names)
table2tex(results, file.path(tempdir(), "raw_headers.tex"),
       format_headers = FALSE)
```

```
# Example 11: Disable significance bolding
table2tex(results, file.path(tempdir(), "no_bold.tex"),
       bold_significant = FALSE,
       booktabs = TRUE)

# Example 12: Stricter significance threshold
table2tex(results, file.path(tempdir(), "strict_sig.tex"),
       bold_significant = TRUE,
       p_threshold = 0.01,  # Bold only if p < 0.01
       booktabs = TRUE)

# Example 13: With caption size control
table2tex(results, file.path(tempdir(), "caption_size.tex"),
       caption_size = 6,
       caption = "Table 1 - Results with Compact Caption\\\\
                  Smaller caption fits better on constrained pages")

# Example 14: Complete publication-ready table
table2tex(results, file.path(tempdir(), "final_table1.tex"),
       booktabs = TRUE,
       caption = "Table 1: Multivariable Analysis of Mortality Risk Factors",
       label = "tab:main_results",
       indent_groups = TRUE,
       zebra_stripes = FALSE,  # Many journals prefer no stripes
       bold_significant = TRUE,
       cell_padding = "normal")

# Example 15: Descriptive statistics table
desc_table <- desctable(clintrial, by = "treatment",
   variables = c("age", "sex", "bmi"), labels = clintrial_labels)

table2tex(desc_table, file.path(tempdir(), "table1_descriptive.tex"),
       booktabs = TRUE,
       caption = "Table 1: Baseline Characteristics",
       label = "tab:baseline")

# Example 16: Model comparison table
models <- list(
   base = c("age", "sex"),
   full = c("age", "sex", "treatment", "stage")
)

# Information criteria assume a common sample, so the candidate
# predictors are restricted to complete cases before comparison
comparison_data <- na.omit(clintrial[, c("os_status", "age", "sex", "treatment", "stage")])

comparison <- compfit(
   data = comparison_data,
   outcome = "os_status",
   model_list = models
)
```

```
table2tex(comparison, file.path(tempdir(), "model_comparison.tex"),
       booktabs = TRUE,
       caption = "Model Comparison Statistics",
       label = "tab:models")
```

---

tablesave *Save a Table to File*

Description

Writes a table to file in the format indicated by the file extension, dispatching to the appropriate specialized export function. Provides a unified interface for table export across all supported formats, and is the counterpart to `forestsave()` for forest plots.

Usage

```
tablesave(table, file, quiet = FALSE, ...)

autotable(table, file, quiet = FALSE, ...)
```

Arguments

`table`	A table produced by `desctable()`, `survtable()`, `fit()`, `uniscreen()`, `fullfit()`, `compfit()`, or `multifit()`. Any data frame, data.table, or matrix is accepted.
`file`	Character string specifying the output filename. The file extension determines the export format: • `.csv` - Comma-separated values (uses `data.table::fwrite()`) • `.tsv` - Tab-separated values (uses `data.table::fwrite()`) • `.pdf` - PDF via LaTeX (uses `table2pdf()`) • `.docx` - Microsoft Word (uses `table2docx()`) • `.html` or `.htm` - HTML (uses `table2html()`) • `.pptx` - Microsoft PowerPoint (uses `table2pptx()`) • `.tex` - LaTeX source (uses `table2tex()`) • `.rtf` - Rich Text Format (uses `table2rtf()`)
`quiet`	Logical. Suppress progress and confirmation messages. The setting is forwarded to the format-specific export function, so the LaTeX compilation notices from `table2pdf()` are suppressed with it. Default is `FALSE`.
`...`	Additional arguments passed to the format-specific function. See the documentation for individual functions for available parameters: **PDF** `table2pdf()` - `orientation`, `paper`, `margins`, `fit_to_page`, *etc.* **DOCX** `table2docx()` - `font_size`, `font_family`, `caption`, *etc.* **HTML** `table2html()` - `format_headers`, `zebra_stripes`, *etc.* **PPTX** `table2pptx()` - `font_size`, `font_family`, `caption`, *etc.*

TEX `table2tex()` - `caption`, `format_headers`, `align`, *etc.*
RTF `table2rtf()` - `font_size`, `font_family`, `caption`, *etc.*
CSV, TSV `data.table::fwrite()` - `sep`, `quote`, `na`, *etc.*

Common parameters across formats include:

`caption` Table caption (supported by most formats)
`font_size` Base font size in points (PDF, DOCX, PPTX, RTF)
`format_headers` Format column headers (all formats)
`bold_significant` Bold significant *p*-values (all formats)
`p_threshold` Threshold for *p*-value bolding (all formats)
`indent_groups` Indent factor levels (all formats)
`condense_table` Condense to essential rows (all formats)
`zebra_stripes` Alternating background colors (most formats)

Details

This function provides a convenient wrapper around format-specific export functions, automatically routing to the appropriate function based on the file extension. All parameters are passed through to the underlying function, so the full range of format-specific options remains available.

For format-specific advanced features, individual export functions may be called directly:

- PDF exports support orientation, paper size, margins, and auto-sizing
- DOCX/PPTX/RTF support font customization and **flextable** formatting
- HTML supports CSS styling, responsive design, and custom themes
- TeX generates standalone LaTeX source with booktabs styling

Value

Invisibly returns `file`.

See Also

`table2pdf`, `table2docx`, `table2pptx`, `table2html`, `table2rtf`, `table2tex`

Other export functions: `table2docx()`, `table2html()`, `table2pdf()`, `table2pptx()`, `table2rtf()`, `table2tex()`

Examples

```
# Create example data
data(clintrial)
data(clintrial_labels)
tbl <- desctable(clintrial, by = "treatment",
    variables = c("age", "sex"), labels = clintrial_labels)

# Example 1: The format follows the file extension
if (requireNamespace("xtable", quietly = TRUE)) {
  tablesave(tbl, file.path(tempdir(), "example.html"))
}
```

```
# Example 2: Delimited output needs no additional packages
tablesave(tbl, file.path(tempdir(), "example.csv"))

# Example 3: Suppress the message reporting the file written
tablesave(tbl, file.path(tempdir(), "example.tsv"), quiet = TRUE)



# Load example data
data(clintrial)
data(clintrial_labels)

# Create a regression table
results <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "treatment"),
    labels = clintrial_labels
)

# Test that LaTeX can actually compile (needed for PDF export)
has_latex <- local({
  if (!nzchar(Sys.which("pdflatex"))) return(FALSE)
  test_tex <- file.path(tempdir(), "summata_latex_test.tex")
  writeLines(c("\\documentclass{article}",
               "\\usepackage{booktabs}",
               "\\begin{document}", "test",
               "\\end{document}"), test_tex)
  result <- tryCatch(
    system2("pdflatex", c("-interaction=nonstopmode",
            paste0("-output-directory=", tempdir()), test_tex),
            stdout = FALSE, stderr = FALSE),
    error = function(e) 1L)
  result == 0L
})

# Word, PowerPoint, and RTF export require flextable and officer
has_office <- requireNamespace("flextable", quietly = TRUE) &&
  requireNamespace("officer", quietly = TRUE)

# Example 4: The format is taken from the file extension
tablesave(results, file.path(tempdir(), "results.html"))  # Creates HTML file
tablesave(results, file.path(tempdir(), "results.tex"))   # Creates LaTeX source
if (has_office) {
  tablesave(results, file.path(tempdir(), "results.docx"))  # Creates Word document
  tablesave(results, file.path(tempdir(), "results.pptx"))  # Creates PowerPoint slide
  tablesave(results, file.path(tempdir(), "results.rtf"))   # Creates RTF document
}
if (has_latex) {
  tablesave(results, file.path(tempdir(), "results.pdf")) # Creates PDF
}
```

```
# Example 5: Format-specific parameters are passed through
if (has_latex) {
  tablesave(results, file.path(tempdir(), "results.pdf"),
             orientation = "landscape",
             paper = "a4",
             font_size = 10)
}

if (has_office) {
  tablesave(results, file.path(tempdir(), "results.docx"),
             caption = "Table 1: Logistic Regression Results",
             font_family = "Times New Roman",
             condense_table = TRUE)
}

tablesave(results, file.path(tempdir(), "results.html"),
           zebra_stripes = TRUE,
           dark_header = TRUE,
           bold_significant = TRUE)

# Example 6: Any summata table may be saved
desc <- desctable(clintrial,
                  by = "treatment",
                  variables = c("age", "sex", "bmi"))
if (has_latex) {
  tablesave(desc, file.path(tempdir(), "demographics.pdf"))
}

# Example 7: Model comparison table
# Information criteria assume a common sample, so the candidate
# predictors are restricted to complete cases before comparison
comparison_data <- na.omit(
    clintrial[, c("os_status", "age", "sex", "treatment", "stage")]
)
comparison <- compfit(
    data = comparison_data,
    outcome = "os_status",
    model_list = list(
        base = c("age", "sex"),
        full = c("age", "sex", "treatment", "stage")
    )
)
if (has_office) {
  tablesave(comparison, file.path(tempdir(), "model_comparison.docx"))
}
```

`uniforest` *Create Forest Plot for Univariable Screening*

Description

Generates a publication-ready forest plot from a `uniscreen()` output object. The plot displays effect estimates (OR, HR, RR, or coefficients) with confidence intervals for each predictor tested in univariable analysis against a single outcome.

Usage

```
uniforest(
  x,
  title = "Univariable Screening",
  effect_label = NULL,
  digits = 2,
  p_digits = 3,
  conf_level = 0.95,
  font_size = 1,
  annot_size = 3.88,
  header_size = 5.82,
  title_size = 23.28,
  plot_width = NULL,
  plot_height = NULL,
  table_width = 0.6,
  show_n = TRUE,
  show_events = NULL,
  indent_groups = FALSE,
  condense_table = FALSE,
  bold_variables = FALSE,
  center_padding = 4,
  zebra_stripes = TRUE,
  color = NULL,
  null_line = NULL,
  log_scale = NULL,
  labels = NULL,
  show_footer = TRUE,
  units = "in",
  number_format = NULL
)
```

Arguments

`x`	Univariable screen result object (data.table with class attributes from `uniscreen()`).
`title`	Character string specifying the plot title. Default is `"Univariable Screening"`. Use descriptive titles for publication.
`effect_label`	Character string for the effect measure label on the forest plot axis. Default is `NULL`, which auto-detects based on model type (*e.g.,* "Odds Ratio", "Hazard Ratio", "Rate Ratio", "Coefficient").
`digits`	Integer specifying the number of decimal places for effect estimates and confidence intervals. Default is 2.

p_digits — Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

conf_level — Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals). The CI percentage is automatically displayed in column headers (*e.g.,* "90% CI" when `conf_level = 0.90`). At the level of the `uniscreen()` result, the intervals of its table are drawn; at another level, they are computed from the coefficients and their standard errors, with the *t* distribution on the residual degrees of freedom for linear models whose table holds exact *t* intervals, and as Wald intervals otherwise.

font_size — Numeric multiplier controlling the base font size for all text elements. Default is 1.0.

annot_size — Numeric value controlling the relative font size for data annotations. Default is 3.88.

header_size — Numeric value controlling the relative font size for column headers. Default is 5.82.

title_size — Numeric value controlling the relative font size for the main plot title. Default is 23.28.

plot_width — Numeric value specifying the intended output width in specified `units`. Used for optimizing layout. Default is `NULL` (automatic). Recommended: 10-16 inches.

plot_height — Numeric value specifying the intended output height in specified `units`. Default is `NULL` (automatic based on number of rows).

table_width — Numeric value between 0 and 1 specifying the proportion of total plot width allocated to the data table. Default is 0.6 (60% table, 40% forest plot).

show_n — Logical. If `TRUE`, includes a column showing sample sizes. Counts describe the observations used in fitting rather than every row supplied. Default is `TRUE`.

show_events — Logical. If `TRUE`, includes a column showing the number of events for each row. Default is `NULL`, which auto-detects based on model type (`TRUE` for binomial/survival, `FALSE` for linear).

indent_groups — Logical. If `TRUE`, indents factor levels under their parent variable name, creating a hierarchical display. If `FALSE` (default), shows variable and level in separate columns.

condense_table — Logical. If `TRUE`, condenses binary categorical variables into single rows by showing only the non-reference category. Automatically sets `indent_groups = TRUE`. Useful for tables with many binary variables. Default is `FALSE`.

bold_variables — Logical. If `TRUE`, variable names are displayed in bold. If `FALSE` (default), variable names are displayed in plain text.

center_padding — Numeric value specifying horizontal spacing between table and forest plot. Default is 4.

zebra_stripes — Logical. If `TRUE`, applies alternating gray background shading to different variables for improved readability. Default is `TRUE`.

`color` Character string specifying the color for point estimates in the forest plot. Default is `NULL`, which auto-selects based on effect type: purple (`"#8A61D8"`) for hazard ratios (Cox), teal (`"#4BA6B6"`) for odds ratios (logistic), blue (`"#3F87EE"`) for rate/risk ratios (Poisson, Gamma, *etc.*), and green (`"#5A8F5A"`) for coefficients (linear models). Use hex codes or R color names for custom colors.

`null_line` Numeric value for the reference line position. Default is `NULL`, which uses 1 for ratio measures (OR, HR, RR) and 0 for coefficients.

`log_scale` Logical. If `TRUE`, uses log scale for the x-axis. Default is `NULL`, which auto-detects (`TRUE` for OR/HR/RR, `FALSE` for coefficients).

`labels` Named character vector providing custom display labels for variables. Applied to predictor names in the plot. Default is `NULL`, in which case the labels attached by the function that produced `x` are used where `x` is a **summata** result rather than a model object, so that labels supplied once are carried through the workflow. Original variable names are used where neither is available.

`show_footer` Logical. If `TRUE`, displays a footer with the outcome variable name. Default is `TRUE`.

`units` Character string specifying units for plot dimensions: `"in"` (inches), `"cm"`, or `"mm"`. Default is `"in"`.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

Details

The forest plot displays univariable (unadjusted) associations between each predictor and the outcome. This is useful for:

- Visualizing results of initial variable screening
- Identifying potential predictors for multivariable modeling
- Presenting crude associations alongside adjusted results
- Quick visual assessment of effect sizes and significance

The plot automatically handles:

- Different effect types (OR, HR, RR, coefficients) with appropriate axis scaling (log vs linear)
- Factor variables with multiple levels (grouped under variable name)
- Continuous variables (single row per predictor)
- Reference categories for categorical variables

Value

A ggplot object containing the complete forest plot. The plot can be:

- Displayed directly: `print(plot)`
- Saved to file: `forestsave(plot, "forest.pdf")`
- Further customized with **ggplot2** functions

The returned object includes an attribute `"rec_dims"` accessible via `attr(plot, "rec_dims")`, which is a list containing:

width Numeric. Recommended plot width in specified units

height Numeric. Recommended plot height in specified units

units Character. The units the dimensions are expressed in, matching the `units` argument

These recommendations are automatically calculated based on the number of variables, text sizes, and layout parameters, and are printed to console if `plot_width` or `plot_height` are not specified. `forestsave()` reads all three and requires no further handling.

The returned object also includes an attribute `"table_data"` accessible via `attr(plot, "table_data")`, a data.table holding the values drawn in the plot, in the order of the model terms. Sample sizes and event counts describe the observations used in fitting rather than every row supplied.

See Also

`autoforest` for automatic model detection, `uniscreen` for generating univariable screening results, `multiforest` for multi-outcome forest plots, `coxforest`, `glmforest`, `lmforest` for single-model forest plots, `forestsave` for saving with recommended dimensions

Other visualization functions: `autoforest()`, `coxforest()`, `forestsave()`, `glmforest()`, `lmforest()`, `multiforest()`, `recdims()`

Examples

```
data(clintrial)
data(clintrial_labels)

# Create example uniscreen result
uni_results <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "smoking", "treatment", "stage"),
    labels = clintrial_labels,
    parallel = FALSE
)

# Example 1: Basic univariable forest plot
p <- uniforest(uni_results, title = "Univariable Associations with Mortality")



old_width <- options(width = 180)
```

```
# Example 2: Survival analysis
library(survival)
surv_results <- uniscreen(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "treatment", "stage"),
    model_type = "coxph",
    labels = clintrial_labels,
    parallel = FALSE
)

# Labels attached by uniscreen() are applied automatically
p2 <- uniforest(surv_results, title = "Univariable Survival Analysis")

# Example 3: Linear regression
lm_results <- uniscreen(
    data = clintrial,
    outcome = "los_days",
    predictors = c("age", "sex", "surgery", "diabetes"),
    model_type = "lm",
    labels = clintrial_labels,
    parallel = FALSE
)

p3 <- uniforest(lm_results, title = "Predictors of Length of Stay")

# Example 4: Customize appearance
p4 <- uniforest(
    uni_results,
    title = "Crude Associations with Mortality",
    color = "#E74C3C",
    indent_groups = TRUE,
    zebra_stripes = TRUE,
    bold_variables = TRUE
)

# Example 5: Save with recommended dimensions
forestsave(p4, file.path(tempdir(), "univariable_forest.pdf"))

options(old_width)
```

---

uniscreen *Univariable Screening for Multiple Predictors*

Description

Performs comprehensive univariable (unadjusted) regression analyses by fitting separate models for each predictor against a single outcome. This function is designed for initial variable screening, hy-

pothesis generation, and understanding crude associations before multivariable modeling. Returns publication-ready formatted results with optional *p*-value filtering.

Usage

```
uniscreen(
  data,
  outcome,
  predictors,
  model_type = "glm",
  family = "binomial",
  random = NULL,
  p_threshold = 0.05,
  conf_level = 0.95,
  reference_rows = TRUE,
  show_n = TRUE,
  show_events = TRUE,
  digits = 2,
  p_digits = 3,
  labels = NULL,
  keep_models = FALSE,
  exponentiate = NULL,
  conf_method = NULL,
  parallel = TRUE,
  n_cores = NULL,
  number_format = NULL,
  verbose = NULL,
  ...
)
```

Arguments

`data`	Data frame or data.table containing the analysis dataset. The function automatically converts data frames to data.tables for efficient processing.
`outcome`	Character string specifying the outcome variable name. For survival analysis, use `Surv()` syntax from the **survival** package (*e.g.,* `"Surv(time, status)"` or `"Surv(os_months, os_status)"`).
`predictors`	Character vector of predictor variable names to screen. Each predictor is tested independently in its own univariable model. Can include continuous, categorical (factor), or binary variables. Non-syntactic names (*e.g.,* `"tumor stage"`) are quoted in the model formula and need no backticks.
`model_type`	Character string specifying the type of regression model to fit. Options include: • `"glm"` - Generalized linear model (default). Supports multiple distributions via the `family` parameter including logistic, Poisson, Gamma, Gaussian, and quasi-likelihood models. • `"lm"` - Linear regression for continuous outcomes with normally distributed errors. Equivalent to `glm` with `family = "gaussian"`.

- "coxph" - Cox proportional hazards model for time-to-event survival analysis. Requires Surv() outcome syntax.
- "clogit" - Conditional logistic regression for matched case-control studies or stratified analyses.
- "negbin" - Negative binomial regression for overdispersed count data (requires MASS package). Estimates an additional dispersion parameter compared to Poisson regression.
- "glmer" - Generalized linear mixed-effects model for hierarchical or clustered data with non-normal outcomes (requires **lme4** package and random parameter).
- "lmer" - Linear mixed-effects model for hierarchical or clustered data with continuous outcomes (requires **lme4** package and random parameter).
- "coxme" - Cox mixed-effects model for clustered survival data (requires coxme package and random parameter).

family For GLM and GLMER models, specifies the error distribution and link function. Can be a character string, a family function, or a family object. Ignored for non-GLM/GLMER models.

Binary/Binomial outcomes:

- "binomial" or binomial() - Logistic regression for binary outcomes (0/1, TRUE/FALSE). Returns odds ratios (OR). Default.
- "quasibinomial" or quasibinomial() - Logistic regression with overdispersion. Use when residual deviance » residual df.
- binomial(link = "probit") - Probit regression (normal CDF link).
- binomial(link = "cloglog") - Complementary log-log link for asymmetric binary outcomes.

Count outcomes:

- "poisson" or poisson() - Poisson regression for count data. Returns rate ratios (RR). Assumes mean = variance.
- "quasipoisson" or quasipoisson() - Poisson regression with overdispersion. Use when variance > mean.

Continuous outcomes:

- "gaussian" or gaussian() - Normal/Gaussian distribution for continuous outcomes. Equivalent to linear regression.
- gaussian(link = "log") - Log-linear model for positive continuous outcomes. Returns multiplicative effects.
- gaussian(link = "inverse") - Inverse link for specific applications.

Positive continuous outcomes:

- "Gamma" or Gamma() - Gamma distribution for positive, right-skewed continuous data (*e.g.,* costs, lengths of stay). When passed as a string, resolves to log link for interpretable multiplicative effects.
- Gamma(link = "inverse") - Gamma with inverse (canonical) link.
- Gamma(link = "identity") - Gamma with identity link for additive effects on positive outcomes.

- `"inverse.gaussian"` or `inverse.gaussian()` - Inverse Gaussian for positive, highly right-skewed data.

For negative binomial regression (overdispersed counts), use `model_type = "negbin"` instead of the `family` parameter.

See `family` for additional details and options.

`random` Character string specifying the random-effects formula for mixed-effects models (`glmer`, `lmer`, `coxme`). Use standard **lme4/coxme** syntax, *e.g.,* `"(1|site)"` for random intercepts by site, `"(1|site/patient)"` for nested random effects. Required when `model_type` is a mixed-effects model type unless random effects are included in the `predictors` vector. Alternatively, random effects can be included directly in the `predictors` vector using the same syntax (*e.g.,* `predictors = c("age", "sex", "(1|site)")`), though they will not be iterated over as predictors. Default is `NULL`.

`p_threshold` Numeric value between 0 and 1 specifying the *p*-value threshold used to count significant predictors in the printed summary. All predictors are always included in the output table. Default is 0.05.

`conf_level` Numeric confidence level for confidence intervals. Must be between 0 and 1. Default is 0.95 (95% confidence intervals).

`reference_rows` Logical. If `TRUE`, adds rows for reference categories of factor variables with baseline values (OR/HR/RR = 1, coefficient = 0). Makes tables complete and easier to interpret. Default is `TRUE`.

`show_n` Logical. If `TRUE`, includes the sample size column in the output table. Default is `TRUE`.

`show_events` Logical. If `TRUE`, includes the events column in the output table (relevant for survival and logistic regression). Default is `TRUE`.

`digits` Integer specifying the number of decimal places for effect estimates (OR, HR, RR, coefficients). Default is 2.

`p_digits` Integer specifying the number of decimal places for *p*-values. Values smaller than `10^(-p_digits)` are displayed as `"< 0.001"` (for `p_digits = 3`), `"< 0.0001"` (for `p_digits = 4`), etc. Default is 3.

`labels` Named character vector or list providing custom display labels for variables. Names should match predictor names, values are the display labels. Predictors not in `labels` use their original names. Default is `NULL`.

`keep_models` Logical. If `TRUE`, stores all fitted model objects in the output as an attribute. This allows access to models for diagnostics, predictions, or further analysis, but can consume significant memory for large datasets or many predictors. Models are accessible via `attr(result, "models")`. Default is `FALSE`.

`exponentiate` Logical. Whether to exponentiate coefficients (display OR/HR/RR instead of log odds/log hazards). Default is `NULL`, which automatically exponentiates for logistic, Poisson, and Cox models, and displays raw coefficients for linear models and other GLM families. Set to `TRUE` to force exponentiation or `FALSE` to force coefficients.

`conf_method` Character string controlling the confidence interval method. If `NULL` (default), uses `getOption("summata.conf_method", "profile")`.

- `"profile"` - Profile likelihood intervals for GLM and negative binomial models (via `MASS::confint.glm()`), exact t-distribution intervals for linear models. Falls back to Wald on profiling failure. Quasi-likelihood families always use Wald (no true likelihood).
- `"wald"` - Wald intervals (coefficient $\pm z \times$ SE) for all model types. Faster but less accurate near boundary conditions or with small subgroups.

Cox and mixed-effects models use Wald intervals regardless of this setting. Set globally with `options(summata.conf_method = "wald")` to use Wald throughout a session.

`parallel` Logical. If `TRUE` (default), fits models in parallel using multiple CPU cores for improved performance with many predictors. On Unix/Mac systems, uses fork-based parallelism via `mclapply`; on Windows, uses socket clusters via `parLapply`. Set to `FALSE` for sequential processing.

`n_cores` Integer specifying the number of CPU cores to use for parallel processing. Default is `NULL`, which automatically detects available cores and uses `detectCores() - 1`. During R CMD check, the number of cores is automatically limited to 2 per CRAN policy. Ignored when `parallel = FALSE`.

`number_format` Character string or two-element character vector controlling thousand and decimal separators in formatted output. Named presets:

- `"us"` - Comma thousands, period decimal: `1,234.56` [default]
- `"eu"` - Period thousands, comma decimal: `1.234,56`
- `"space"` - Thin-space thousands, period decimal: `1 234.56` (SI/ISO 31-0)
- `"none"` - No thousands separator: `1234.56`

Or provide a custom two-element vector `c(big.mark, decimal.mark)`, *e.g.*, `c("'", ".")` for Swiss-style: `1'234.56`.

When `NULL` (default), uses `getOption("summata.number_format", "us")`. Set the global option once per session to avoid passing this argument repeatedly:

```
options(summata.number_format = "eu")
```

`verbose` Logical. If `TRUE`, displays model fitting warnings (*e.g.,* singular fit, convergence issues). If `FALSE` (default), routine fitting messages are suppressed while unexpected warnings are preserved. When `NULL`, uses `getOption("summata.verbose", FALSE)`.

`...` Additional arguments passed to the underlying model fitting functions (`glm`, `lm`, `coxph`, *etc.*). Common options include `weights`, `subset`, `na.action`, and model-specific control parameters.

Details

Analysis Approach:

The function implements a comprehensive univariable screening workflow:

1. For each predictor in `predictors`, fits a separate model: `outcome ~ predictor`
2. Extracts coefficients, confidence intervals, and p-values from each model

3. Combines results into a single table for easy comparison
4. Formats output for publication with appropriate effect measures

Each predictor is tested *independently* - these are crude (unadjusted) associations that do not account for confounding or interaction effects.

When to Use Univariable Screening:

- **Initial variable selection**: Identify predictors associated with the outcome before building multivariable models
- **Hypothesis generation**: Explore potential associations in exploratory analyses
- **Understanding crude associations**: Report unadjusted effects alongside adjusted estimates
- **Variable reduction**: Use p-value thresholds (*e.g.,* $p < 0.20$) to identify candidates for multivariable modeling
- **Checking multicollinearity**: Compare univariable and multivariable effects to identify potential collinearity

Threshold for *p*-values:

The `p_threshold` parameter controls the significance threshold used in the printed summary to count how many predictors are significant. All predictors are always included in the output table regardless of this setting.

Effect Measures by Model Type:

- **Logistic regression** (`model_type = "glm"`, `family = "binomial"`): Odds ratios (OR)
- **Cox regression** (`model_type = "coxph"`): Hazard ratios (HR)
- **Poisson regression** (`model_type = "glm"`, `family = "poisson"`): Rate/risk ratios (RR)
- **Negative binomial** (`model_type = "negbin"`): Rate ratios (RR)
- **Linear regression** (`model_type = "lm"` or GLM with identity link): Raw coefficient estimates
- **Gamma regression** (`model_type = "glm"`, `family = "Gamma"`): Multiplicative effects (with default log link)

Memory Considerations:

When `keep_models = FALSE` (default), fitted models are discarded after extracting results to conserve memory. Set `keep_models = TRUE` only when the following are needed:

- Model diagnostic plots
- Predictions from individual models
- Additional model statistics not extracted by default
- Further analysis of specific models

Value

A data.table with S3 class "uniscreen_result" containing formatted univariable screening results. The table structure includes:

Variable Character. Predictor name or custom label (from labels)

Group Character. For factor variables: category level. For continuous variables: typically empty or descriptive statistic label

n Integer. Sample size used in the model (if show_n = TRUE)

n_group Integer. Sample size for this specific factor level (factor variables only)

events Integer. Total number of events in the model for survival or logistic regression (if show_events = TRUE)

events_group Integer. Number of events for this specific factor level (factor variables only)

OR/HR/RR/Coefficient (95% CI) Character. Formatted effect estimate with confidence interval. Column name depends on model type: "OR (95% CI)" for logistic, "HR (95% CI)" for survival, "RR (95% CI)" for counts, "Coefficient (95% CI)" for linear models

***p*-value** Character. Formatted *p*-value from the Wald test

The returned object includes the following attributes accessible via attr():

raw_data data.table. Unformatted numeric results with separate columns for coefficients, standard errors, confidence interval bounds, *etc.* Suitable for further statistical analysis or custom formatting

analysis_counts List. How much of the supplied data reached the model, with elements n_supplied, n_analyzed, n_missing_outcome and n_missing_predictor, plus events_supplied and events_analyzed where the model type carries an event count. Each predictor is screened in its own model, so n_analyzed holds one value per predictor and the reported sample may be a range. The print() method always reports the analyzed sample

models List (if keep_models = TRUE). Named list of fitted model objects, with predictor names as list names. Access specific models via attr(result, "models")[["predictor_name"]]

outcome Character. The outcome variable name used

model_type Character. The regression model type used

conf_level Numeric. The confidence level of the intervals, which uniforest() uses to draw the same intervals

model_scope Character. Always "Univariable" for screening results

screening_type Character. Always "univariable" to identify the analysis type

p_threshold Numeric. The *p*-value threshold used for significance

significant Character vector. Names of predictors with *p*-value below the screening threshold, suitable for passing directly to downstream modeling functions

See Also

fit for fitting a single multivariable model, fullfit for complete univariable-to-multivariable workflow, compfit for comparing multiple models, m2dt for converting individual models to tables, uniforest for forest plots of screening results, forestsave for exporting forest plots, tablesave for exporting tables

Other regression functions: compfit(), fit(), fullfit(), multifit(), print.compfit_result(), print.fit_result(), print.fullfit_result(), print.multifit_result(), print.uniscreen_result()

Examples

```
# Load example data
data(clintrial)
data(clintrial_labels)

# Example 1: Basic logistic regression screening
screen1 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking", "hypertension"),
    model_type = "glm",
    family = "binomial",
    labels = clintrial_labels,
    parallel = FALSE
)
print(screen1)



# Example 2: With custom variable labels
screen2 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "treatment"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(screen2)

# Example 3: Filter by p-value threshold
# Only keep predictors with p < 0.20 (common for screening)
screen3 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking", "hypertension",
                   "diabetes", "stage"),
    p_threshold = 0.20,
    labels = clintrial_labels,
    parallel = FALSE
)
print(screen3)
# Only significant predictors are shown

# Example 4: Cox proportional hazards screening
library(survival)
cox_screen <- uniscreen(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "treatment", "stage", "grade"),
    model_type = "coxph",
    labels = clintrial_labels,
    parallel = FALSE
```

```
)
print(cox_screen)
# Returns hazard ratios (HR) instead of odds ratios

# Example 5: Keep models for diagnostics
screen5 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "bmi", "creatinine"),
    keep_models = TRUE,
    parallel = FALSE
)

# Access stored models
models <- attr(screen5, "models")
summary(models[["age"]])
plot(models[["age"]])  # Diagnostic plots

# Example 6: Linear regression screening
linear_screen <- uniscreen(
    data = clintrial,
    outcome = "bmi",
    predictors = c("age", "sex", "smoking", "creatinine", "hemoglobin"),
    model_type = "lm",
    labels = clintrial_labels,
    parallel = FALSE
)
print(linear_screen)

# Example 7: Poisson regression for equidispersed count outcomes
# fu_count has variance ~= mean, appropriate for standard Poisson
poisson_screen <- uniscreen(
    data = clintrial,
    outcome = "fu_count",
    predictors = c("age", "stage", "treatment", "surgery"),
    model_type = "glm",
    family = "poisson",
    labels = clintrial_labels,
    parallel = FALSE
)
print(poisson_screen)
# Returns rate ratios (RR)

# Example 8: Negative binomial for overdispersed counts
# ae_count has variance > mean (overdispersed), use negbin
if (requireNamespace("MASS", quietly = TRUE)) {
    nb_screen <- uniscreen(
        data = clintrial,
        outcome = "ae_count",
        predictors = c("age", "treatment", "diabetes", "surgery"),
        model_type = "negbin",
        labels = clintrial_labels,
        parallel = FALSE
```

```
    )
    print(nb_screen)
}

# Example 9: Gamma regression for positive continuous outcomes (\emph{e.g.,} costs)
gamma_screen <- uniscreen(
    data = clintrial,
    outcome = "los_days",
    predictors = c("age", "sex", "treatment", "surgery"),
    model_type = "glm",
    family = Gamma(link = "log"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(gamma_screen)

# Example 10: Hide reference rows for factor variables
screen10 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("treatment", "stage", "grade"),
    reference_rows = FALSE,
    labels = clintrial_labels,
    parallel = FALSE
)
print(screen10)

# Example 11: Customize decimal places
screen11 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "bmi", "creatinine"),
    digits = 3,      # 3 decimal places for OR
    p_digits = 4,    # 4 decimal places for p-values
    labels = clintrial_labels
)
print(screen11)

# Example 12: Hide sample size and event columns
screen12 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi"),
    show_n = FALSE,
    show_events = FALSE,
    labels = clintrial_labels,
    parallel = FALSE
)
print(screen12)

# Example 13: Access raw numeric data
screen13 <- uniscreen(
    data = clintrial,
```

```
    outcome = "os_status",
    predictors = c("age", "sex", "treatment"),
    parallel = FALSE
)
raw_data <- attr(screen13, "raw_data")
print(raw_data)

# Example 14: Force coefficient display instead of OR
screen14 <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "bmi"),
    model_type = "glm",
    family = "binomial",
    labels = clintrial_labels,
    parallel = FALSE,
    exponentiate = FALSE  # Show log odds instead of OR
)
print(screen14)

# Example 15: Screening with weights
screen15 <- uniscreen(
    data = clintrial,
    outcome = "Surv(os_months, os_status)",
    predictors = c("age", "sex", "bmi"),
    model_type = "coxph",
    weights = runif(nrow(clintrial), min = 0.5, max = 2),  # Random numbers for example
    labels = clintrial_labels,
    parallel = FALSE
)

# Example 16: Strict significance filter (p < 0.05)
sig_only <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking", "hypertension",
                   "diabetes", "ecog", "treatment", "stage", "grade"),
    p_threshold = 0.05,
    labels = clintrial_labels,
    parallel = FALSE
)

# Check how many predictors passed the filter
n_significant <- length(unique(sig_only$Variable[sig_only$Variable != ""]))
cat("Significant predictors:", n_significant, "\n")

# Example 17: Complete workflow - screen then use in multivariable
# Step 1: Screen with liberal threshold
candidates <- uniscreen(
    data = clintrial,
    outcome = "os_status",
    predictors = c("age", "sex", "bmi", "smoking", "hypertension",
                   "diabetes", "treatment", "stage", "grade"),
```

```
    p_threshold = 0.20,
    parallel = FALSE
)

# Step 2: Extract significant predictor names
sig_predictors <- attr(candidates, "significant")

# Step 3: Fit multivariable model with selected predictors
multi_model <- fit(
    data = clintrial,
    outcome = "os_status",
    predictors = sig_predictors,
    labels = clintrial_labels
)
print(multi_model)

# Example 18: Mixed-effects logistic regression (glmer)
# Accounts for clustering by site
if (requireNamespace("lme4", quietly = TRUE)) {
    glmer_screen <- uniscreen(
        data = clintrial,
        outcome = "os_status",
        predictors = c("age", "sex", "treatment", "stage"),
        model_type = "glmer",
        random = "(1|site)",
        family = "binomial",
        labels = clintrial_labels,
        parallel = FALSE
    )
    print(glmer_screen)
}

# Example 19: Mixed-effects linear regression (lmer)
if (requireNamespace("lme4", quietly = TRUE)) {
    lmer_screen <- uniscreen(
        data = clintrial,
        outcome = "biomarker_x",
        predictors = c("age", "sex", "treatment", "smoking"),
        model_type = "lmer",
        random = "(1|site)",
        labels = clintrial_labels,
        parallel = FALSE
    )
    print(lmer_screen)
}

# Example 20: Mixed-effects Cox model (coxme)
if (requireNamespace("coxme", quietly = TRUE)) {
    coxme_screen <- uniscreen(
        data = clintrial,
        outcome = "Surv(os_months, os_status)",
        predictors = c("age", "sex", "treatment", "stage"),
        model_type = "coxme",
```

```
        random = "(1|site)",
        labels = clintrial_labels,
        parallel = FALSE
    )
    print(coxme_screen)
}

# Example 21: Mixed-effects with nested random effects
# Patients nested within sites
if (requireNamespace("lme4", quietly = TRUE)) {
    nested_screen <- uniscreen(
        data = clintrial,
        outcome = "os_status",
        predictors = c("age", "treatment"),
        model_type = "glmer",
        random = "(1|site/patient_id)",
        family = "binomial",
        labels = clintrial_labels,
        parallel = FALSE
    )
}

# Example 22: Quasipoisson for overdispersed count data
# Alternative to negative binomial when MASS not available
quasi_screen <- uniscreen(
    data = clintrial,
    outcome = "ae_count",
    predictors = c("age", "treatment", "diabetes", "surgery", "stage"),
    model_type = "glm",
    family = "quasipoisson",
    labels = clintrial_labels,
    parallel = FALSE
)
print(quasi_screen)
# Adjusts standard errors for overdispersion

# Example 23: Quasibinomial for overdispersed binary data
quasibin_screen <- uniscreen(
    data = clintrial,
    outcome = "any_complication",
    predictors = c("age", "bmi", "diabetes", "surgery", "ecog"),
    model_type = "glm",
    family = "quasibinomial",
    labels = clintrial_labels,
    parallel = FALSE
)
print(quasibin_screen)

# Example 24: Inverse Gaussian for highly skewed positive data
invgauss_screen <- uniscreen(
    data = clintrial,
    outcome = "recovery_days",
    predictors = c("age", "surgery", "pain_score", "los_days"),
```

```
    model_type = "glm",
    family = inverse.gaussian(link = "log"),
    labels = clintrial_labels,
    parallel = FALSE
)
print(invgauss_screen)
```

Index

∗ **datasets**
clintrial, 6
clintrial_labels, 8
∗ **descriptive functions**
desctable, 21
survtable, 94
∗ **export functions**
table2docx, 102
table2html, 109
table2pdf, 115
table2pptx, 124
table2rtf, 131
table2tex, 138
tablesave, 145
∗ **regression functions**
compfit, 9
fit, 29
fullfit, 46
multifit, 74
uniscreen, 153
∗ **sample data**
clintrial, 6
clintrial_labels, 8
∗ **utility functions**
m2dt, 70
∗ **visualization functions**
autoforest, 3
coxforest, 14
forestsave, 43
glmforest, 56
lmforest, 63
multiforest, 88
recdims, 93
uniforest, 148

autoforest, 3, *19*, *37*, *45*, *53*, *62*, *68*, *92, 93*, *152*
autoforest(), *19*, *45*, *62*, *68*, *92, 93*, *152*
autotable *(*tablesave*)*, 145

clintrial, 6, *9*
clintrial_labels, *7*, 8
compfit, 9, *37*, *53*, *159*
compfit(), *37*, *53*, *82*, *159*
coxforest, *5*, 14, *45*, *62*, *68*, *73*, *92*, *152*
coxforest(), *5*, *45*, *62*, *68*, *92, 93*, *152*
coxph, *19*, *34*, *157*

desctable, 21, *53*, *98*, *113*, *120*, *142*
desctable(), *98*

family, *10*, *32*, *49*, *76*, *156*
fit, *5*, *12*, *19*, *26*, 29, *53*, *62*, *68*, *73*, *82*, *98*, *113*, *120*, *142*, *159*
fit(), *12*, *53*, *82*, *159*
flextable, *107*, *129*, *135*
forestsave, *5*, *19*, *37*, 43, *53*, *62*, *68*, *82*, *92*, *93*, *152*, *159*
forestsave(), *5*, *19*, *62*, *68*, *92, 93*, *152*
fullfit, *5*, *12*, *37*, 46, *82*, *159*
fullfit(), *12*, *37*, *82*, *159*

glm, *34*, *62*, *157*
glmer, *34*
glmforest, *5*, *19*, *45*, 56, *68*, *73*, *92*, *152*
glmforest(), *5*, *19*, *45*, *68*, *92, 93*, *152*

lm, *34*, *68*, *157*
lmforest, *5*, *19*, *45*, *62*, 63, *73*, *92*, *152*
lmforest(), *5*, *19*, *45*, *62*, *92, 93*, *152*

m2dt, *37*, 70, *159*
multifit, *5*, 74, *92*
multifit(), *12*, *37*, *53*, *159*
multiforest, *5*, *19*, *45*, *62*, *68*, *82*, 88, *152*
multiforest(), *5*, *19*, *45*, *62*, *68*, *93*, *152*

print.compfit_result(), *12*, *37*, *53*, *82*, *159*
print.fit_result(), *12*, *37*, *53*, *82*, *159*

print.fullfit_result(), *12*, *37*, *53*, *82*, *159*
print.multifit_result(), *12*, *37*, *53*, *82*, *159*
print.survtable(), *26*, *98*
print.uniscreen_result(), *12*, *37*, *53*, *82*, *159*

read_docx, *104*, *107*
read_pptx, *126*, *129*
recdims, 93
recdims(), *5*, *19*, *45*, *62*, *68*, *92*, *152*

save_as_rtf, *135*
survdiff, *98*
survfit, *97, 98*
survtable, *26*, 94
survtable(), *26*

table2docx, *26*, *98*, 102, *113*, *120*, *129*, *135*, *142*, *146*
table2docx(), *113*, *120*, *129*, *135*, *142*, *146*
table2html, *26*, *107*, 109, *120*, *129*, *135*, *142*, *146*
table2html(), *107*, *120*, *129*, *135*, *142*, *146*
table2pdf, *26*, *98*, *107*, *113*, 115, *129*, *135*, *142*, *146*
table2pdf(), *107*, *113*, *129*, *135*, *142*, *146*
table2pptx, *107*, *113*, *120*, 124, *135*, *142*, *146*
table2pptx(), *107*, *113*, *120*, *135*, *142*, *146*
table2rtf, *107*, *113*, *120*, *129*, 131, *142*, *146*
table2rtf(), *107*, *113*, *120*, *129*, *142*, *146*
table2tex, *107*, *113*, *120*, *129*, *135*, 138, *146*
table2tex(), *107*, *113*, *120*, *129*, *135*, *146*
tablesave, *12*, *26*, *37*, *45*, *53*, *82*, *98*, *107*, *113*, *120*, *129*, *135*, *142*, 145, *159*
tablesave(), *107*, *113*, *120*, *129*, *135*, *142*

uniforest, *5*, *19*, *45*, *62*, *68*, *92*, 148, *159*
uniforest(), *5*, *19*, *45*, *62*, *68*, *92, 93*
uniscreen, *5*, *37*, *53*, *82*, *152*, 153
uniscreen(), *12*, *37*, *53*, *82*

xtable, *110*, *118*, *140*