Package: spicy (via r-universe)

November 3, 2026

Title Publication-Ready Tables for Descriptive Statistics and Regression Models

Version 0.14.1.9000

Description Provides publication-ready tables for descriptive statistics and regression models: frequency tables and cross-tabulations with association measures (Cramér's V, Kendall's Tau-b, and others), categorical and continuous summary tables, by group or from a complex survey design, and regression tables for one or more models side by side, across more than thirty model classes from mixed-effects to survival and Bayesian, with robust standard errors, average marginal effects, and univariable screening. Tables follow APA conventions by default, can switch to named journal styles such as JAMA, NEJM, or The Lancet, and render identically in the console and in 'gt', 'tinytable', 'flextable', 'Word', 'Excel', or the clipboard. Declared missing values in labelled data are honored and disclosed throughout the descriptive tables. Helpers cover codebooks, variable inspection, and row-wise summaries.

License MIT + file LICENSE

URL https://amaltawfik.github.io/spicy/, https://github.com/amaltawfik/spicy

BugReports https://github.com/amaltawfik/spicy/issues

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Imports crayon, dplyr, labelled, rlang (>= 1.1.0), sandwich (>= 3.1-2), stats, stringr, tibble, tidyselect, utils

Suggests bit64, boot, broom, car, clipr, clubSandwich, effectsize, emmeans, estimatr (>= 1.0.0), fixest (>= 0.11), flexsurv (>= 2.2), flextable, geepack (>= 1.3.9), gt, haven, hms, htmltools, insight, glmmTMB (>= 1.1.7), knitr, lme4 (>= 1.1-35), lmerTest

(>= 3.1-3), lmtest, merDeriv (>= 0.2.6), marginaleffects, modelsummary, collapse, MASS, mgcv (>= 1.9), nlme (>= 3.1-160), nnet, numDeriv, ordinal (>= 2022.11), AER (>= 1.2), betareg (>= 3.1), mlogit (>= 1.1), pscl (>= 1.5), quantreg (>= 5.86), quarto, RLRsim, reformulas, rms (>= 6.7), sampleSelection (>= 1.2), posterior (>= 1.5.0), rstanarm (>= 2.21), brms (>= 2.20), loo (>= 2.6), rstan, survey (>= 4.5), survival (>= 3.5), officer, openxlsx2, parameters, pdftools, performance, rmarkdown, spelling, testthat (>= 3.0.0), tinytable, withr

VignetteBuilder knitr

Depends R (>= 4.1.0)

Config/testthat/edition 3

LazyData true

Config/roxygen2/version 8.1.1

Config/pak/sysreqs make libicu-dev libx11-dev zlib1g-dev

Repository https://amaltawfik.r-universe.dev

Date/Publication 2026-11-03 00:57:01 UTC

RemoteUrl https://github.com/amaltawfik/spicy

RemoteRef HEAD

RemoteSha 652f19c22e4791cf10e8b9f98628ddfce9f95555

Contents

as_flextable.spicy_flextable . . . 3
as_structured . . . 4
assoc_measures . . . 8
code_book . . . 9
contingency_coef . . . 15
copy_clipboard . . . 17
count_n . . . 19
cramer_v . . . 22
cross_tab . . . 24
freq . . . 28
gamma_gk . . . 33
goodman_kruskal_tau . . . 34
inline . . . 35
kendall_tau_b . . . 38
kendall_tau_c . . . 39
label_from_names . . . 41
lambda_gk . . . 42
mean_n . . . 44
phi . . . 47
sochealth . . . 49
somers_d . . . 51

spicy_labels 53
spicy_style 55
sum_n 63
table_categorical 66
table_categorical_svy 83
table_continuous 90
table_continuous_lm 107
table_continuous_svy 128
table_multiresponse 136
table_outcome 146
table_regression 155
table_regression_models 183
table_regression_uv 188
uncertainty_coef 192
var_labels 194
varlist 195
yule_q 199

Index **201**

`as_flextable.spicy_flextable` *Convert a spicy flextable output to a plain flextable*

Description

`table_regression()` and `table_continuous_lm()` return their `output = "flextable"` tables with a lightweight `spicy_flextable` class tag whose only job is HTML note styling. Every flextable verb already works on the tagged object; this method returns the clean underlying `flextable::flextable()` – the note is already part of its footer – for workflows that want the untagged object (e.g. custom knit hooks, `flextable::save_as_docx()` pipelines, or composition with other flextable tooling), mirroring `gtsummary::as_flex_table()`.

Usage

```
## S3 method for class 'spicy_flextable'
as_flextable(x, ...)
```

Arguments

`x`	A `spicy_flextable` object.
`...`	Unused; for generic consistency.

Details

Rendering in Quarto / R Markdown does NOT require this conversion: the `knit_print` method auto-detects the output format and delegates to flextable's native rendering for non-HTML targets (Word, PowerPoint, PDF).

Value

A `flextable` object.

`as_structured` *Extract the typed (structured) view of a spicy table*

Description

spicy's tables return a *display* representation by default – a character `data.frame` with stars suffixes, en-dash for reference rows, bracketed `"[L, U]"` confidence intervals, and APA padding on p-values. This accessor returns the *typed* view that the output engines (Excel, gt, tinytable, flextable, clipboard) consume internally: a fully numeric body with CI pre-split into `LL` / `UL` columns, NAs for non-applicable / reference cells, plus per-cell markers and a format specification.

Usage

```
as_structured(x)
```

Arguments

`x`	A spicy table built with `output = "default"`: `table_regression()`, `table_categorical()`, `table_continuous()`, `table_continuous_lm()`, `table_outcome()`, `table_multiresponse()` or the survey twins `table_categorical_svy()` and `table_continuous_svy()`. All return the same schema.

Details

This is the right entry point for users who want to:

- **Filter coefficients programmatically**, e.g. `as_structured(tbl)$body[which(as_structured(tbl)$body$p < 0.05), ]` (`which()` drops the structurally empty rows – factor headers, reference levels – whose `p` is `NA`; a bare logical index would keep them as all-`NA` rows).
- **Aggregate raw values across rows**, e.g. `mean(as_structured(tbl)$body[["B"]], na.rm = TRUE)`.
- **Build a custom downstream renderer** that consumes the same structured contract as spicy's built-in engines.

Value

A list with the structured view (see Details for the schema).

Schema

- `version` – integer contract version (see *Versioning* below).
- `body` – `data.frame` with a `Variable` character column, one or more numeric value columns, and four dot-prefixed *identity* columns at the end. Confidence intervals are split into a lower and an upper column, named as each family names them: `"95% CI: LL"` / `"95% CI: UL"` in a regression table (prefixed with the model label in multi-model output), `"95% CI LL"` / `"95% CI UL"` in `table_continuous()`, `table_continuous_lm()`, `table_outcome()` and `table_continuous_svy()`, `"CI lower"` / `"CI upper"` for the association measure of `table_categorical()`, and `"<group> % CI lower"` / `"<group> % CI upper"` in `table_categorical_svy()`. A cell with no value (reference level, term absent from a model, factor header) is `NA`.
- `body$.variable` – source variable of the row: the factor name on a factor header / level / reference row, the term otherwise, and the fit-statistic token (`"nobs"`, `"r2"`, `"fixed_effects"`, ...) on a fit-statistic row.
- `body$.level` – the factor level (or the absorbed factor of a fixed-effects block, or the grouping factor of an `"N (...)"` row); `NA` outside a factor.
- `body$.row_role` – what the row *is*: `"coef"`, `"factor_header"`, `"level"`, `"reference"`, `"fit_stat"`, `"outcome"`, `"vc"` (variance component) in a regression table, plus `"summary"` (a row summarizing one variable), `"group"` (a row keyed by one level of `by`), `"missing"` (a row keyed by the *missing* value), `"none"` (the respondents who chose no item of a multiple-response question) and `"total"` (the Total row of `table_categorical()` under `total_row = TRUE`) in the descriptive ones. The role is the key a consumer matches on: `"(Missing)"` is a display label – translatable, and relabeled through `options(spicy.labels)` – and the role survives both.
- `body$.indent` – display indent depth of the label (`0` or `1`). Renderers indent `.indent > 0` rows; the label text itself is already indented in the character body only.
- `cell_status` – per-**cell** semantics, keyed by column name, one character vector as long as `body` per column that needs one: `"reference"` (reference level, *in this estimate block and this model*), `"undefined"` (the statistic applies to the row but no number expresses it – an unavailable variance-component standard error, a fit statistic undefined for one model's class), `""` otherwise. Both marked statuses display as an en-dash. A cell whose value is `NA` with no status is *absent* and displays blank. Columns with no marked cell are omitted.
- `outcome_labels_by_col` – for the outcome row (explicit `outcome_labels` with two or more models), the display label keyed by each model's first structured column name.
- `col_meta` – per-column metadata keyed by structured column name (token, model_id, precision, p-style, below-threshold, CI pair / role / label). A column whose cells cannot be reconstructed from one number – the `"events/N"` counts of `show_columns = "n_events"` – also carries `display_cells`, a character vector as long as `body` holding the display string of each cell (`NA` where the number formats normally). A renderer must prefer it over the numeric value.
- `stars` – `NULL` unless `stars` was requested, otherwise a list with `thresholds` (symbol to p cutoff) and `markers` (per-cell marker strings, keyed by column name, `""` where a cell takes none).
- `spanners` – named list mapping a grouping label to its column indices in `body`: the model labels of a multi-model regression table, the `by` levels of a categorical one (each spanning its `n` / `%` pair).

- `ci_pairs` – list of `(label, cols)` entries describing each CI pair in body.
- `format_spec` – global format defaults (decimal mark, digits, p-style, CI level, etc.).

Descriptive tables

The descriptive tables return the same schema; only the `col_meta` tokens and the row roles they emit differ. `table_outcome()` has the columns of `table_continuous()` and the rows of `table_categorical()`: a `"summary"` row for the whole sample, then a `"factor_header"` row per grouping and its `"level"` (or `"missing"`) rows. The survey twins follow the table they twin, `table_categorical_svy()` with the interval of each percentage (token `"prop_ci"`) and no association measure.

- `table_categorical()` – one `"factor_header"` row per variable then one `"level"` (or `"missing"`) row per category, and under `total_row = TRUE` the `"total"` row that closes each block, exactly as the console lays them out. Tokens: `"n"`, `"pct"`, `"p"`, `"assoc"` (the association measure), `"assoc_ci"` (its bounds) and `"smd"` (the standardized mean difference, when `smd = TRUE`). In a by table each group owns an `n / %` pair with its own spanner, and the margin column is flagged `col_meta$<col>$total`, never matched on the label `"Total"`.
- `table_continuous()` – one `"summary"` row per variable, or one `"group"` row per level of by (`"missing"` for the missing-by group), with the level in `.level`. Tokens are the `show_columns` vocabulary itself (`"m"`, `"sd"`, `"med"`, `"iqr"`, `"med_iqr"`, `"q1"`, `"q3"`, `"min"`, `"max"`, `"ci"`, `"med_ci"`, `"n"`, `"weighted_n"`) plus `"statistic"`, `"p"`, `"es"` and `"smd"` for the group comparison – the last four are `col_meta` tokens without being `show_columns` values, because they belong to a comparison BETWEEN groups rather than to any one group. A statistic another variable displays is *absent* (`NA`, no status); one that applies but has no value is `"undefined"`.
- `table_continuous_lm()` – one `"summary"` row per outcome. Tokens: `"emmean"` (a marginal mean; the column says which level in `col_meta$<col>$level`, also its `display_label`: the table heads each mean by its level under the mean symbol), `"delta"` (the contrast), `"b"` (a numeric predictor's slope), `"ci"`, `"statistic"`, `"p"`, `"r2"`, `"es"`, `"n"`, `"weighted_n"`.
- `table_multiresponse()` – one `"summary"` row per item, then the rows of `none_row`: a `"none"` row (`.variable == "none"`) for the respondents of the base with no item chosen and a `"missing"` row (`.variable == "no_data"`) for those with no data for the question, then one `"fit_stat"` row (`.variable == "nobs"`) holding the respondents of each group. Tokens: `"n"`, `"pct"` (percentage of cases), `"pct_responses"` (percentage of responses), `"statistic"` and `"p"`. Each group owns its columns under its own spanner, and the margin is flagged `col_meta$<col>$total`.

Cells the console builds from more than one number – a `"Mdn [Q1, Q3]"`, a test gloss, an effect size with its interval – keep their numeric anchor in body and their exact printed string in `col_meta$<col>$display_cells`. So does a cell of a variable that a named `digits` gives decimals other than the column's `precision`, which stays the table's `digits`. `stars` is always `NULL`: descriptive tables carry no significance markers.

Under `p_adjust`, the p-value column holds the adjusted values and its `col_meta` adds `p_adjust` (the method), `p_adjust_m` (the number of tests adjusted together) and `p_unadjusted` (the raw p-value of each cell, as long as body). In `table_categorical()` and `table_multiresponse()`, the p-value column's `col_meta` also carries `test_type`, the test of each cell, as long as body (`"chi_squared"`, `"n1_chi_squared"`, `"fisher_exact"`, `"fisher_midp"`, `"fisher_exact_mc"`, `NA` where the cell holds no p-value).

Versioning

`version` says which contract an object carries. Version `3` moved row identity out of index vectors and into the body itself:

- `reference_rows` and `reference_models_by_row` become `cell_status`, which marks the reference cell instead of the whole row – the row-scoped flag was blanking estimate blocks that have no per-level reference at all.
- `factor_header_rows` becomes `.row_role == "factor_header"`.
- `fit_stat_rows` becomes `.row_role == "fit_stat"`.
- `level_rows` becomes `.indent > 0`.
- `outcome_row` becomes `.row_role == "outcome"`.

Index vectors are the structure that corrupts as soon as two bodies are stacked or merged, so they were removed rather than kept alongside. An object carrying an older contract (or none) is refused with the correspondence above; rebuild it with the function that produced it. An object carrying a *newer* contract than the spicy reading it is refused as well.

Version `3` also opened the accessor to the descriptive families, which had no typed view before: `.row_role` gained `"summary"`, `"group"` and `"missing"`, then `"none"` with `table_multiresponse()` and `"total"` with the `total_row` of `table_categorical()`. The vocabulary is extended by addition – an existing role never changes meaning.

Version `4` renames one component: in the `format_spec` of `table_categorical()`, the number of decimals of the association measure and the SMD is `effect_size_digits`, the key of the other tables, where it was `v_digits` (the argument was renamed alike). An object carrying version `3` is refused as an older contract.

See Also

`table_regression()`, `table_categorical()`, `table_continuous()`, `table_continuous_lm()` for the user-facing entry points.

Examples

```
fit <- lm(mpg ~ wt + factor(cyl), data = mtcars)
tbl <- table_regression(fit)
s <- as_structured(tbl)
s$body                                # raw numeric body
s$body[which(s$body$p < 0.05), ]      # filter significant rows
# which() drops the structural NA rows (headers, reference levels)
s$body$.row_role                      # what each row is
s$body[s$body$.variable == "wt", ]    # address a row by its variable
s$col_meta$B                          # column metadata for B

# The same schema on a descriptive table.
ct <- table_categorical(mtcars, c(cyl, gear), by = am)
sc <- as_structured(ct)
sc$body[, c("Variable", ".variable", ".level", ".row_role")]
sc$spanners                           # one per `by` group
```

assoc_measures *Association measures summary table*

Description

assoc_measures() computes a range of association measures for a two-way contingency table and returns them in a tidy data frame.

Usage

```
assoc_measures(
  x,
  type = c("all", "nominal", "ordinal"),
  conf_level = 0.95,
  digits = 3L
)
```

Arguments

x	A contingency table (of class table).
type	Which family of measures to compute: "all" (default), "nominal", or "ordinal".
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3).

Details

type = "all" (the default) returns all nominal and ordinal measures. Use type = "nominal" or type = "ordinal" to restrict the output to a single family.

The nominal family includes cramer_v(), contingency_coef(), lambda_gk(), goodman_kruskal_tau(), uncertainty_coef(), and (for 2x2 tables) phi() and yule_q().

The ordinal family includes gamma_gk(), kendall_tau_b(), kendall_tau_c(), and somers_d().

Measures that are undefined on the given table appear as NA rows (printed as an en dash). The classed warnings the individual functions raise (e.g. spicy_undefined_stat) are re-emitted once per distinct message after the table is assembled, so condition handlers and suppressWarnings() behave as they do for the individual functions.

Standard error formulas follow the DescTools implementations (Signorell et al., 2024), except for Kendall's Tau-b, whose ASE follows Brown and Benedetti (1977) as printed by SPSS / PSPP CROSSTABS; see kendall_tau_b().

Value

A data frame with columns `measure`, `estimate`, `se`, `ci_lower`, `ci_upper`, and `p_value`. The `p_value` comes from two test families:

- **Pearson chi-squared test of independence** for Cramér's V, Phi, and the Contingency Coefficient (the three chi-squared-derived nominal measures). All three carry the same chi-squared *p*-value on a given table.
- **Wald z-test of H0: measure = 0** for every other measure: Yule's Q, Lambda, Goodman-Kruskal's Tau, the Uncertainty Coefficient, and all ordinal measures (Gamma, Tau-b, Tau-c, Somers' D).

Direction-dependent measures (`lambda_gk()`, `goodman_kruskal_tau()`, `uncertainty_coef()`, `somers_d()`) contribute one row per direction (`symmetric` / `R|C` / `C|R` where applicable), so the output has more rows than the number of helper functions.

References

Agresti, A. (2002). *Categorical Data Analysis* (2nd ed.). Wiley.

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

Liebetrau, A. M. (1983). *Measures of Association*. Sage.

Signorell, A. et al. (2024). *DescTools: Tools for Descriptive Statistics*. R package.

See Also

`cramer_v()`, `gamma_gk()`, `kendall_tau_b()`

Other association measures: `contingency_coef()`, `cramer_v()`, `gamma_gk()`, `goodman_kruskal_tau()`, `kendall_tau_b()`, `kendall_tau_c()`, `lambda_gk()`, `phi()`, `somers_d()`, `uncertainty_coef()`, `yule_q()`

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
assoc_measures(tab)
assoc_measures(tab, type = "nominal")
assoc_measures(tab, type = "ordinal")
```

---

code_book *Build a codebook of a data frame*

Description

`code_book()` documents the variables of a data frame: one row per variable (position, name, label, type, valid and missing counts, summary statistics), and one row per category of its categorical and logical variables (code, label, count, percentages). The codebook prints as the list of variables, and `output` writes it to an Excel workbook or to a PDF.

Counts and percentages are unweighted: they describe the data file and are not estimates for a population.

Usage

```
code_book(
  x,
  ...,
  title = "Codebook",
  subtitle = NULL,
  authors = NULL,
  notes = NULL,
  source = NULL,
  values = 100,
  range = TRUE,
  factor_levels = c("all", "observed"),
  user_na = TRUE,
  decimal_mark = NULL,
  font = NULL,
  font_code = NULL,
  font_size = 10,
  colors = NULL,
  paper = c("a4", "letter"),
  index_columns = NULL,
  output = NULL
)
```

Arguments

`x`	A data frame or tibble.
`...`	Optional tidyselect-style column selectors (e.g. `starts_with("bmi")`, `where(is.numeric)`). Columns can be selected or reordered, but renaming selections is not supported.
`title`	Title of the codebook, such as the name of the study. The PDF adds the word "Codebook" above it on the cover and before it in the page header, unless the title already contains the word. Defaults to `"Codebook"`. `NULL` leaves the console and the Excel file without a title; the cover of the PDF then shows "Codebook" once.
`subtitle`	Subtitle of the codebook, under the title: the wave, the edition, the extract (`"Wave 3, 2026, public-use file"`).
`authors`	Authors of the codebook: `NULL` (the default), a character vector whose names are the authors and whose values are their affiliations (`c("Jane Doe" = "University of Somewhere")`; an unnamed element is a name without affiliation), or a list of

lists with name and the optional affiliation and orcid. An ORCID given as its https://orcid.org/ address is kept as the identifier alone.

notes — Character vector of notes on the data (source, exclusions, coding rules, ...), one note per element; blank notes are dropped. In the PDF, an element that starts with "- " or "* " is a list item, consecutive items making one list, and any other element is a paragraph: notes = c("Wave 3 only.", "- Weight: design weight.", "- BMI: kg/m2."). A note may carry *italics*, **bold**, and `code` (not nested), and a web address or a doi: becomes a link. The PDF renders these, turns straight apostrophes into typographic ones and, in French, the space before :, ;, !, or ? into a non-breaking space; the console and the Excel file show the words without the marks.

source — Named character vector mapping the current column names to the codes they had in the source file, the vector dplyr::rename(all_of()) takes. Its names must be selected columns, and each code must be non-empty.

values — The maximum number of categories listed per variable in the values table; under factor_levels = "all", unused levels and codes count. A variable with more lists its first values categories in their order, with their counts and percentages of the whole variable, then its declared and system missing values; n_categories in variables gives the total, and the PDF adds a row for the categories not listed. Defaults to 100; Inf lists them all.

range — Logical. If TRUE (the default), variables gives the minimum and maximum of each numeric variable and the earliest and latest date of each date; FALSE drops those four columns.

factor_levels — Character. "all" (the default; varlist() uses "observed") lists every declared level of a factor, every labelled code, every code of na_values, and both values of a logical, with a count of 0 when unused. "observed" lists only the values present in the data.

user_na — Logical. If TRUE (the default), declared missing values count as missing: they are left out of n_valid, counted in n_missing and n_declared_missing, and listed in values with declared_missing = TRUE. If FALSE, they count as valid values. See the "Declared missing values" section.

decimal_mark — Decimal mark of the numbers in the PDF, a single character such as "." or ",". NULL (the default) takes the mark of options(spicy.style), then the one of the language (options(spicy.language = "fr") gives the comma), then ".". The console list prints counts only, and the Excel file keeps numbers as numbers.

font, font_code — Fonts of the PDF: font for the text, the values and their codes included, and font_code for the variable names and the source codes. NULL (the default) uses New Computer Modern and DejaVu Sans Mono, which Typst embeds, so the PDF looks the same on every machine for the characters these fonts cover; any other character falls back to a font found on the machine. Any other font must be one Typst finds, named exactly as quarto typst fonts lists it; a .typ output keeps the name as given, unchecked. A font also sets the font of the Excel file, which otherwise keeps its default font.

font_size — Size of the text of the PDF, in points: 10 (the default), or a number from 6 to 24. The title of the cover and the small size of a long name follow it; the margins do not.

`colors` Named character vector of `"#RRGGBB"` colors replacing part of the palette of the PDF: `primary` (title, headings, and the text of table headers), `accent` (the word "Codebook" above the title, and the links), `band` (behind table headers), `band_dark` (the band of each variable, under white text), `zebra` (behind the label rows of a variable), `grid` (rules), `text`, and `muted`. A `band` given without `zebra` brings a lighter tint of itself as `zebra`, and a `band_dark` given without `grid` a light tint of itself as `grid`. The headers of the Excel file take `primary` and `band` too.

`paper` Paper size of the PDF: `"a4"` (the default) or `"letter"`.

`index_columns` Columns of the index of variables at the end of the PDF: `1` or `2`. `NULL` (the default) sets two columns past 40 variables when no name exceeds 40 characters, one column otherwise. A name longer than that may overflow a column of two.

`output` `NULL` (the default) returns the codebook, which prints as the list of variables. A path writes the codebook to that file, in the format of its extension, and returns it invisibly: `.xlsx` for an Excel workbook (this requires `openxlsx2`), `.pdf` for a PDF (this requires the `quarto` package and Quarto 1.7 or later, found on the PATH or through the `QUARTO_PATH` environment variable), `.typ` for the Typst source of that PDF. The path names a file, in a directory that exists.

Details

The type of a variable is read off its R class, never guessed: a factor is categorical (nominal), an ordered factor categorical (ordinal), a `haven_labelled` vector categorical (labelled codes), an integer or double vector numeric, a logical, character, or `Date` vector logical, text, or date, a `POSIXct` or `POSIXlt` vector date-time, and an `hms` vector time. The level of measurement comes from the declaration alone: a factor whose order was not declared with `ordered()` is nominal. A vector of any other class is shown by its first class (`difftime`, ...), without statistics. The R class stays in its own column. A `haven_labelled` vector without value labels is numeric, or text when it stores characters, and so is one whose value labels all sit on declared missing codes, with `user_na = TRUE`.

`values` lists the categories of factors and labelled vectors and the two values of a logical, then the declared missing values of the variable and, when it has any, a row for its system missing values (`code = "NA"`). Numeric, text, and date variables have no category rows: they appear in `values` only through their declared missing values. The codes of a labelled vector come in code order, the levels of a factor in level order. An explicit `NA` level of a factor (from `addNA()`) counts as missing, in the row of the system missing values.

Dates are written as `YYYY-MM-DD`, date-times as `YYYY-MM-DD HH:MM:SS` followed by the name of the time zone the variable carries, or `UTC` when it carries none, and times as `HH:MM:SS`.

The words the codebook adds (column headers, types, field and sheet names) follow `options(spicy.language)` when the codebook is built (see `spicy_labels()`); the variable and value labels of the data are never translated.

`print()` shows a line break or a tab in a label as a space, and shortens long variable labels, counting display columns, when that makes the list fit the console; the object keeps the labels whole.

Value

A `spicy_codebook` object, returned invisibly when `output` is given: a list with

`header` A list: `title`, `subtitle`, `authors` (a tibble with `name`, `affiliation`, and `orcid`), `date`, `n_obs`, `n_vars`, `notes` (blank notes dropped), and `declared_missing`, a tibble of the declared missing values found in the data and, under `factor_levels = "all"`, of the declared codes no observation carries, sorted by code then label (`code`, `label`, `variables`, `n_variables`).

`variables` A tibble, one row per variable: `position` (the column's position in `x`), `name`, `label`, `type`, `class`, `source`, `n_valid`, `n_missing`, `n_declared_missing`, `declared_codes` (the `na_values` and `na_range` of a `haven_labelled_spss` vector, as text, the two parts separated by a semicolon; `NA` without them or under `user_na = FALSE`), `n_distinct` (the distinct values observed, of any variable), `n_categories` (the categories a categorical or logical variable declares, its levels, codes, or `TRUE` and `FALSE`, whether `values` lists them all or not; `NA` for a numeric, text, or date variable; it exceeds `n_distinct` when a category has no observation, and the Excel workbook shows it as "Categories" next to "Distinct values"), then `min`, `max`, `mean`, `sd`, and `median` for numeric variables and `earliest` and `latest` for dates and times. `range = FALSE` drops `min`, `max`, `earliest`, and `latest`.

`values` A tibble, one row per value: `position` and `variable` (of the variable), `code`, `label`, `declared_missing`, `n`, `pct_total`, and `pct_valid`. The row of the system missing values (`code = "NA"`) exists only for a variable that has missing values.

The attributes `language` and `decimal_mark` record the language and the decimal mark the codebook was built with, and `appearance` the look of its PDF: a list of `font`, `font_code`, `font_size`, `colors` (all eight), `paper`, and `index_columns`.

Excel output

The workbook has three worksheets, named in the language of the codebook. The first, `codebook`, holds the header as field-value pairs: title, subtitle, one row per author, date, numbers of observations and variables, declared missing values, notes, and the versions of spicy and R that wrote it, with the date and time of writing. The other two, `variables` and `values`, hold the two tables of the object from the first row, under the column headers the console and the PDF show (`code` is "Value", `n_valid` "Valid"), with a frozen header and filters; in `variables`, the position, name, and label stay in view while the statistics scroll. The workbook always has the same columns, in the same order: a column that does not apply to a variable, or to any variable of the file, is empty, never left out. Numbers stay numeric cells at their full precision, shown as the PDF shows them: the percentages to one decimal, the mean, SD, and median of a variable to three significant digits of its SD; a statistic that is not finite (of a column holding `Inf`) is an empty cell, and dates stay text, written as above. Each worksheet prints in landscape, fitted to the width of the page, its header row on every page.

PDF output

The PDF opens on a cover: the word "Codebook" above the title (unless the title already contains it), the subtitle, the authors with their affiliations and ORCID addresses, the date, and at the foot the versions of spicy and R that made it. A page about the data follows (numbers of observations and variables, then the notes and the declared missing values), then the list of variables with the page of each.

One sheet per variable comes next, under the heading "Variable sheets": a dark band with the position, name, and type (a name over 45 characters takes a row of its own), rows for the label, the declared missing codes, and the source code, then the counts (valid, missing, and distinct values),

the statistics, and the table of the values. The number of categories is not among the counts: the table of values shows them, one row each, and a row "Other categories (n)" the rest when `values` caps the list; `n_categories` is a column of the object and of the Excel workbook only. On a sheet that lists declared missing codes, a Missing column marks them with `M`; the row of the system missing values reads "System missing". Minimum and maximum are written at the precision of the data; mean, SD, and median at three significant digits of the SD. A negative statistic takes the minus sign of a printed table, and a code keeps the hyphen the data has, since a reader types it back into a filter; `spicy_style(minus_sign = "ascii")`, set through `options(spicy.style)`, keeps the hyphen on the statistics too. A sheet breaks across pages only when it does not fit on one; its table of values then repeats its header under "name (continued)". An index of the variables closes the document, sorted by name in byte order, where uppercase letters come before lowercase ones.

`code_book()` writes the Typst source and compiles it with the Typst that Quarto bundles; Typst warnings, such as a character missing from the fonts, arrive as one R warning of class `spicy_typst_warning`. Without Quarto, `output = "<path>.typ"` writes the same source, self-contained: `typst compile` makes the PDF on any machine with Typst 0.12 or later.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

See Also

`varlist()` to explore the variables in the Viewer; `freq()` for the frequency table of one variable; the article Explore variables and build codebooks.

Other variable inspection: `label_from_names()`, `varlist()`

Examples

```
code_book(sochealth)

cb <- code_book(
```

```
  sochealth,
  starts_with("bmi"),
  title = "Social health survey",
  subtitle = "Body mass index",
  authors = c("Jane Doe" = "University of Somewhere"),
  notes = "Simulated data (see ?sochealth)."
)
cb
cb$variables
cb$values

# Labelled survey data: declared missing codes count as missing and are
# flagged in `values`.
if (requireNamespace("haven", quietly = TRUE)) {
  trust <- haven::labelled_spss(
    c(1, 2, 2, 3, 4, 8, 9, 1, NA, 2),
    labels = c(
      "Not at all" = 1, "A little" = 2, "Somewhat" = 3, "A lot" = 4,
      "Don't know" = 8, "Refused" = 9
    ),
    na_values = c(8, 9),
    label = "Trust in the health system"
  )
  code_book(tibble::tibble(trust))$values
}

if (requireNamespace("openxlsx2", quietly = TRUE)) {
  path <- tempfile(fileext = ".xlsx")
  code_book(sochealth, output = path)
}

# The Typst source of the PDF, which `output = "<path>.pdf"` compiles
# when Quarto is installed.
code_book(sochealth, output = tempfile(fileext = ".typ"))
```

`contingency_coef` *Pearson's contingency coefficient*

Description

`contingency_coef()` computes Pearson's contingency coefficient C for a two-way contingency table.

Usage

```
contingency_coef(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

x	A contingency table (of class table).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

The contingency coefficient is $C = \sqrt{\chi^2/(\chi^2 + n)}$. It ranges from 0 (independence) to a maximum that depends on the table dimensions. No standard asymptotic standard error exists, so the confidence interval is not computed.

Value

Same structure as cramer_v(): a scalar when detail = FALSE, a named vector when detail = TRUE. The p-value tests the null hypothesis of no association (Pearson chi-squared test). CI values are NA because no standard asymptotic SE exists for C.

References

Liebetrau, A. M. (1983). *Measures of Association*. Sage.

See Also

cramer_v(), assoc_measures()

Other association measures: assoc_measures(), cramer_v(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), lambda_gk(), phi(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
contingency_coef(tab)
```

copy_clipboard *Copy data to the clipboard*

Description

Copies a `data.frame`, matrix, 2D or higher array, table, or atomic vector to the system clipboard, ready to paste into a text editor, spreadsheet, or word processor. Wraps `clipr::write_clip()` (a Suggests dependency); requires `clipr` to be installed and a clipboard backend to be available on the platform.

Usage

```
copy_clipboard(
  x,
  row_names_as_col = FALSE,
  row_names = TRUE,
  col_names = TRUE,
  show_message = TRUE,
  quiet = FALSE,
  ...
)
```

Arguments

`x`	A `data.frame`, matrix, 2D array, 3D array, table, or atomic vector to be copied.
`row_names_as_col`	Logical or character. If `FALSE` (the default), row names are not added as a column. If `TRUE`, a column named `"rownames"` is prepended. If a character string, it is used as the column name for the promoted row names. Ignored (with a warning) when `x` is neither a `data.frame` nor a strict matrix.
`row_names`	Logical. If `TRUE` (the default), row names are included in the clipboard output; `FALSE` omits them.
`col_names`	Logical. If `TRUE` (the default), column names are included in the clipboard output; `FALSE` omits them.
`show_message`	Logical. If `TRUE` (the default), prints a success message after copying.
`quiet`	Logical. If `FALSE` (the default), messages are shown. If `TRUE`, suppresses all messages, including the success message, coercion notices, and warnings.
`...`	Additional arguments passed to `clipr::write_clip()`. The pre-0.13.0 dot.case argument names (`row.names.as.col`, `row.names`, `col.names`) are trapped here and raise an error naming their snake_case replacements.

Details

Objects that are not `data.frames` or 2D matrices (atomic vectors, arrays, tables) are automatically coerced to character on the way to the clipboard, as required by `clipr::write_clip()`. The

caller's object is never modified in place; transformations happen on a local copy (see the return value).

Multidimensional arrays (3D and higher) are flattened to a 1D character vector with one element per line. To preserve a tabular layout, extract a 2D slice first, e.g. `copy_clipboard(my_array[, , 1])`.

Messages and warnings raised by the clipboard backend are re-emitted as regular R conditions, so `suppressMessages()` / `suppressWarnings()` work as usual; `quiet = TRUE` silences them all at once.

Value

Invisibly returns the object as it was sent to the clipboard: identical to `x` by default, but reflecting the `row_names_as_col` transformation when one was requested (e.g. a matrix comes back as a `data.frame` with the promoted row-name column). The function is called for its clipboard side effect.

Examples

```
# Writes to the system clipboard, so never run by checks:
## Not run:
if (clipr::clipr_available()) {
  # Data frame
  copy_clipboard(sochealth)

  # Data frame with row names as column
  copy_clipboard(head(sochealth), row_names_as_col = "id")

  # Matrix
  mat <- matrix(1:6, nrow = 2)
  copy_clipboard(mat)

  # Table
  tbl <- table(sochealth$education)
  copy_clipboard(tbl)

  # Array (3D) -- flattened to character
  arr <- array(1:8, dim = c(2, 2, 2))
  copy_clipboard(arr)

  # Recommended: copy 2D slice for tabular layout
  copy_clipboard(arr[, , 1])

  # Numeric vector
  copy_clipboard(c(3.14, 2.71, 1.618))

  # Character vector
  copy_clipboard(c("apple", "banana", "cherry"))

  # Quiet mode (no messages shown)
  copy_clipboard(sochealth, quiet = TRUE)
}
```

```
## End(Not run)
```

---

count_n *Row-wise count of specific or special values*

Description

Counts, for each row of a `data.frame` or `matrix`, how many times one or more values appear across selected columns. Supports type-safe comparison (`allow_coercion = FALSE`), case-insensitive string matching (`ignore_case = TRUE`), and detection of special values (`NA`, `NaN`, `Inf`, `-Inf`) via `special`. Designed to flow inside `dplyr::mutate()` pipelines.

Usage

```
count_n(
  data = NULL,
  select = tidyselect::everything(),
  exclude = NULL,
  count = NULL,
  special = NULL,
  allow_coercion = TRUE,
  ignore_case = FALSE,
  regex = FALSE,
  verbose = FALSE,
  user_na = TRUE
)
```

Arguments

`data`	A `data.frame` or `matrix`. Optional inside `dplyr::mutate()`, where the current data context is used automatically.
`select`	Columns to include. Defaults to `tidyselect::everything()`. Uses tidyselect helpers like `tidyselect::starts_with()`, etc.; a character vector of names is validated with `tidyselect::all_of()`, so unknown names raise an error (as in `mean_n()` and `sum_n()`). If `regex = TRUE`, `select` is treated as a regex string.
`exclude`	Columns to exclude after selection (names or positions, as accepted by `tidyselect::any_of()`). Defaults to `NULL` (no exclusion).
`count`	Value(s) to count. Defaults to `NULL`. Ignored if `special` is used. Multiple values are allowed (e.g., `count = c(1, 2, 3)` or `count = c("yes", "no")`). R automatically coerces all values in `count` to a common type (e.g., `c(2, "2")` becomes `c("2", "2")`), so all values are expected to be of the same final type. If `allow_coercion = FALSE`, matching is type-safe using `identical()`, and the type of `count` must match that of the values in the data. A zero-length `count` (e.g. an upstream filter that emptied it) raises a classed error rather than silently counting nothing, and so does a `count` made only of missing values (use `special = "NA" / "NaN"`).

special	Character vector of special values to count: "NA", "NaN", "Inf", "-Inf", or "all". Defaults to NULL. Every entry is validated, including alongside "all": any other value raises an error, and so does an empty vector (special = character(0) selects nothing to count). "NA" uses is.na(), and therefore includes both NA and NaN values. "NaN" uses is.nan() to match only actual NaN values.
allow_coercion	Logical. If TRUE (the default), values are compared after coercion. If FALSE, uses strict matching via identical().
ignore_case	Logical. If FALSE (the default), comparisons are case-sensitive. If TRUE, performs case-insensitive string comparisons.
regex	Logical. If FALSE (the default), uses tidyselect helpers. If TRUE, interprets select as a regular expression pattern.
verbose	Logical. If FALSE (the default), messages are suppressed. If TRUE, prints processing messages.
user_na	Logical. If TRUE (the default), special = "NA" counts declared missing values together with regular NA (the SPSS MISSING / NMISS convention). If FALSE, only genuine NA / NaN count as missing. The count = path matches the underlying codes in both modes, so explicitly listed declared codes are always counted. See the "Declared missing values" section of freq().

Value

A numeric vector of row-wise counts (unnamed), of length nrow(data). Missing values never match a regular count value, so an all-NA row counts 0 unless special targets missing values. If the selection resolves to zero usable columns, a classed warning (spicy_no_selection) is emitted and NA is returned for all rows, as in mean_n() and sum_n().

Strict matching (allow_coercion = FALSE)

Comparison falls back to identical() when types differ, which also inspects factor levels. Two consequences:

- count = "b" does not match a factor "b" value: pass a factor, e.g. count = factor("b", levels = levels(df$x)).
- Even with a factor count, comparisons against columns whose level set differs will return 0. To guarantee a perfect match (label *and* levels), reuse a value taken from the data itself (e.g. df$x[2]).

Case-insensitive matching (ignore_case = TRUE)

All values are converted to lowercase via tolower() before matching; factor columns are first coerced to character. This mode takes precedence over allow_coercion: equality becomes lowercase string equality, so "b" and "B" match even when allow_coercion = FALSE.

Coercion of count itself

R coerces mixed-type vectors at construction time: count = c(2, "2") becomes c("2", "2") before the function ever sees it. To get type-sensitive matching, keep count homogeneous.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

See Also

`datawizard::row_count()` for a closely related row-wise counter; `count_n()` adds element-wise type-safe matching, multi-value `count`, and special-value detection.

Other row-wise summaries: `mean_n()`, `sum_n()`

Examples

```
library(dplyr)
library(tibble)
library(labelled)

# Basic usage
df <- tibble(
  x = c(1, 2, 2, 3, NA),
  y = c(2, 2, NA, 3, 2),
  z = c("2", "2", "2", "3", "2")
)
count_n(df, count = 2)
count_n(df, count = 2, allow_coercion = FALSE)
df |> mutate(num_twos = count_n(count = 2))

# Mixed types and special values
df <- tibble(
  num   = c(1, 2, NA, -Inf, NaN),
  char  = c("a", "B", "b", "a", NA),
  fact  = factor(c("a", "b", "b", "a", "c")),
  date  = as.Date(c("2023-01-01", "2023-01-01", NA, "2023-01-02", "2023-01-01")),
  lab   = labelled(c(1, 2, 1, 2, NA), labels = c(No = 1, Yes = 2)),
```

```
  logic = c(TRUE, FALSE, NA, TRUE, FALSE)
)
count_n(df, count = 2)
count_n(df, count = "b", ignore_case = TRUE)
count_n(df, count = "a", select = fact)
count_n(df, count = as.Date("2023-01-01"), select = date)

# Count special values
count_n(df, special = "NA")

# Column selection strategies
df <- tibble(
  score_math    = c(1, 2, 2, 3, NA),
  score_science = c(2, 2, NA, 3, 2),
  score_lang    = c("2", "2", "2", "3", "2"),
  name          = c("Jean", "Marie", "Ali", "Zoe", "Nina")
)
count_n(df, select = c(score_math, score_science), count = 2)
count_n(df, select = starts_with("score_"), exclude = "score_lang", count = 2)
count_n(df, select = "^score_", regex = TRUE, count = 2)
df |> mutate(nb_two = count_n(count = 2))

# Strict type-safe matching with factor columns
df <- tibble(
  x = factor(c("a", "b", "c")),
  y = factor(c("b", "B", "a"))
)

# Coercion: character "b" matches both x and y
count_n(df, count = "b")

# Strict match: fails because "b" is character, not factor (returns only 0s)
count_n(df, count = "b", allow_coercion = FALSE)

# Strict match with factor value: works only where levels match
count_n(df, count = factor("b", levels = levels(df$x)), allow_coercion = FALSE)
```

`cramer_v` *Cramér's V*

Description

`cramer_v()` computes Cramér's V for a two-way contingency table, measuring the strength of association between two categorical variables.

Usage

```
cramer_v(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

x	A contingency table (of class table).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

Cramér's V is computed as $V = \sqrt{\chi^2/(n \cdot (k-1))}$, where χ^2 is the Pearson chi-squared statistic, n is the total count, and $k = \min(r, c)$. The point estimate matches the DescTools (Signorell et al., 2024) and SPSS implementations. The confidence interval uses the Fisher z-transformation on V ($\tanh(\operatorname{atanh}(V) \pm z_{\alpha/2}/\sqrt{n-3})$), which differs from the noncentral chi-squared or bootstrap CIs reported by DescTools::CramerV().

Value

When detail = FALSE: a single numeric value (the estimate). When detail = TRUE and conf_level is non-NULL: c(estimate, se, ci_lower, ci_upper, p_value). When detail = TRUE and conf_level = NULL: c(estimate, se, p_value). The p-value tests the null hypothesis of no association (Pearson chi-squared test).

References

Agresti, A. (2002). *Categorical Data Analysis* (2nd ed.). Wiley.

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

Liebetrau, A. M. (1983). *Measures of Association*. Sage.

Signorell, A. et al. (2024). *DescTools: Tools for Descriptive Statistics*. R package.

See Also

phi(), contingency_coef(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), lambda_gk(), phi(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
cramer_v(tab)
cramer_v(tab, detail = TRUE)
```

```
cramer_v(tab, detail = TRUE, conf_level = NULL)
```

`cross_tab` *Cross-tabulation*

Description

Computes a two-way cross-tabulation with optional weights, grouping (including combinations of multiple variables via `interaction()`), row / column percentages, and inferential statistics (Chi-squared test with an APA-style association measure).

Both `x` and `y` are required; for one-way frequency tables, use `freq()`.

Usage

```
cross_tab(
  data,
  x,
  y = NULL,
  by = NULL,
  weights = NULL,
  rescale = FALSE,
  percent = c("none", "column", "row"),
  include_stats = TRUE,
 assoc_measure = c("auto", "cramer_v", "phi", "gamma", "tau_b", "tau_c", "somers_d",
    "lambda", "none"),
  assoc_ci = FALSE,
  correct = FALSE,
  simulate_p = FALSE,
  simulate_B = 2000,
  digits = NULL,
  output = c("default", "data.frame"),
  show_n = TRUE,
  decimal_mark = ".",
  p_digits = 3L,
  user_na = TRUE,
  styled
)
```

Arguments

`data`	A data frame. Alternatively, a vector when using the vector-based interface.
`x`	Row variable (unquoted).
`y`	Column variable (unquoted). Required; the `NULL` default in the signature is a placeholder and triggers an error if left unset (use `freq()` for one-way tables).

by — Optional grouping variable or expression. Can be a single variable or a combination of multiple variables (e.g. `interaction(vs, am)`). A row with a missing by belongs to no group and leaves every table, where a missing `x` or `y` leaves only its own; each note lists by with that count after `x` and `y` ("Missing values removed: x (2), region (3).").

weights — Optional numeric weights. A logical vector is also accepted and coerced to 1/0 (include / exclude).

rescale — Logical. If `FALSE` (the default), weights are used as-is. If `TRUE`, rescales weights so total weighted N matches raw N.

percent — One of `"none"` (the default), `"column"`, or `"row"`. Unique abbreviations are accepted (e.g. `"n"`, `"c"`, `"r"`).

include_stats — Logical. If `TRUE` (the default), computes Chi-squared and an association measure (see `assoc_measure`).

assoc_measure — Character. Which association measure to report. `"auto"` (default) selects Kendall's Tau-b when both variables are ordered factors and Cramér's V otherwise. Other choices: `"cramer_v"`, `"phi"`, `"gamma"`, `"tau_b"`, `"tau_c"`, `"somers_d"`, `"lambda"`, `"none"`.

assoc_ci — Logical. If `TRUE`, includes the 95 percent confidence interval of the association measure in the note. Defaults to `FALSE`.

correct — Logical. If `FALSE` (the default), no continuity correction is applied. If `TRUE`, applies Yates correction (only for 2x2 tables).

simulate_p — Logical. If `FALSE` (the default), uses asymptotic p-values. If `TRUE`, uses Monte Carlo simulation.

simulate_B — Integer. Number of replicates for Monte Carlo simulation. Defaults to `2000`.

digits — Number of decimals for cell values: a single non-negative integer. Defaults to `NULL`, which is resolved to `1` when `percent != "none"` and `0` when `percent = "none"` (counts are integers unless fractional weights are used; raise `digits` to display fractional weighted counts exactly). Same role as `digits` in `freq()`, which formats percentages only and therefore uses a fixed default of `1`. Displayed values round ties half to even (the R / IEC 60559 convention, shared with Stata), so an exact tie like 6.25 prints as `6.2` where SPSS would print `6.3`.

output — Output format. `"default"` (the default) returns a `spicy_cross_table` object (for formatted printing); `"data.frame"` returns a plain `data.frame`. The values match the `output` argument of the `table_*()` family; the rendered engines that family also accepts (`"tinytable"`, `"gt"`, `"flextable"`, ...) are not available in `cross_tab()`.

show_n — Logical. If `TRUE` (the default), adds marginal N totals when `percent != "none"`.

decimal_mark — Character used as the decimal mark in printed numeric values (cells, chi-squared, association estimate, CI bounds, p-value, table note). Either `"."` or `","`. The default follows the language: `options(spicy.language = "fr")` gives the comma here and in `freq()`, exactly as it does in the reporting `table_*()` family. An argument you type wins, so `decimal_mark = "."` under a French language gives French words and a decimal point. Under a comma the p-value keeps its leading zero (`p = 0,659`), the form French typography requires. The resolved mark is frozen on the object when it is built, like every other formatting argument:

a table built under a French language still prints its commas when it is printed later with the language option cleared.

`p_digits` Integer number of decimals used to format the p-value (and to determine the small-p threshold below which `< .001` notation is used). Defaults to `3` (the APA standard); matches the `p_digits` argument of the `table_*()` family.

`user_na` Logical. If `TRUE` (the default), declared missing values in `x`, `y`, or `by` are treated as missing: they are excluded from the table and its statistics like `NA`, and the exclusion is disclosed in the table note, under `Declared missing values removed: ...` for `x` and `y`. Those of `by` are counted with its other missing values, under `Missing values removed: ...`. If `FALSE`, the declared codes tabulate as categories (and, in `by`, define groups). See the "Declared missing values" section of `freq()`.

`styled` Defunct. `styled = TRUE` is now `output = "default"` (the default) and `styled = FALSE` is now `output = "data.frame"`; supplying `styled` is an error.

Value

Depends on `output` and `by`:

- `output = "default"`, no `by`: a `spicy_cross_table` object (a `data.frame` carrying rendering metadata as attributes: `title`, `digits`, `decimal_mark`, `n_row_idx`, `n_col_name`, and the inferential block when `include_stats = TRUE`). Printing dispatches to `print.spicy_cross_table()`.
- `output = "default"`, `by` supplied: a `spicy_cross_table_list`, i.e. a named list of `spicy_cross_table` objects (one element per group level, named by that level). Printing dispatches to `print.spicy_cross_table_list()` which renders each table in turn separated by a blank line.
- `output = "data.frame"`: the same payload returned as a plain `data.frame` (or named list of `data.frames` with `by`), stripped of the `spicy_*` classes and of every metadata attribute (`title`, `note`, `n_total`, `chi2`, `p_value`, `assoc_*`, ...). For programmatic access to the statistics, read the attributes of the default object, e.g. `attr(cross_tab(...), "p_value")`.

Cell columns are the levels of `y`; rows are the levels of `x`. When `percent != "none"`, the `N` column (or `N` row) is added according to `show_n`. When `include_stats = TRUE`, the result carries a Chi-squared row (statistic, df, *p*) and an association-measure row (estimate, optional CI via `assoc_ci`).

Global Options

The function recognizes the following global options that modify its default behavior:

- `options(spicy.percent = "column")` Sets the default percentage mode for all calls to `cross_tab()`. Valid values are `"none"`, `"row"`, and `"column"`. Equivalent to setting `percent = "column"` (or another choice) in each call.
- `options(spicy.simulate_p = TRUE)` Enables Monte Carlo simulation for all Chi-squared tests by default. Equivalent to setting `simulate_p = TRUE` in every call.
- `options(spicy.rescale = TRUE)` Automatically rescales weights so that total weighted N equals the raw N. Equivalent to setting `rescale = TRUE` in each call. Also read by `freq()`, so one option governs both tabulators.

These options are convenient for users who wish to enforce consistent behavior across multiple calls: `spicy.percent` and `spicy.simulate_p` apply to `cross_tab()`, and `spicy.rescale` applies to both `cross_tab()` and `freq()`. They can be disabled or reset by setting them to `NULL`: `options(spicy.percent = NULL, spicy.simulate_p = NULL, spicy.rescale = NULL)`.

Example:

```
options(spicy.simulate_p = TRUE, spicy.rescale = TRUE)
cross_tab(sochealth, smoking, education, weights = weight)
```

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

Examples

```
# Basic crosstab
cross_tab(sochealth, smoking, education)

# Column percentages
cross_tab(sochealth, smoking, education, percent = "column")

# Weighted (rescaled)
cross_tab(sochealth, smoking, education, weights = weight, rescale = TRUE)

# Grouped by sex
cross_tab(sochealth, smoking, education, by = sex)

# Grouped by combination of variables
cross_tab(sochealth, smoking, education, by = interaction(sex, age_group))

# Ordinal variables: auto-selects Kendall's Tau-b
cross_tab(sochealth, education, self_rated_health)
```

```
# 2x2 table with Yates correction
cross_tab(sochealth, smoking, physical_activity, correct = TRUE)

# APA-style p-value precision and European decimal mark
cross_tab(sochealth, smoking, education, decimal_mark = ",", p_digits = 4)
```

`freq` *Frequency table*

Description

Creates a frequency table for a vector or variable from a data frame, with options for weighting, sorting, handling *labelled* data, defining custom missing values, and displaying cumulative percentages.

With `output = "default"`, the function returns a `spicy_freq_table` object that auto-prints as a spicy-formatted ASCII table via `print.spicy_freq_table()` and `spicy_print_table()`; with `output = "data.frame"`, it returns a plain `data.frame` containing frequencies and proportions.

Usage

```
freq(
  data,
  x = NULL,
  weights = NULL,
  digits = 1L,
  valid = TRUE,
  cum = FALSE,
  sort = "",
  na_val = NULL,
  labelled_levels = c("prefixed", "labels", "values"),
  factor_levels = c("observed", "all"),
  rescale = FALSE,
  decimal_mark = ".",
  output = c("default", "data.frame"),
  user_na = TRUE,
  styled
)
```

Arguments

`data`	A `data.frame`, vector, or factor. If a data frame is provided, specify the target variable `x`. If both `data` and `x` are supplied as vectors, `data` is ignored with a warning.
`x`	A variable from `data` (unquoted).

`weights` Optional numeric vector of weights (same length as `x`). A logical vector is also accepted and coerced to 1/0 (include / exclude). The variable may be referenced as a bare name when it belongs to `data`, or as a qualified expression like `other$w` (evaluated in the calling environment), which always takes precedence over `data` lookup. Observations with `NA` weights are dropped from the table with a warning; see `Details`.

`digits` Number of decimal digits to display for percentages (default: `1`). Same role as `digits` in `cross_tab()`, where the `NULL` default resolves to the same `1` decimal whenever percentages are shown. Displayed values round ties half to even (the R / IEC 60559 convention, shared with Stata), so an exact tie like 6.25 prints as `6.2` where SPSS would print `6.3`.

`valid` Logical. If `TRUE` (default), display valid percentages (excluding missing values).

`cum` Logical. If `FALSE` (the default), cumulative percentages are omitted. If `TRUE`, adds cumulative percentages.

`sort` Sorting method for values:

- `""` - the order of the values (default): the levels of a factor or of a labelled variable, numbers in ascending order, text in the C locale (capitals before lower case, the same on every machine) and `FALSE` before `TRUE`, as in `cross_tab()` and the tables
- `"+"` - increasing frequency
- `"-"` - decreasing frequency
- `"name+"` - alphabetical A-Z, in the C locale
- `"name-"` - alphabetical Z-A, in the C locale

For labelled variables displayed with their codes (`labelled_levels "prefixed"` or `"values"`), `"name+"` / `"name-"` sort by the underlying code (so `[10]` follows `[2]`, as in SPSS), not by the display string. With `labelled_levels = "labels"`, labels sort alphabetically.

`na_val` Atomic vector of numeric or character values to be treated as missing (`NA`).

For *labelled* variables (from **haven** or **labelled**), this argument must refer to the underlying coded values, not the visible labels.

Example:

```
x <- labelled(c(1, 2, 3, 1, 2, 3), c("Low" = 1, "Medium" = 2, "High" = 3))
freq(x, na_val = 1) # Treat all "Low" as missing
```

`labelled_levels`

For `labelled` variables, defines how labels and values are displayed:

- `"prefixed"` or `"p"` - show labels as `[value] label` (default)
- `"labels"` or `"l"` - show only labels
- `"values"` or `"v"` - show only numeric codes

`factor_levels` Character. Controls how factor and labelled values are displayed in the frequency table. `"observed"` (the default; matches Stata's `tab` and SPSS `FREQUENCIES`, which both list only values present in the data) shows only levels present in the data. `"all"` (`code_book()`'s default) keeps every declared level, including unused ones, which appear with `n = 0`.

rescale — Logical. If FALSE (the default), weights are used as-is. If TRUE, rescale weights so that their total equals the unweighted sample size (length(weights)). When the argument is not supplied, the default can be set globally with options(spicy.rescale = TRUE), which is read by both freq() and cross_tab(). See Details for the interaction with NA weights.

decimal_mark — Character used as the decimal mark in printed percentages. Either "." or ",". The default follows the language: options(spicy.language = "fr") gives the comma here and in cross_tab(), exactly as it does in the reporting table_*() families. An argument you type wins, so decimal_mark = "." under a French language gives French words and a decimal point. The resolved mark is frozen on the object when it is built: a table built under a French language still prints its commas when it is printed later with the language option cleared. Its words are not – freq() resolves its labels at print time – so that table prints English headings over French numbers.

output — Output format. "default" (the default) returns a spicy_freq_table object that auto-prints as a formatted spicy table; "data.frame" returns a plain data.frame with frequency values. The values match the output argument of the table_*() family; the rendered engines that family also accepts ("tinytable", "gt", "flextable", ...) are not available in freq().

user_na — Logical. If TRUE (the default), declared missing values are treated as missing: each observed declared value becomes its own row of the Missing block (with its value label), and valid percentages exclude those observations. If FALSE, the declaration is ignored and the declared codes tabulate as valid categories. See the "Declared missing values" section.

styled — Defunct. styled = TRUE is now output = "default" (the default) and styled = FALSE is now output = "data.frame"; supplying styled is an error.

Details

Designed to mimic common frequency procedures from SPSS or Stata while integrating the flexibility of R's data structures. The input type (vector, factor, labelled) is auto-detected; see @param labelled_levels and @param factor_levels for the schema-vs-observed level controls, and @param na_val for optional sentinel-value recoding.

Weighting (weights): frequencies and percentages are computed proportionally to the weights. Missing values in weights cause those observations to be dropped from the table entirely (with a warning), matching the behavior of cross_tab() in spicy 0.11.0+. With rescale = TRUE, the remaining (non-NA-weighted) weights are normalized so the total weighted N equals the count of non-NA-weighted rows. With rescale = FALSE, the total weighted N is the actual sum of non-NA weights.

For schema-level inspection without computing frequencies, use varlist(); code_book() documents every variable with the counts of its categories.

Value

With output = "data.frame", a plain data.frame with no extra attributes and columns:

- value - unique values or factor levels

- n - frequency count (weighted if applicable)
- prop - proportion of total
- valid_prop - proportion of valid responses (if valid = TRUE)
- cum_prop, cum_valid_prop - cumulative proportions, between 0 and 1 (if cum = TRUE)

With output = "default" (the default), a spicy_freq_table object: the same data.frame carrying rendering metadata as attributes (digits, data_name, var_name, var_label, class_name, n_total, n_valid, weighted, rescaled, weight_var) used by print.spicy_freq_table(). The object is returned visibly, so a bare freq(...) call auto-prints at the console while f <- freq(...) stays silent (print f to display the table).

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as 8 = Don't know or 9 = Refused that the source file marks as missing while keeping them distinct from a plain NA. Two kinds of declaration exist: na_values / na_range metadata on haven::labelled_spss() vectors, and tagged missing values created by haven::tagged_na() (the Stata .a, .b, ... convention). haven::read_sav() keeps the na_values / na_range declaration only with user_na = TRUE: its default turns the declared codes into NA on import.

spicy honors the declaration by default (user_na = TRUE): declared missing values are excluded from every statistic exactly like NA – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. freq() lists each observed declared value as its own row of the Missing block, with its value label; cross_tab(), table_categorical(), and table_continuous() disclose the exclusion in the table note (Declared missing values remo varlist() counts them as missing in N_valid / NAs / N_distinct while still listing the declared codes in Values; code_book() counts them in n_declared_missing and flags them in its values table.

Every function involved offers the same escape hatch: set user_na = FALSE to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine NAs either way; for them, user_na = FALSE only collapses the per-tag breakdown back into the regular NA count.

See Also

cross_tab() for two-way cross-tabulations; table_categorical() for multi-variable categorical summary tables; varlist() / code_book() for variable inspection; print.spicy_freq_table() for formatted printing; spicy_print_table() for the underlying ASCII rendering engine.

Examples

```
# Frequency table with labelled ordered factor
freq(sochealth, education)
freq(sochealth, self_rated_health, sort = "-")

library(labelled)

# Simple numeric vector
x <- c(1, 2, 2, 3, 3, 3, NA)
freq(x)
```

```
# Plain vector with a sentinel value recoded as missing
freq(c(1, 2, 3, 99, 99), na_val = 99)

# Labelled variable (haven-style)
x_lbl <- labelled(
  c(1, 2, 3, 1, 2, 3, 1, 2, NA),
  labels = c("Low" = 1, "Medium" = 2, "High" = 3)
)
var_label(x_lbl) <- "Satisfaction level"

# Treat value 1 ("Low") as missing
freq(x_lbl, na_val = 1)

# Display only labels, add cumulative %
freq(x_lbl, labelled_levels = "labels", cum = TRUE)

# Display values only, sorted descending
freq(x_lbl, labelled_levels = "values", sort = "-")

# Show all declared factor levels, including unused ones (n = 0).
# The default "observed" mirrors Stata's `tab` and SPSS FREQUENCIES,
# which both drop unused levels.
f <- factor(c("Yes", "No", "Yes"), levels = c("Yes", "No", "Maybe"))
freq(f, factor_levels = "all")

# With weighting
df <- data.frame(
  sex = factor(c("Male", "Female", "Female", "Male", NA, "Female")),
  weight = c(12, 8, 10, 15, 7, 9)
)

# Weighted frequencies (raw weighted counts, the default)
freq(df, sex, weights = weight)

# Weighted frequencies rescaled so the total matches the sample size
freq(df, sex, weights = weight, rescale = TRUE)

# Base R style, with weights and cumulative percentages
freq(df$sex, weights = df$weight, cum = TRUE)

# Piped version (tidy syntax) and sort alphabetically descending ("name-")
df |> freq(sex, sort = "name-")

# European decimal mark (matches `cross_tab()` and the `table_*()` family)
freq(sochealth, education, decimal_mark = ",")

# Plain data.frame return (for programmatic use)
f <- freq(df, sex, output = "data.frame")
head(f)
```

gamma_gk *Goodman-Kruskal Gamma*

Description

`gamma_gk()` computes the Goodman-Kruskal Gamma statistic for a two-way contingency table of ordinal variables.

Usage

```
gamma_gk(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

`x`	A contingency table (of class `table`).
`detail`	Logical. If `FALSE` (default), return the estimate as a numeric scalar. If `TRUE`, return a named numeric vector including confidence interval and p-value.
`conf_level`	A single number strictly between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval. Any other value – including percentages such as `95` – raises a classed error (`spicy_invalid_input`).
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.

Details

Gamma is computed as $\gamma = (C - D)/(C + D)$, where C and D are the numbers of concordant and discordant pairs. It ignores tied pairs, making it appropriate for ordinal variables with many ties. When the asymptotic standard error is zero (e.g. a perfect association), the Wald z-test is undefined and the p-value is `NA`, matching the other measures in the family. Standard error formulas follow the DescTools implementations (Signorell et al., 2024); see `cramer_v()` for full references.

Value

Same structure as `cramer_v()`: a scalar when `detail = FALSE`, a named vector when `detail = TRUE`. The p-value tests H0: gamma = 0 (Wald z-test).

References

Goodman, L. A., & Kruskal, W. H. (1954). Measures of association for cross classifications. *Journal of the American Statistical Association*, 49(268), 732-764. doi:10.2307/2281536

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

See Also

kendall_tau_b(), kendall_tau_c(), somers_d(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), cramer_v(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), lambda_gk(), phi(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$education, sochealth$self_rated_health)
gamma_gk(tab)
gamma_gk(tab, detail = TRUE)
```

---

goodman_kruskal_tau *Goodman-Kruskal's Tau*

Description

goodman_kruskal_tau() computes Goodman-Kruskal's Tau, a proportional reduction in error (PRE) measure for nominal variables.

Usage

```
goodman_kruskal_tau(
  x,
  direction = c("row", "column"),
  detail = FALSE,
  conf_level = 0.95,
  digits = 3L
)
```

Arguments

x	A contingency table (of class table).
direction	Direction of prediction: "row" (default, column predicts row) or "column" (row predicts column).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

Unlike `lambda_gk()`, Goodman-Kruskal's Tau uses all cell frequencies rather than only the modal categories, making it more sensitive to association patterns where lambda may be zero. Goodman-Kruskal's Tau is intrinsically directional and has no canonical symmetric form (unlike `lambda_gk()` or `uncertainty_coef()`); only `"row"` and `"column"` are supported.

Standard error formulas follow the DescTools implementations (Signorell et al., 2024); see `cramer_v()` for full references.

Value

Same structure as `cramer_v()`: a scalar when `detail = FALSE`, a named vector when `detail = TRUE`. The p-value tests H0: tau = 0 (Wald z-test).

References

Goodman, L. A., & Kruskal, W. H. (1954). Measures of association for cross classifications. *Journal of the American Statistical Association*, 49(268), 732-764. doi:10.2307/2281536

See Also

`lambda_gk()`, `uncertainty_coef()`, `assoc_measures()`

Other association measures: `assoc_measures()`, `contingency_coef()`, `cramer_v()`, `gamma_gk()`, `kendall_tau_b()`, `kendall_tau_c()`, `lambda_gk()`, `phi()`, `somers_d()`, `uncertainty_coef()`, `yule_q()`

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
goodman_kruskal_tau(tab)
goodman_kruskal_tau(tab, direction = "column", detail = TRUE)
```

---

inline *Cite a table cell in inline text*

Description

Returns one cell of a spicy table as a character scalar, formatted exactly as the table displays it – same decimals, same *p* style, same interval punctuation, same journal style. Designed for inline R chunks in Quarto / R Markdown:

```
Smokers had higher odds (`r inline(tbl, smoking, "Yes", "or")`).
```

Usage

```
inline(x, variable, level = NULL, column = NULL, model = NULL)
```

Arguments

x	A table returned by table_regression(), table_categorical(), table_continuous(), table_continuous_lm(), table_outcome(), table_categorical_svy(), table_continuous_svy() or table_multiresponse() (default output) – every family as_structured() accepts.
variable	The source variable, unquoted or as a string; or a fit-statistic token ("nobs", "r2", ...) or label ("n").
level	For a factor variable, the level, as a string: a category shown as a row. "(Missing)" addresses the missing-value category by role. A group of by is selected with model.
column	A column token, or a {token} pattern. NULL (the default) returns the estimate-like column of the row when it is unambiguous: the family's primary estimate. That is the coefficient for table_regression() – always token "b": an exponentiated table changes its header to OR, IRR or HR, never its token – the contrast ("delta") or, across a numeric by, the slope ("b") for table_continuous_lm(), the mean ("m") or, on a median-only table, the median ("med", "med_iqr") for table_continuous(), and the count ("n") for table_categorical() and table_multiresponse(). A row carrying none of them refuses and lists its tokens. A table_continuous_lm() row that shows the mean of each group and no contrast (a by of three or more levels, or contrast = "none") has no default: it refuses and gives the address of a mean.
model	A group of by, or in a multi-model table the model: its label (as displayed) or its position. See *Groups and models*.

Value

A character scalar.

Addressing

The row is found by **identity**, not by display text: variable names the source column (.variable in the typed body), level the level (.level). Custom labels, a style, or a translated display never change the call. As a convenience, a variable that matches no source column is looked up among the displayed labels before erroring. The missing-value category is addressed by level = "(Missing)" whatever its displayed (possibly translated or relabeled) label, through its row role, and the Total row of table_categorical() under total_row = TRUE by level = "Total", the same way. Fit statistics are addressed by their token as variable (inline(tbl, "nobs"), inline(tbl, "r2")) or by their label (inline(tbl, "n")). The token is the one show_fit_stats takes ("r2", "adj_r2", "aic", ...), the key of the row in the variable column of output = "data.frame". glance() and the long output name the same statistic as broom does (r.squared, adj.r.squared), a name inline() does not read. Identity comes first: a source column wins over a token or a label of the same name, and an exact label over a fit statistic's label read regardless of case. A variable that still matches the rows of two variables or statistics is refused, and the error names an address for each.

A statistic that belongs to a whole variable rather than to one of its levels sits on the variable's own row: the *p* of a table_categorical() block, its association measure, its SMD. Leave level out to cite it (inline(tbl, smoking, column = "p")). The same call cites the *p* of a by-group

table_continuous() or table_continuous_svy(), which sits on the first group's row: when one row of the variable alone carries the statistic, that row is the address.

Groups and models

model selects a group of by in every table, by its label or its position, wherever the table sets the groups out: the column spanners of table_categorical(), table_categorical_svy() and table_multiresponse(), the margin included (inline(tbl, smoking, "Yes", "pct", model = "Female")), the per-group columns of table_continuous_lm(), headed Female and Male under the mean symbol M (inline(tbl, bmi, column = "emmean", model = "Female")), and the rows of table_continuous() and table_continuous_svy(), one per group (inline(tbl, bmi, model = "Female")). The group of missing by values is model = "(Missing)". In a multi-model table, model selects the model the same way. The cells that belong to no group (a *p*, the contrast of table_continuous_lm() and its interval, an association measure) are cited without model. In table_continuous() and table_continuous_svy(), the test, its *p*, the effect size and the SMD are statistics of the variable, printed once on the row of the first group: inline(tbl, bmi, column = "p"). Asked with model, whichever group it names, they are an error (class spicy_invalid_input) that gives that call.

level names a category shown as a row: the level of a variable, its row of missing values. Until spicy 0.14.0 it also selected the groups of table_continuous(), table_continuous_svy() and table_continuous_lm(): that use is an error (class spicy_defunct) whose message gives the call to use instead.

table_multiresponse() has one row per item, addressed by its column name, then the rows of none_row, addressed as "none" and "no_data", and the base row, the number of respondents, addressed as the fit statistic "nobs": inline(tbl, "nobs", model = "Total"). The item p and statistic belong to no group and are cited without model (inline(tbl, info_internet, column = "p")).

Columns

The column is a **token** of the typed contract ("b", "se", "p", "ci", "or", "ame", "n", "pct", "m", ... – see as_structured()'s col_meta), never a display header. "ci" composes the interval with the style's brackets and separator, and so does every other interval token the table carries ("med_ci", "ame_ci", "assoc_ci"): each addresses its own bounds. "partial_chi2" cites the statistic alone, without the degrees of freedom the table prints beside it.

Patterns

A column containing { is a pattern: each {token} is replaced by the corresponding cell, so one call quotes a full sentence fragment:

```
inline(tbl, smoking, "Yes", "{or} ({ci_label} {ci}; p = {p})")
```

{ci_label} inserts the interval label of the interval the pattern cites (95% CI, or Mdn 95% CI in a pattern quoting {med_ci}) – the first one when it cites several, the table's first when it cites none. Note that {p} carries the floor operator when the table does (<.001), so write p {p} rather than p = {p} in patterns that may hit the floor.

Errors

Every misaddressing is a classed error that lists the available choices: unknown variables list the variables, missing levels list the levels, unknown tokens list the table's tokens, ambiguous models and groups list them. A `level` that names a group of a by table points to `model`. A cell the table itself displays as undefined (an aliased coefficient's en-dash) refuses with the reason rather than pasting a dash into a sentence.

See Also

`as_structured()` for the typed contract behind the addressing.

Examples

```
fit <- lm(wellbeing_score ~ age + sex, data = sochealth)
tbl <- table_regression(fit)
inline(tbl, age, column = "b")
inline(tbl, sex, "Male", "{b} ({ci_label} {ci}; p {p})")
inline(tbl, "n")

# A group of `by` is a `model`, whether the table sets it out as
# columns or as rows.
ct <- table_categorical(sochealth, select = smoking, by = sex)
inline(ct, smoking, "Yes", "pct", model = "Female")
cn <- table_continuous(sochealth, select = bmi, by = sex)
inline(cn, bmi, model = "Female")
inline(cn, bmi, column = "p")
```

---

kendall_tau_b *Kendall's Tau-b*

Description

`kendall_tau_b()` computes Kendall's Tau-b for a two-way contingency table of ordinal variables.

Usage

```
kendall_tau_b(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

`x`	A contingency table (of class `table`).
`detail`	Logical. If `FALSE` (default), return the estimate as a numeric scalar. If `TRUE`, return a named numeric vector including confidence interval and p-value.
`conf_level`	A single number strictly between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval. Any other value – including percentages such as `95` – raises a classed error (`spicy_invalid_input`).
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.

Details

Kendall's Tau-b is computed as $\tau_b = (C - D)/\sqrt{(n_0 - n_1)(n_0 - n_2)}$, where $n_0 = n(n-1)/2$, n_1 is the number of pairs tied on the row variable, and n_2 is the number tied on the column variable. Tau-b corrects for ties and is appropriate for square tables. When the asymptotic standard error is zero (e.g. a perfect association), the Wald z-test is undefined and the p-value is `NA`, matching the other measures in the family.

The asymptotic standard error is the Brown and Benedetti (1977) ASE1, as printed by SPSS / PSPP `CROSSTABS`. It deliberately diverges from `DescTools::KendallTauB()`, whose implementation mis-scales one margin term of the gradient; see `cramer_v()` for full references.

Value

Same structure as `cramer_v()`: a scalar when `detail = FALSE`, a named vector when `detail = TRUE`. The p-value tests H0: tau-b = 0 (Wald z-test).

References

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, 30(1-2), 81-93. doi:10.2307/2332226

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

See Also

`kendall_tau_c()`, `gamma_gk()`, `somers_d()`, `assoc_measures()`

Other association measures: `assoc_measures()`, `contingency_coef()`, `cramer_v()`, `gamma_gk()`, `goodman_kruskal_tau()`, `kendall_tau_c()`, `lambda_gk()`, `phi()`, `somers_d()`, `uncertainty_coef()`, `yule_q()`

Examples

```
tab <- table(sochealth$education, sochealth$self_rated_health)
kendall_tau_b(tab)
```

---

kendall_tau_c *Kendall's Tau-c (Stuart's Tau-c)*

Description

`kendall_tau_c()` computes Stuart's Tau-c (also known as Kendall's Tau-c) for a two-way contingency table of ordinal variables.

Usage

```
kendall_tau_c(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

x	A contingency table (of class table).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

Stuart's Tau-c is computed as $\tau_c = 2m(C - D)/(n^2(m-1))$, where $m = \min(r, c)$. It is designed for rectangular tables; the estimate is bounded by $[-1, 1]$ only when the table is square, and may fall outside that range otherwise. When the asymptotic standard error is zero (e.g. a perfect association), the Wald z-test is undefined and the p-value is NA, matching the other measures in the family. When one variable is constant (all observations in a single row or column), there are no untied pairs and the statistic degenerates to a meaningless 0: the function returns NA with a spicy_undefined_stat warning, like its siblings, matching the SPSS / PSPP behavior of reporting no value. Standard error formulas follow the DescTools implementations (Signorell et al., 2024); see cramer_v() for full references.

Value

Same structure as cramer_v(): a scalar when detail = FALSE, a named vector when detail = TRUE. The p-value tests H0: tau-c = 0 (Wald z-test).

References

Stuart, A. (1953). The estimation and comparison of strengths of association in contingency tables. *Biometrika*, 40(1-2), 105-110. doi:10.2307/2333101

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

See Also

kendall_tau_b(), gamma_gk(), somers_d(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), cramer_v(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), lambda_gk(), phi(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$education, sochealth$self_rated_health)
kendall_tau_c(tab)
```

`label_from_names` *Derive variable labels from column names* `name<sep>label`

Description

Splits each column name at the **first** occurrence of `sep`, renames the column to the part before `sep` (the *name*, trimmed of surrounding whitespace), and assigns the part after `sep` as a `"label"` attribute on the column. The label attribute follows the haven convention also used by `labelled::var_label()`, so labelled-aware tooling (`labelled`, `haven`, `varlist()`, `code_book()`, ...) reads it transparently. Splitting at the *first* `sep` means the label itself may contain the separator.

Usage

```
label_from_names(df, sep = ". ")
```

Arguments

`df`	A `data.frame` or tibble with column names of the form `"name<sep>label"` (e.g. `"code. question text"`).
`sep`	Character string used as separator between name and label. Default `". "` (LimeSurvey's default); any literal string can be used. Matched as a fixed string, so regex metacharacters such as `.` or `|` carry no special meaning.

Details

Designed primarily for **LimeSurvey CSV exports** with *Headings: Question code & question text*, which produce column names like `"code. question text"`. The default separator `". "` matches that export.

LimeSurvey question codes (the part *before* `sep`) are restricted to alphanumerics, must start with a letter, and contain no spaces – so the column name has to carry both the code and the question text. If your export uses *Headings: Question code* (codes only), re-export with *Question code & question text* before calling this function; there is no way to recover a label from a code alone.

Whitespace handling: the **name** (left of `sep`) is trimmed of surrounding whitespace, because R column names are intended to be referenced bare (without backticks) and leading / trailing whitespace would force quoting throughout the user's downstream code. The **label** (right of `sep`) is preserved verbatim, following the Stata / SPSS convention that variable labels are faithful user content – spicy does not silently mutate label strings. To trim labels yourself, post-process with `labelled::var_label(df) <- lapply(labelled::var_label(df), trimws)`.

Value

An object of the **same class as** `df` – a base `data.frame` if `df` was a base `data.frame`, a `tbl_df` if `df` was a tibble. The output has column names equal to the trimmed names (before `sep`) and, for every column whose original name contained `sep`, a `"label"` attribute equal to the label (after `sep`). Columns whose name does not contain `sep` are passed through unchanged with no label attached.

Errors

The function raises an actionable error – rather than letting the downstream constructor raise a cryptic one – when the split produces:

- duplicate column names that the renaming itself creates (two original names share the same prefix before `sep`, or a new name collides with an existing one). Names that were already duplicated in the input (`check.names = FALSE` data) are passed through untouched, not blamed on the split; or
- an empty column name (the original name starts with `sep` and has nothing before it).

See Also

`labelled::var_label()` reads the `"label"` attribute set by this function; `varlist()` and `code_book()` surface it in their inspection outputs.

Other variable inspection: `code_book()`, `varlist()`

Examples

```
# LimeSurvey-style column names (default sep = ". ").
df <- data.frame(
  "age. Age of respondent" = c(25, 30),
  "score. Total score. Manually computed." = c(12, 14),
  check.names = FALSE
)
out <- label_from_names(df)
attr(out$age, "label")
attr(out$score, "label")

# Custom separator.
df2 <- data.frame(
  "id|Identifier" = 1:3,
  "score|Total score" = c(10, 20, 30),
  check.names = FALSE
)
out2 <- label_from_names(df2, sep = "|")
```

`lambda_gk` *Goodman-Kruskal's Lambda*

Description

`lambda_gk()` computes Goodman-Kruskal's Lambda, a proportional reduction in error (PRE) measure for nominal variables.

Usage

```
lambda_gk(
  x,
  direction = c("symmetric", "row", "column"),
  detail = FALSE,
  conf_level = 0.95,
  digits = 3L
)
```

Arguments

x	A contingency table (of class table).
direction	Direction of prediction: "symmetric" (default), "row" (column predicts row), or "column" (row predicts column).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

Lambda measures how much prediction error is reduced when the independent variable is used to predict the dependent variable. It ranges from 0 (no reduction) to 1 (perfect prediction). Lambda can equal zero even when variables are associated if the modal category dominates in every column (or row).

The default direction = "symmetric" follows the SPSS and DescTools convention: symmetric lambda is a standard, well-defined variant with its own asymptotic standard error. somers_d() deliberately differs (its default is "row") because its symmetric form is a derived quantity without an analytic SE; see its documentation.

Standard error formulas follow the DescTools implementations (Signorell et al., 2024); see cramer_v() for full references.

Value

Same structure as cramer_v(): a scalar when detail = FALSE, a named vector when detail = TRUE. The p-value tests H0: lambda = 0 (Wald z-test).

References

Goodman, L. A., & Kruskal, W. H. (1954). Measures of association for cross classifications. *Journal of the American Statistical Association*, 49(268), 732-764. doi:10.2307/2281536

See Also

goodman_kruskal_tau(), uncertainty_coef(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), cramer_v(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), phi(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
lambda_gk(tab)
lambda_gk(tab, direction = "row")
lambda_gk(tab, direction = "column", detail = TRUE)
```

---

mean_n *Row means with an optional minimum-valid-values rule*

Description

Computes row-wise means across selected numeric columns of a data.frame or matrix. Missing values are handled per row via min_valid (an integer count or proportion of non-NA values required); rows that fail the rule return NA, and rows with no valid values at all return NA even when min_valid = 0. Non-numeric columns are dropped silently (set verbose = TRUE to see which). Designed to flow inside dplyr::mutate(): when called without an explicit data argument, the current data context is used.

Usage

```
mean_n(
  data = NULL,
  select = tidyselect::everything(),
  exclude = NULL,
  min_valid = NULL,
  digits = NULL,
  regex = FALSE,
  verbose = FALSE,
  user_na = TRUE
)
```

Arguments

data	A data.frame or matrix. Optional inside dplyr::mutate(), where the current grouping/data context is used automatically.
select	Columns to include. If regex = FALSE, use tidyselect syntax (default: tidyselect::everything()). If regex = TRUE, provide a regular expression pattern (character string).
exclude	Columns to exclude (default: NULL).

`min_valid` Minimum number of valid (non-NA) values required per row. Accepts:

- NULL (the default) – every selected column must be valid.
- a proportion in `(0, 1)` – `round(ncol(x) * min_valid)` valid columns required (e.g. `min_valid = 0.5` requires at least half of the selected columns to be non-NA).
- a non-negative integer count up to the number of selected numeric columns.

Non-integer values `>= 1` (e.g. `1.5`) and counts greater than `ncol(x)` raise an actionable error.

Rows with zero valid values always return NA, even when `min_valid = 0`: an empty row-wise summary is undefined, so the raw `rowMeans()` / `rowSums()` identities (`NaN` / `0`) are never returned.

`digits` Optional non-negative integer giving the number of decimal places to round the result to. Defaults to NULL (no rounding).

`regex` Logical. If FALSE (the default), uses tidyselect helpers. If TRUE, the `select` argument is treated as a regular expression.

`verbose` Logical. If FALSE (the default), messages are suppressed. If TRUE, prints a message about non-numeric columns excluded.

`user_na` Logical. If TRUE (the default), declared missing values count as missing – both in the computed summary and in the `min_valid` valid-count gate. If FALSE, the declared codes are treated as ordinary numbers. See the "Declared missing values" section of `freq()`.

Value

A numeric vector of row-wise means.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain NA. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into NA on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like NA – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`
`varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine NAs either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular NA count.

See Also

Other row-wise summaries: count_n(), sum_n()

Examples

```
library(dplyr)

# Create a simple numeric data frame
df <- tibble(
  var1 = c(10, NA, 30, 40, 50),
  var2 = c(5, NA, 15, NA, 25),
  var3 = c(NA, 30, 20, 50, 10)
)

# Compute row-wise mean (all values must be valid by default)
mean_n(df)

# Require at least 2 valid (non-NA) values per row
mean_n(df, min_valid = 2)

# Require at least 50% valid (non-NA) values per row
mean_n(df, min_valid = 0.5)

# Round the result to 1 decimal
mean_n(df, digits = 1)

# Select specific columns
mean_n(df, select = c(var1, var2))

# Select specific columns using a pipe
df |>
  select(var1, var2) |>
  mean_n()

# Exclude a column
mean_n(df, exclude = "var3")

# Select columns ending with "1"
mean_n(df, select = ends_with("1"))

# Use with native pipe
df |> mean_n(select = starts_with("var"))

# Use inside dplyr::mutate()
df |> mutate(mean_score = mean_n(min_valid = 2))

# Select columns directly inside mutate()
df |> mutate(mean_score = mean_n(select = c(var1, var2), min_valid = 1))

# Select columns before mutate
df |>
  select(var1, var2) |>
```

```
  mutate(mean_score = mean_n(min_valid = 1))

# Show verbose processing info
df |> mutate(mean_score = mean_n(min_valid = 2, digits = 1, verbose = TRUE))

# Add character and grouping columns
df_mixed <- mutate(df,
  name = letters[1:5],
  group = c("A", "A", "B", "B", "A")
)
df_mixed

# Non-numeric columns are ignored
mean_n(df_mixed)

# Use within mutate() on mixed data
df_mixed |> mutate(mean_score = mean_n(select = starts_with("var")))

# Use everything() but exclude non-numeric columns manually
mean_n(df_mixed, select = everything(), exclude = "group")

# Select columns using regex
mean_n(df_mixed, select = "^var", regex = TRUE)
mean_n(df_mixed, select = "ar", regex = TRUE)

# Apply to a subset of rows (first 3)
df_mixed[1:3, ] |> mean_n(select = starts_with("var"))

# Store the result in a new column
df_mixed$mean_score <- mean_n(df_mixed, select = starts_with("var"))
df_mixed

# With a numeric matrix
mat <- matrix(c(1, 2, NA, 4, 5, NA, 7, 8, 9), nrow = 3, byrow = TRUE)
mat
mat |> mean_n(min_valid = 2)
```

---

phi *Phi coefficient*

Description

`phi()` computes the phi coefficient for a 2x2 contingency table.

Usage

```
phi(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

x	A contingency table (of class table).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

The phi coefficient is $\phi = \sqrt{\chi^2/n}$. It is equivalent to Cramér's V for 2x2 tables and equals the absolute value of the Pearson correlation between the two binary variables – spicy returns only the magnitude (always non-negative), matching the DescTools (Signorell et al., 2024) and PSPP conventions. SPSS itself SIGNS phi on 2x2 tables (its CROSSTABS algorithm sets the sign to that of the Pearson correlation), so SPSS output can show a negative value of the same magnitude. To recover the signed direction of the 2x2 association, compute the Pearson correlation directly (e.g. cor(x, y) after coding both variables 0/1).

The confidence interval uses the Fisher z-transformation on ϕ; see cramer_v() for the formula and full references.

Value

Same structure as cramer_v(): a scalar when detail = FALSE, a named vector when detail = TRUE. The p-value tests the null hypothesis of no association (Pearson chi-squared test).

References

Liebetrau, A. M. (1983). *Measures of Association*. Sage.

See Also

cramer_v(), yule_q(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), cramer_v(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), lambda_gk(), somers_d(), uncertainty_coef(), yule_q()

Examples

```
tab <- table(sochealth$smoking, sochealth$sex)
phi(tab)
phi(tab, detail = TRUE)
```

`sochealth` *Simulated social-health survey*

Description

A simulated dataset of 1200 respondents from a fictional social-health survey, designed to illustrate the main features of the spicy package: variable labels, ordered factors, survey weights, association measures, a multiple-response question, and publication-ready reporting.

Usage

```
sochealth
```

Format

A tibble with 1200 rows and 29 variables:

sex Factor. Sex of the respondent.

age Numeric. Age in years (25–75).

age_group Ordered factor. Age group (25–34, 35–49, 50–64, 65–75).

education Ordered factor. Highest education level (Lower secondary, Upper secondary, Tertiary).

social_class Ordered factor. Subjective social class (Lower, Working, Lower middle, Middle, Upper middle).

region Factor. Region of residence (6 regions).

employment_status Factor. Employment status (Employed, Student, Unemployed, Inactive).

income_group Ordered factor. Household income group (Low, Lower middle, Upper middle, High). Contains missing values.

income Numeric. Monthly household income in CHF (1000–7400).

smoking Factor. Current smoker (No, Yes). Contains missing values.

physical_activity Factor. Regular physical activity (No, Yes).

dentist_12m Factor. Dentist visit in the last 12 months (No, Yes).

info_doctor Logical. Source of health information in the past 12 months: doctor or other health professional (`TRUE` = chosen). Missing for the 36 respondents who skipped the question; see `Details`.

info_pharmacist Logical. Source of health information: pharmacist. Missing as `info_doctor`.

info_internet Logical. Source of health information: internet, websites or apps. Missing as `info_doctor`.

info_family Logical. Source of health information: family or friends. Missing as `info_doctor`.

info_media Logical. Source of health information: television, radio or newspapers. Missing as `info_doctor`.

self_rated_health Ordered factor. Self-rated health (Poor, Fair, Good, Very good). Contains missing values.

wellbeing_score Numeric. WHO-5 wellbeing index (0–100).

bmi Numeric. Body mass index in kg/m^2 (16–39). Contains missing values.

bmi_category Ordered factor. BMI category (Normal weight, Overweight, Obesity). Contains missing values.

institutional_trust Ordered factor. Trust in institutions (Very low, Low, High, Very high).

political_position Numeric. Political position on a 0 (left) to 10 (right) scale. Contains missing values.

life_sat_health Integer. Satisfaction with own health (1–5 Likert scale). Contains missing values.

life_sat_work Integer. Satisfaction with work or main activity (1–5 Likert scale). Contains missing values.

life_sat_relationships Integer. Satisfaction with personal relationships (1–5 Likert scale). Contains missing values.

life_sat_standard Integer. Satisfaction with standard of living (1–5 Likert scale). Contains missing values.

response_date POSIXct. Date and time of survey response (September–November 2024).

weight Numeric. Survey design weight (range 0.29–3.45); calibrated: `mean(weight)` is about 1 (0.997) and `sum(weight)` is 1196.5 for the 1200 rows. See `Details`.

Details

Every variable carries a `"label"` attribute (read by `labelled::var_label()` and surfaced by `varlist()` / `code_book()`). The mix of factor types is deliberate: nominal factors (`sex`, `region`, ...) and ordered factors (`education`, `self_rated_health`, ...) live side by side so that `cross_tab()` and `table_categorical()` can demonstrate the automatic ordinal-vs-nominal dispatch (Cramér's V, Phi, Kendall's Tau-b, Goodman-Kruskal Gamma) on the same dataset.

Survey weights (`weight`) are calibrated: they average about 1 (`mean(weight)` is 0.997) and sum to 1196.5 for the 1200 rows. Weighted means therefore agree with unweighted means up to sampling noise without further rescaling.

The five `info_*` variables answer one multiple-response question, "Sources of health information used in the past 12 months (check all that apply)", the example data of `table_multiresponse()`. Each item is logical: `TRUE` when the source was chosen, `FALSE` when it was not, and `NA` on all five items for the 36 respondents who skipped the question. A respondent who answered without choosing any source is `FALSE` on all five (67 of the 1164 who answered). Like the rest of the data, the block is simulated: doctor and media rise with age, internet falls with age and rises with education, family is more frequent among women, and the pharmacist item depends on no other variable.

Source

Simulated data for illustration purposes; reproducible by sourcing `data-raw/sochealth.R`. The script seeds the main generation block with `set.seed(2025)`, the two missing-value injection blocks with `set.seed(2027)` (the four `life_sat_*` items) and `set.seed(2026)` (`smoking`, `self_rated_health`, `income_group`, `political_position`, `bmi`), and the multiple-response block (the five `info_*` items) with `set.seed(2030)`.

Examples

```
data(sochealth)
varlist(sochealth)
freq(sochealth, education)
cross_tab(sochealth, education, self_rated_health)
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  by = age_group
)
```

---

somers_d *Somers' D*

Description

somers_d() computes Somers' D for a two-way contingency table of ordinal variables.

Usage

```
somers_d(
  x,
  direction = c("row", "column", "symmetric"),
  detail = FALSE,
  conf_level = 0.95,
  digits = 3L
)
```

Arguments

x	A contingency table (of class table).
direction	Direction of prediction: "row" (default, column predicts row), "column" (row predicts column), or "symmetric" (harmonic mean of both directions).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

Somers' D is an asymmetric ordinal measure defined as $d = (C - D)/(C + D + T)$, where T is the number of pairs tied on the independent variable. The symmetric version (`direction = "symmetric"`) is the *harmonic* mean of the two asymmetric values, matching the SPSS / PSPP convention; this is **not** identical to Kendall's Tau-b (which is the *geometric* mean of the same two quantities), although the two often agree to two decimals. It is computed via the equivalent closed form $2(C - D)$ divided by the sum of the two asymmetric denominators, so a table with exactly as many concordant as discordant pairs (e.g. an independence pattern) yields the well-defined value 0 – the harmonic-mean form is 0/0 there – as printed by SPSS / PSPP. The symmetric estimate is `NA` only when one of the asymmetric directions is itself undefined (with the same `spicy_undefined_stat` warning). No analytic SE / CI is reported for the symmetric form: its `se` is always `NA`, matching PSPP, which prints no ASE for it (DescTools offers no symmetric form at all).

The default `direction = "row"` differs deliberately from `lambda_gk()` and `uncertainty_coef()` (which default to `"symmetric"`): Somers' D is intrinsically asymmetric, its symmetric form is a derived convention without inference, and the reference implementation (`DescTools::SomersDelta()`) offers only the two asymmetric directions, defaulting to `"row"`.

Standard error formulas for the asymmetric directions follow the DescTools implementations (Signorell et al., 2024); see `cramer_v()` for full references.

Value

Same structure as `cramer_v()`: a scalar when `detail = FALSE`, a named vector when `detail = TRUE`. The p-value tests H0: D = 0 (Wald z-test).

References

Somers, R. H. (1962). A new asymmetric measure of association for ordinal variables. *American Sociological Review*, 27(6), 799-811. doi:10.2307/2090408

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association*, 72(358), 309-315. doi:10.1080/01621459.1977.10480995

See Also

`kendall_tau_b()`, `gamma_gk()`, `assoc_measures()`

Other association measures: `assoc_measures()`, `contingency_coef()`, `cramer_v()`, `gamma_gk()`, `goodman_kruskal_tau()`, `kendall_tau_b()`, `kendall_tau_c()`, `lambda_gk()`, `phi()`, `uncertainty_coef()`, `yule_q()`

Examples

```
tab <- table(sochealth$education, sochealth$self_rated_health)
somers_d(tab, direction = "row")
somers_d(tab, direction = "column", detail = TRUE)
```

`spicy_labels` *Table labels and their language*

Description

Every string a reader of a spicy table sees – column headers, row labels, titles, table footnotes – is held under a stable key, and `spicy_labels()` returns those keys with the label each one currently resolves to. It is the companion of the two options that move them.

Usage

```
spicy_labels(language = NULL)
```

Arguments

`language`	A language name (`"en"`, `"fr"`), or `NULL` (the default) to report the labels in force. Any `options(spicy.labels)` override applies either way.

Value

A named character vector, one element per key, in registry order.

Global options

- `options(spicy.language = "fr")` The language of the table, for the whole document – its labels and the typography its numbers are written in. Two sets ship: `"en"` (the default) and `"fr"`. The language of a report is a property of the report, so this is set once in a setup chunk rather than passed to each table.
- `options(spicy.labels = list(<key> = "<label>"))` A per-label override, for the case where one word has to change and a language does not: a named list (or named character vector) whose names are keys of `spicy_labels()`. An unknown key is an error.
- `options(spicy.var_labels = TRUE)` How the descriptive tables (`table_categorical()`, `table_continuous()`, `table_continuous_lm()`, `table_outcome()` and the survey twins) and the regression tables (`table_regression()`, `table_regression_uv()`) name your variables: `TRUE` shows each variable's `label` attribute, `FALSE` (the default) its name. It is the default of their `labels` argument, and the variables a named `labels` vector leaves out follow it. A `TRUE`, a `FALSE` or a `var_labels()` typed in the call wins over it. Any other value is an error.

A label resolves through `spicy.labels`, then the `spicy.language` set, then English. A set carries only the keys it translates, so anything it does not name falls back to English rather than erroring or coming out blank. All three options are cleared with `NULL`.

What a language does not change

Only DISPLAY strings translate. The column names of the exported frames (`as.data.frame()`, `tidy()`, `as_structured()`) are a documented contract that user code indexes into, so `out[["Yes %"]]` resolves under every language, and so do the block identities, the encoded cell tokens and the mathematical glyphs. Errors, warnings and messages stay English: they are read by developers and quoted in bug reports.

One exception, and it follows from the same rule. A column named after a LEVEL of `by` takes that level's own spelling – which is why `"Yes %"` stays `"Yes %"`: your data is never translated. spicy's own missing category is such a level, so it is the one name that does move: `table_categorical(by = )` on a variable with missing values gives `"(Missing) n"` in English and `"(Manquant) n"` under `"fr"` (or whatever `row_missing_level` is overridden to). Address that column through `row_missing_level` rather than by typing it.

Numbers follow the language too

A language brings its typographic locale with it, so `options(spicy.language = "fr")` is one gesture for a coherent French report table: French words, comma decimal mark, and the leading zero French typography keeps on a p-value (`0,003`). The sources are the BIPM's SI brochure and the European Union's *Code de redaction interinstitutionnel*; `spicy_style()` quotes them. `"en"` brings no locale, and nothing changes for anyone who sets no language.

The locale rides the style layer, so it reaches the reporting families – `table_regression()`, `table_categorical()`, `table_continuous()`, `table_continuous_lm()`, `table_outcome()` and the survey twins. The language reaches every table, the exploration pair included: `freq()` and `cross_tab()` have no style layer, so it sets the DEFAULT of their `decimal_mark` – the one typographic lever they carry – and an argument you type wins over it. Under a comma their p-value keeps its leading zero (`p = 0,659`), the form French typography requires.

The locale sits at the BOTTOM of the formatting resolution. A journal style outranks it – `style = "jama"` under `"fr"` gives JAMA's p-values and the French comma, `style = "lancet"` keeps the journal's midline decimal point – and an argument you type outranks both, so `decimal_mark = "."` gives French words with a decimal point (the p-value keeps its locale zero; `spicy_style()` documents that lever).

Two timing notes. Figures are frozen when a table is BUILT, like every formatting argument, while words resolve when it is printed – so set the language once, at the top of the document. And a comma mark makes an Excel export write its body as text rather than live numbers, since Excel would otherwise re-punctuate a numeric cell with the viewer's own locale.

See Also

`spicy_style()` for the journal styles, which outrank the language's own typography.

Examples

```
head(spicy_labels())

# The same keys in French.
fr <- spicy_labels("fr")
fr[["header_mean"]]
fr[["row_missing_level"]]
```

```
# One label, not a language: the missing CATEGORY of the grouping
# variable is a refusal to answer here, not an absent value.
old <- options(spicy.labels = list(row_missing_level = "(No answer)"))
table_categorical(sochealth, select = sex, by = smoking)
options(old)
```

`spicy_style` *Build or select a table style*

Description

A *style* is a small set of number-formatting rules – how many decimals a p-value gets, where it bottoms out, whether it keeps its leading zero, what the decimal mark is, how a confidence interval is written – with the word that opens the note under the table and the abbreviations that note defines. `spicy_style()` composes one by hand; the named themes (`"jama"`, `"nejm"`, `"lancet"`, `"annals"`, `"apa"`, `"aer"` – `spicy_style_names()` returns the list) are pre-composed ones, each encoding rules taken verbatim from an official document of the institution.

Usage

```
spicy_style(
  base = NULL,
  ...,
  p_style = NULL,
  p_digits = NULL,
  p_floor = NULL,
  p_bands = NULL,
  p_sigfig = NULL,
  decimal_mark = NULL,
  ci_sep = NULL,
  ci_brackets = NULL,
  stars = NULL,
  digits = NULL,
  effect_size_digits = NULL,
  fit_digits = NULL,
  ic_digits = NULL,
  percent_digits = NULL,
  note_label = NULL,
  abbreviations = NULL,
  minus_sign = NULL,
  v_digits
)

spicy_style_names()
```

Arguments

base	Optional theme to start from – a theme name or another spicy_style – whose levers the arguments below override. This is the composition escape hatch: spicy_style("lancet", ci_sep = " to ") is The Lancet's rules with one of them changed. The theme's provenance travels with the result and names the levers you overrode, so a modified theme never passes for the theme itself.
...	Must be empty. Any argument landing here is a misspelled lever and raises an error rather than being ignored.
p_style	How p-values carry their leading zero: "apa" drops it (.003), "standard" keeps it (0.003). NULL leaves spicy's default, which follows the mark: dropped under a point, kept under a comma.
p_digits	Decimal places for p-values (a positive integer). Under p_sigfig it acts as the decimal cap instead.
p_floor	The value below which a p-value prints as <floor rather than exactly, e.g. 0.001 for <.001. A number strictly between 0 and 1. NULL uses 10^-p_digits.
p_bands	Decimal places that vary with the size of the p-value: a list of two-element numeric vectors c(cutoff, digits), cutoffs strictly increasing, the last one Inf. list(c(0.01, 3), c(Inf, 2)) reads as "three decimals below .01, two otherwise". The comparison is p < cutoff. Mutually exclusive with p_sigfig.
p_sigfig	Significant figures for p-values (a positive integer), capped at p_digits decimal places. Mutually exclusive with p_bands.
decimal_mark	A single character; the mark between the integer and the decimal part of every number in the table.
ci_sep	The string between the two bounds of a confidence interval. NULL keeps spicy's default, which is ", " and turns into "; " when the decimal mark is a comma.
ci_brackets	A character vector of length two, the opening and closing delimiters of a confidence interval, e.g. c("[", "]") or c("(", ")").
stars	Significance stars: FALSE to suppress them, or a named numeric vector of symbol-to-threshold pairs such as c("***" = .001, "**" = .01, "*" = .05).
digits, effect_size_digits, fit_digits, ic_digits	Decimal places for estimates, effect sizes, fit statistics and information criteria. Each is applied only by the table functions that have the matching argument: effect_size_digits reaches the association measure and the SMD of table_categorical() too.
percent_digits	Decimal places for percentages.
note_label	The word that opens the note under the table, set in italics in every rendered output: "Note." (the default, APA Manual 7 section 7.14), "Note:" (Chicago Manual of Style 18, paragraphs 3.77-3.81), or "" for no lead-in word (AMA Manual of Style 11, section 4.1.4.10). The language spells it: under options(spicy.language = "fr"), "Note:" is written "Note :", with the no-break space French puts before a colon, in every output. A note you type is printed as typed after it, unless it opens with a lead-in of its own, set in italics when it is the lead-in in force.
abbreviations	Which abbreviations the note under the table defines: those of its headers, its spanners and the labels of its stub (the fit statistics and the random effects of a

regression), not the symbols inside its cells. A definition identifies the quantity by its standard name, in one expression, the author included when the name carries one (g = Hedges's g); it never explains what the quantity measures or how to read it, which belongs to the text. `NULL` (the default) follows the rule of the theme in force, the APA rule when there is none (see *Themes*). `"all"` defines every abbreviation and symbol the table shows, abbreviated words such as Min and Max included, except n, N, p and the test statistics (t, z, F, χ^2), which no style and no journal defines: the setting for readers outside the field. `"none"` defines none, but keeps the disambiguations, the definitions without which the symbol is ambiguous (partial eta squared, the IQR as the width Q3 - Q1, the method of a standardized β, the design-based standard error). Either way the definitions take the form and the place of the theme in force. A `note` you type is printed as typed.

`minus_sign` The sign of a negative number in the outputs that are printed: `output = "flextable"` and `"word"` (a flextable knitted into Word too), `output = "gt"` in HTML, Word and LaTeX, the HTML, Typst and LaTeX of `output = "tinytable"` (and its markdown inside a knit to Word), and the PDF of `code_book()`, which reads the lever from `options(spicy.style)`. `"typographic"` (the default) sets the minus sign (Unicode U+2212, as wide as the plus sign) in place of the hyphen-minus, `"ascii"` keeps the hyphen-minus. It applies to the cells of the statistics (estimates, intervals, differences, effect sizes, thresholds, the summaries of a codebook) and to the numbers of a table's note (a hyphen before a digit, after a space, `=`, `(`, `[`, `,` or `;`: a date or `N-1` keep theirs), not to the labels, the codes of a codebook or the headers. LaTeX sets the sign in math mode, which every engine and font has. The outputs whose text is copied or read back keep the hyphen-minus – the data frame, the console, `output = "excel"`, `"clipboard"`, markdown and the long output: a spreadsheet or a parser reads a minus sign as text, not as part of a number. A printed table pasted into a spreadsheet therefore gives text; build it with `minus_sign = "ascii"` for that use. No theme sets it.

`v_digits` Defunct. Renamed `effect_size_digits` in spicy 0.14.0; supplying `v_digits` is an error (class `spicy_defunct`).

Details

A style is accepted by the `style` argument of the nine table functions (`table_categorical()`, `table_categorical_svy()`, `table_continuous()`, `table_continuous_svy()`, `table_continuous_lm()`, `table_outcome()`, `table_multiresponse()`, `table_regression()` and `table_regression_uv()`), either as a theme name or as the object returned here, and by `options(spicy.style = )` for document-wide scope.

Value

`spicy_style()` returns an object of class `spicy_style`: a named list holding only the levers that were set. Themes returned by name carry a `provenance` attribute with the journal, its source document and the exact list of encoded rules; `print()` shows it. `spicy_style_names()` returns the character vector of available theme names.

What a theme claims, and what it does not

A theme covers **numeric formatting conformity**, the word that opens the table note, the abbreviations that note defines and, in English, how the mean and the median are headed, **only** – not full editorial conformity. It does not check reporting guidelines, table structure, footnote symbols, units, or anything else a journal's instructions ask of a manuscript. Each theme below lists the exact rules it encodes, with the sentence it encodes them from; anything not listed is spicy's own default, not the journal's rule.

Themes only move **defaults**. An argument you type wins over the theme, always: `table_regression(fit, style = "lancet", digits = 3)` is a Lancet table with three decimals. One consequence worth stating: typing `p_digits` is a request for that many decimals on every p-value, so it also switches off the theme's own ways of choosing p-precision – `p_bands`, `p_sigfig`, and the `p_floor` derived from them. The leading-zero rule is orthogonal and stays.

Resolution order

An argument you type, then the `style` argument, then `getOption("spicy.style")`, then the typography of `getOption("spicy.language")`, then spicy's defaults. Within a style, an explicit function argument beats the style's value for the same lever.

The language's locale

A language brings its typography with it, at the bottom of that order. `options(spicy.language = "fr")` is therefore one gesture for a coherent French report table – French words, and numbers written the French way: comma decimal mark, and the leading zero French typography keeps on a p-value (`0,003`). "La virgule est utilisee pour separer les unites des decimales" (European Union, *Code de redaction interinstitutionnel*, French edition 2022, point 6.5, `https://style-guide.europa.eu/fr`); "Si le nombre se situe entre +1 et -1, le separateur decimal est toujours precede d'un zero" (BIPM, *Le Systeme international d'unites*, 9th edition, 2019, section 5.4.4, `https://www.bipm.org/documents/20126/41483022/S`
`"en"` brings no locale, and nothing changes for anyone who sets no language. The locale reaches the exploration pair as well: `freq()` and `cross_tab()` have no style layer, so the language sets the DEFAULT of their `decimal_mark` and nothing else. An argument you type still wins. The leading zero of a p-value works the same way in both worlds: its DEFAULT follows the mark – a comma keeps it (`0,003`), a point drops it (`.003`) – and `p_style` is the explicit lever that overrides that default wherever there is one, which is why a theme's rule survives any mark in the `table_*()` families while the pair, having no such lever, always follows its mark.

A theme composes with a locale rather than fighting it, because a theme encodes only what its own source states: `"jama"` fixes no decimal mark, so JAMA under a French language gives JAMA's p-value rules and the French comma. Where the two do meet, the theme wins – you asked for it by name. `"lancet"` keeps its midline decimal point; `"apa"` keeps its missing leading zero, which under a comma prints `,003`, a form the SI brochure forbids. That is the price of an explicit gesture; to keep APA's other rules and restore the zero, compose the way out yourself: `style = spicy_style("apa", p_style = "standard")`. One lever bends the other way: a theme's `", "` interval separator was sourced under a dot mark, and under a comma mark it would BE the mark – so it yields to the derived `"; "`, exactly as the French adaptations of APA style themselves write (`[3,45; 6,78]`). Any other separator (`" to "`, an en dash) is unambiguous and stays.

An argument beats both, which is the escape hatch for a bilingual table: `decimal_mark = "."` under a French language gives French words and a decimal point. It moves only the mark: the p-value

keeps the locale's leading zero (`0.003`), because `p_style` is a style lever with no argument of its own – `style = spicy_style(p_style = "apa")` drops it again.

See `spicy_labels()` for the language option itself.

Themes

`"jama"` – JAMA / JAMA Network:

Source: *Instructions for Table Creation*, JAMA Network author document, 23 February 2016 (`https://jamanetwork.com/DocumentLibrary/InstructionsForAuthors/InstructionsForTableCreation.pdf`), consulted 2026-08-14.

Encoded:

- p-values on two decimals, three below `.01`, floored at `<.001`, with no leading zero – "All P values should be reported to exact numbers to 2 digits past the decimal point, regardless of statistical significance. For values lower than .01, present the P value to 3 digits. Express any values lower than .001 as P<.001."
- no lead-in word before the note under the table (`note_label = ""`) – the *AMA Manual of Style*, 11th edition (2020), section 4.1.4.10, gives table footnotes none: a footnote on the whole table is called by a letter "placed after the table title".
- the abbreviations of the note on its first line, `Abbreviations: HR, hazard ratio; OR, odds ratio.`, in alphabetical order, `Abbreviation:` for one – section 4.1.4.10 again: that footnote comes first, "set off with an introductory word or phrase instead of a letter", and "abbreviations are expanded in alphabetical order". SD, SE and CI are not expanded (section 13.11, "no need to expand"), nor are the symbols of section 19.6 that do not carry "expand at first mention" (n, N, P, d, R^2, Δ, σ, ρ), B and M among them, which that section writes b and $\bar{x}$: it lists them as usable "without expansion except where noted".
- the mean and the median headed `Mean` and `Median`, not `M` and `Mdn` – section 4.1.5 writes "mean (SD)" and "median (IQR)" in full.

Not encoded: everything else in that document (percentages carrying their numerator and denominator, one datum per cell, footnote letters, SI conversion factors) is table *construction*, not number formatting. The document states no rule for the confidence-interval separator or the decimal mark, so spicy's defaults apply.

`"nejm"` – The New England Journal of Medicine:

Source: NEJM Author Center, *New Manuscripts*, section *Statistical Reporting Guidelines* (`https://www.nejm.org/autho`
a living page; its full text was read in a browser by the package maintainer on 2026-08-14 – the site refuses automated retrieval, which is why earlier surveys reported these rules as unavailable.

Encoded:

- tiered p-values, leading zero kept – "In general, P values larger than 0.01 should be reported to two decimal places, and those between 0.01 and 0.001 to three decimal places; P values smaller than 0.001 should be reported as P<0.001." The guideline's own text writes `0.01` / `0.001`: unlike JAMA, the leading zero stays.
- measures of association on two decimals – "measures of association, such as odds ratios, should ordinarily be reported to two decimal places" (a pin of spicy's default).

Not encoded: the guideline's stated exceptions to the p rule (stopping-rule tests, genomewide studies) are analysis contexts the style layer cannot see; its inference policy (no p-values without a prespecified multiplicity plan, estimates + 95% CI instead, no p-values in the Table 1 of a

randomized trial) is about what to report, not how to format it – request those layouts through `show_columns` and `p_value = FALSE` where you need them. No rule on the abbreviations of a table note was found, so the note defines them as spicy's default does.

`"lancet"` **– The Lancet:**

Sources: *Information for Authors*, April 2026; *Randomised trials in The Lancet: formatting guidelines* and *Observational studies in The Lancet: formatting guidelines*, both last updated July 2025 (`https://www.thelancet.com/pb-assets/Lancet/authors/tl-info-for-authors-1690986041530.pdf`), consulted 2026-08-14.

Below, the journal's midline decimal point (Unicode U+00B7, MIDDLE DOT) is written `[.]` so it cannot be confused with an ordinary period.

Encoded:

- midline decimal mark, U+00B7, on every number in the table – "Type decimal points midline (ie, 23`[.]`4, not 23.4)."
- p-values on two significant figures, capped at four decimals, floored at `p<0[.]0001`, leading zero kept – "Supply p values to two significant figures (capped at four decimal places), or p<0[.]0001."
- en dash between the bounds of a confidence interval. **This one is not a written rule**: the journal states none. It matches the intervals printed throughout the journal's own model tables and figures (`0[.]78 (0[.]60-1[.]00)`) and is encoded as conformity to the journal's published examples, nothing stronger.
- no lead-in word before the note under the table (`note_label = ""`). **Not a written rule either**: the model tables of the two formatting guidelines open their note on "Data are n (%)", and none of the 38 table notes of 15 research articles published in 2025-2026, checked on 2026-10-20, opens with "Note".
- the abbreviations spelled out at the end of the note, one `HR=hazard ratio.` sentence each, in alphabetical order, with *n*, *N*, *p*, SD, SE, IQR and CI left undefined. The artwork guidelines ask authors to "list and spell out all abbreviations in the table"; **the form and the exemptions are not a written rule**: they are those of the model tables of the two formatting guidelines ("Data are n (%), median (IQR), mean (SD) ... BMI=body-mass index.") and of 35 tables of 13 research articles published in 2025-2026, checked on 2026-10-20.
- the mean and the median headed `Mean` and `Median`, not `M` and `Mdn`, as the "median (IQR), mean (SD)" of the same model tables (not a written rule).

Not encoded: the empty-cell filler, the ban on p-values in a randomized trial's baseline table, and the absolute-rather-than- relative effect rule are content decisions, not number formats.

One caveat on the en dash. The journal's examples are ratio measures, whose bounds are positive. On the identity scale a negative lower bound puts a minus sign next to the dash (`[-5.17--2.58]`), which reads badly. The journal states no rule for that case and none is invented here; override the separator when it arises: `spicy_style("lancet", ci_sep = " to ")`.

`"annals"` **– Annals of Internal Medicine:**

Source: *Information for Authors*, American College of Physicians, document publication date 08/04/2026 (`https://www.acpjournals.org/pb-assets/pdf/AnnalsAuthorInfo-1755188286957.pdf`), consulted 2026-08-14.

Encoded:

- p-values on three decimals up to 0.20 and two decimals above it, floored at `P<0.001`, leading zero kept – "For P values between 0.001 and 0.20, please report the value to the nearest thousandth. For P values greater than 0.20, please report the value to the nearest hundredth. For P values less than 0.001, report as 'P<0.001.'"

Not encoded: the percentage rule ("Report percentages to one decimal place ... when sample size is $\geq$ 200", no decimals below 200) is conditional on a per-group sample size that the style layer does not see; set `percent_digits` yourself. The `mean (SD)` notation the journal asks for, and its refusal of $\pm$, are already how spicy renders dispersion. No rule on the abbreviations of a table note was found, so the note defines them as spicy's default does.

`"apa"` **– APA Style, 7th edition:**

Sources: *APA Style numbers and statistics guide*, last updated 11 September 2024 (`https://apastyle.apa.org/instruct`
Sample tables, last updated June 2024 (`https://apastyle.apa.org/style-grammar-guidelines/tables-figures/sa`
both consulted 2026-08-14.

Encoded – and this theme **pins** rather than changes: spicy's defaults already follow the guide, so `style = "apa"` is a promise that a future change of those defaults will not move an APA-formatted table.

- p-values on three decimals, floored at `<.001`, no leading zero – "Report exact p values to two or three decimals (e.g., p = .006, p = .03)", "report p values less than .001 as 'p < .001.'", and "Do not use a zero before a decimal when the statistic cannot be greater than 1 (proportion, correlation, level of statistical significance)."
- estimates and dispersion on two decimals – "Report other means and standard deviations and correlations, proportions, and inferential statistics (t, F, chi-square) to two decimals."
- confidence intervals in square brackets, bounds separated by a comma – the *Sample tables* page gives `95% CI [LL, UL]` as one of the two official layouts.
- the note under the table opens with "Note." in italics (`note_label = "Note."`) – the *Publication Manual*, 7th edition (2020), section 7.14: the general note is "designated by the word 'Note' (italicized) followed by a period".
- the abbreviations defined at the end of the general note, `CI = confidence interval; LL = lower limit`, in the order they appear in the table – sections 7.14 and 7.15 ("define the abbreviations in the order that they appear in the table"); AIC and BIC are among those section 6.44 names. The symbols that represent statistics are left undefined ("Do not define symbols or abbreviations that represent statistics (e.g., M, SD, F, t, df, p, N, n, OR, r)", section 6.44; section 7.15 adds *SE*), and spicy counts *Mdn* (the median of Table 6.5) and Hedges's *g* among them. So are the symbols composed of Greek letters (the same two sections) and those its sample tables leave undefined (*B*, β, R^2, ΔR^2, Cohen's *d*). This is also spicy's default rule, the one `"nejm"`, `"annals"` and `"aer"` apply.

Not encoded: the one-decimal rule for means of integer scales (depends on what the variable measures, which spicy cannot know), and the thousands separator (see *Known gaps* below). Nor the star thresholds. The official correlation and ANOVA sample tables mark `.05` / `.01` / `.001`, which is what `stars = TRUE` gives you – but as spicy's own default, not as anything this theme pins: one `stars` lever carries both the thresholds and the decision to show them, and the APA regression sample table shows none. So the theme sets no `stars`, and `style = "apa"` never switches them on.

`"aer"` **– American Economic Review / AEA journals:**

Source: *AER Style Guide*, American Economic Association (`https://www.aeaweb.org/journals/aer/style-guide`), consulted 2026-08-14. The Tables section is shared with AEJ: Applied, AEJ: Macro, AEJ: Policy, JEL and AEA P&P.

Encoded:

- leading zero on every decimal fraction, p-values included – "Place a zero in front of the decimal point in all decimal fractions (e.g., 0.357, not .357)."
- no significance stars, pinned – "Do not use asterisks to denote significance of estimation results. Report the standard errors in parentheses." This is the only outright written ban on stars in the whole surveyed corpus, so it is encoded even though spicy already defaults to `stars = FALSE`. Ask for `show_columns = c("b", "se")` for the standard-error layout.

Not encoded: the guide fixes no number of decimals, no p-value floor, no interval format and no decimal mark, so spicy's defaults apply. Horizontal-rules-only, no shading, a nine-column maximum and "Panel A / Panel B" blocks are layout, not number formatting. No rule on the abbreviations of a table note was found, so the note defines them as spicy's default does.

Known gaps

Two rules the sources state but this release does not encode:

- **thousands separator** (APA's comma, the EU code's thin no-break space). spicy formats numbers through more than one path, and a separator applied to some of them only would print `1,234` in one column and `1234` in the next – the exact inconsistency the SI brochure forbids ("the format used should not vary within one column").
- **significant-figure rounding of estimates** (Epidemiology's `nn` / `n.n` / `0.nn`, Science's "report only significant digits"). spicy's `digits` is a decimals contract applied per column family; a blanket significant-figure mode would also round fit statistics, which those rules do not ask for.

Journals deliberately absent: **BMJ** (its house-style page could not be read from any official source, and the widely repeated "95% CI 1.2 to 3.4" rule traces to no BMJ document), **Econometrica** (verified negative: the official guidelines contain no numeric table rule), **QJE** (its instructions page could not be read), **Epidemiology** (see *Known gaps*). Naming any of them would mean inventing their rules.

Examples

```
fit <- lm(mpg ~ wt + hp, data = mtcars)

# A named theme.
table_regression(fit, style = "jama")

# An argument you type beats the theme.
table_regression(fit, style = "jama", p_digits = 4)

# A style composed by hand.
table_regression(fit, style = spicy_style(decimal_mark = ",",
                                          p_style = "standard"))

# A theme with one rule changed.
table_regression(fit, style = spicy_style("lancet", ci_sep = " to "))
```

```
# Document-wide scope.
old <- options(spicy.style = "apa")
table_continuous(mtcars, c(mpg, wt))
options(old)

# Every abbreviation defined, for readers outside the field.
table_continuous(mtcars, c(mpg, wt),
                 style = spicy_style(abbreviations = "all"))

# What a theme encodes, and where it comes from.
spicy_style("lancet")
```

---

sum_n *Row sums with an optional minimum-valid-values rule*

Description

Computes row-wise sums across selected numeric columns of a `data.frame` or `matrix`. Missing values are handled per row via `min_valid` (an integer count or proportion of non-`NA` values required); rows that fail the rule return `NA`, and rows with no valid values at all return `NA` even when `min_valid = 0`. Non-numeric columns are dropped silently (set `verbose = TRUE` to see which). Designed to flow inside `dplyr::mutate()`: when called without an explicit `data` argument, the current data context is used.

Usage

```
sum_n(
  data = NULL,
  select = tidyselect::everything(),
  exclude = NULL,
  min_valid = NULL,
  digits = NULL,
  regex = FALSE,
  verbose = FALSE,
  user_na = TRUE
)
```

Arguments

`data`	A `data.frame` or `matrix`. Optional inside `dplyr::mutate()`, where the current grouping/data context is used automatically.
`select`	Columns to include. If `regex = FALSE`, use tidyselect syntax (default: `tidyselect::everything()`). If `regex = TRUE`, provide a regular expression pattern (character string).
`exclude`	Columns to exclude (default: `NULL`).
`min_valid`	Minimum number of valid (non-`NA`) values required per row. Accepts: • `NULL` (the default) – every selected column must be valid.

- a proportion in `(0, 1)` – `round(ncol(x) * min_valid)` valid columns required (e.g. `min_valid = 0.5` requires at least half of the selected columns to be non-`NA`).
- a non-negative integer count up to the number of selected numeric columns.

Non-integer values `>= 1` (e.g. `1.5`) and counts greater than `ncol(x)` raise an actionable error.

Rows with zero valid values always return `NA`, even when `min_valid = 0`: an empty row-wise summary is undefined, so the raw `rowMeans()` / `rowSums()` identities (`NaN` / `0`) are never returned.

`digits` Optional non-negative integer giving the number of decimal places to round the result to. Defaults to `NULL` (no rounding).

`regex` Logical. If `FALSE` (the default), uses tidyselect helpers. If `TRUE`, the `select` argument is treated as a regular expression.

`verbose` Logical. If `FALSE` (the default), messages are suppressed. If `TRUE`, prints a message about non-numeric columns excluded.

`user_na` Logical. If `TRUE` (the default), declared missing values count as missing – both in the computed summary and in the `min_valid` valid-count gate. If `FALSE`, the declared codes are treated as ordinary numbers. See the "Declared missing values" section of `freq()`.

Value

A numeric vector of row-wise sums.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

See Also

Other row-wise summaries: `count_n()`, `mean_n()`

Examples

```
library(dplyr)

# Create a simple numeric data frame
df <- tibble(
  var1 = c(10, NA, 30, 40, 50),
  var2 = c(5, NA, 15, NA, 25),
  var3 = c(NA, 30, 20, 50, 10)
)

# Compute row-wise sums (all values must be valid by default)
sum_n(df)

# Require at least 2 valid (non-NA) values per row
sum_n(df, min_valid = 2)

# Require at least 50% valid (non-NA) values per row
sum_n(df, min_valid = 0.5)

# Round the results to 1 decimal
sum_n(df, digits = 1)

# Select specific columns
sum_n(df, select = c(var1, var2))

# Select specific columns using a pipe
df |>
  select(var1, var2) |>
  sum_n()

# Exclude a column
sum_n(df, exclude = "var3")

# Select columns ending with "1"
sum_n(df, select = ends_with("1"))

# Use with native pipe
df |> sum_n(select = starts_with("var"))

# Use inside dplyr::mutate()
df |> mutate(sum_score = sum_n(min_valid = 2))

# Select columns directly inside mutate()
df |> mutate(sum_score = sum_n(select = c(var1, var2), min_valid = 1))

# Select columns before mutate
df |>
  select(var1, var2) |>
  mutate(sum_score = sum_n(min_valid = 1))

# Show verbose message
df |> mutate(sum_score = sum_n(min_valid = 2, digits = 1, verbose = TRUE))
```

```
# Add character and grouping columns
df_mixed <- mutate(df,
  name = letters[1:5],
  group = c("A", "A", "B", "B", "A")
)
df_mixed

# Non-numeric columns are ignored
sum_n(df_mixed)

# Use inside mutate with mixed data
df_mixed |> mutate(sum_score = sum_n(select = starts_with("var")))

# Use everything(), but exclude known non-numeric
sum_n(df_mixed, select = everything(), exclude = "group")

# Select columns using regex
sum_n(df_mixed, select = "^var", regex = TRUE)
sum_n(df_mixed, select = "ar", regex = TRUE)

# Apply to a subset of rows
df_mixed[1:3, ] |> sum_n(select = starts_with("var"))

# Store the result in a new column
df_mixed$sum_score <- sum_n(df_mixed, select = starts_with("var"))
df_mixed

# With a numeric matrix
mat <- matrix(c(1, 2, NA, 4, 5, NA, 7, 8, 9), nrow = 3, byrow = TRUE)
mat
mat |> sum_n(min_valid = 2)
```

`table_categorical` *Categorical summary table*

Description

Builds a publication-ready frequency or cross-tabulation table for one or many categorical variables selected with tidyselect syntax.

With `by`, produces grouped cross-tabulation summaries (using `cross_tab()` internally) with a *p*-value per variable, from Pearson's chi-squared test or Fisher's exact test (see `test`), and optional association measures. Without `by`, produces one-way frequency-style summaries.

Multiple output formats are available via `output`: a printed ASCII table (`"default"`), a numeric `data.frame` (`"data.frame"`, the displayed table, or `"long"`, one row per variable, level and group), or publication-ready tables (`"tinytable"`, `"gt"`, `"flextable"`, `"excel"`, `"clipboard"`, `"word"`).

Usage

```
table_categorical(
  data,
  select = tidyselect::everything(),
  by = NULL,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  levels_keep = NULL,
  include_total = TRUE,
  total_row = FALSE,
  drop_na = FALSE,
  weights = NULL,
  rescale = getOption("spicy.rescale", FALSE),
  correct = FALSE,
  simulate_p = FALSE,
  simulate_B = 2000,
  test = c("auto", "chisq", "fisher", "midp", "n1"),
  p_value = NULL,
  p_adjust = "none",
  percent = c("column", "row"),
  percent_base = c("valid", "total"),
  percent_digits = 1,
  p_digits = 3,
  effect_size_digits = 2,
  assoc_measure = "auto",
  assoc_ci = FALSE,
  smd = FALSE,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  indent_text = "  ",
  indent_text_excel_clipboard = strrep(" ", 6),
  add_multilevel_header = TRUE,
  blank_na_wide = FALSE,
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  user_na = TRUE,
  style = NULL,
  v_digits
)
```

Arguments

data	A data frame.
select	Columns to include as row variables. Supports tidyselect syntax and character vectors of column names. When omitted, defaults to every eligible categorical column in data: factor, character, logical, and labelled (haven / labelled) columns, excluding the by column – matching the select-less defaults of table_continuous() and table_continuous_lm(). An explicit select is taken verbatim (numeric columns included), so numeric-coded categorical variables can still be tabulated by naming them. The by and weights columns are not part of the selection: everything() and the other helpers skip them, and naming one in select, or giving its position, is an error. A name that is not a column of data is an error (class spicy_missing_column), and so is a selection with no column to tabulate. The levels of a variable are its rows, in the order by gives its groups (see by), a labelled variable's included.
by	Optional grouping column used for columns/groups. Accepts an unquoted column name or a single character column name. Factor levels keep their declared order, which is the way to choose another order; any other by is sorted, as in freq() and cross_tab(): text in the C locale (capitals before lower case, the same on every machine), FALSE before TRUE, numbers ascending, whatever the order of the rows. The column of missing values comes last. A haven labelled by names its groups by their value labels, as the rows of a labelled variable name its levels ("[1] Women"), in the order of its labels attribute when every observed value carries a label and by ascending code otherwise; a value label no observation holds adds no group, and declared missing values follow user_na as usual. The rule of the whole family.
labels	How each variable is named in the table: in the rows, in the title, and in the notes. Four forms: • labels = FALSE shows the column name. • labels = TRUE shows the column's label attribute (as set by haven, labelled, or label_from_names()) when it is present, not NA, and not empty, and the column name otherwise. • labels = c(bmi = "Body mass index"), a named character vector, gives the columns it names these labels. Every other column follows options(spicy.var_labels): its name when the option is unset or FALSE, its label attribute when it is TRUE. • labels = var_labels(bmi = "BMI (kg/m2)") gives the columns it names these labels, and every other column its label attribute, whatever the option says. var_labels() alone is labels = TRUE. A name that matches no column of the table warns (class spicy_unknown_name) and is left out, so one vector can still serve every table of a project. The default reads options(spicy.var_labels), and a TRUE or FALSE typed in the call wins over it. Any other value, NULL included, is an error. A variable label documents the data file; a published table carries the wording you set here. note_labels gives a variable a shorter label in the title and the notes.
note_labels	The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one labels gives it: a named character

vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and `NULL` (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`levels_keep` Levels to keep, in the order to display them. If `NULL` (the default), all observed levels are kept. Entries must match the level strings the table displays (for labelled columns these are the `"[code] label"` strings, not the bare label text). Two forms, as in `table_categorical_svy()`:

- a character vector applies to every variable: each keeps the levels it has, and a variable with none of them is dropped from the table with a classed warning (`spicy_no_selection`) listing its level strings;
- a named list applies per variable (`levels_keep = list(smoking = "Yes")`): each name must be a variable of the table, each level one the variable displays (an error otherwise), and the variables it does not name keep all their levels.

`include_total` Logical. If `TRUE` (the default), includes a `Total` group when available. A level of `by` displayed with the label of that margin is then an error (class `spicy_label_collision`): rename the level, or give the margin another label with `options(spicy.labels = list(header_margin_total = "All"))`.

`total_row` Logical. If `TRUE`, closes the block of each variable with a `Total` row under its levels (after the `"(Missing)"` row when there is one): the count of the variable in each group column and in the `Total` column, the `"(Missing)"` row included, and its percentage, 100 in every column under `percent = "column"` and, under `percent = "row"`, the share of each group in the variable's total, the base rate the row percentages are compared with (on the observed values under the default `percent_base`). Without `by`, the row holds the count of the variable and 100. A thin rule sets the row apart, in the console and in the rendered outputs. The row never enters the test, the association measure or the SMD. It takes the label of the margin, so a level displayed with that label is then an error (class `spicy_label_collision`), as beside the `Total` column. `FALSE` (the default) adds no row: the note gives the sample size.

`drop_na` Logical. The rule of the family: a table that counts shows the missing values by default, as a frequency table reports how many answers are missing; a table that computes a statistic removes them by default and counts them in the note (see `table_continuous()`). This table counts, so `FALSE` (the default) displays the missing values as a dedicated `"(Missing)"` level (and, under `by`, a `"(Missing)"` group column) – the field convention for descriptive tables (gtsummary's "Unknown" row, janitor's `NA` row; see the Epidemiologist R Handbook, Descriptive tables). Under the default `percent_base = "valid"`, the row shows their count and no percentage: the percentages are on the observed values, and the Total row of row percentages too. A level displayed beside that row or column with its label is an error (class `spicy_label_collision`): rename the level, or give the row another label with `options(spicy.labels = list(row_missing_level =`

"(Not stated)")). If TRUE, rows with NA in the tabulated variable (and in by, when supplied) are removed BEFORE each cross-tabulation, and the removal is disclosed in a table note, by last ("Missing values removed: smoking (25), sex (40).") rather than silent: a missing by takes the row out of every variable of the table, a missing value only out of its own variable. A by that no observation holds is then an error (class spicy_invalid_data).

weights — Optional weights. Either NULL (the default), a numeric vector of length nrow(data), or a single column in data supplied as an unquoted name or a character string. Rows with an NA weight leave the table, and the note says how many.

rescale — Logical. If FALSE, weights are used as-is. If TRUE, rescales weights so total weighted N matches raw N. Passed to spicy::cross_tab(). The default reads options(spicy.rescale), FALSE when it is unset, matching cross_tab().

correct — Logical. If FALSE (the default), no continuity correction is applied. If TRUE, applies Yates correction in 2x2 chi-squared contexts. Passed to spicy::cross_tab(). A variable tested by Fisher's exact test, its mid-p version, or the N-1 chi-squared (see test) takes no correction.

simulate_p — Logical. If FALSE (the default), uses asymptotic p-values. If TRUE, uses Monte Carlo simulation. Passed to spicy::cross_tab(). Applies to the Pearson chi-squared test only (see test).

simulate_B — Integer. Number of Monte Carlo replicates when simulate_p = TRUE, and for the Monte Carlo version of Fisher's exact test (see test). Defaults to 2000.

test — Character. The test of each variable when by is used. One of "auto" (default), "chisq", "fisher", "midp", or "n1".

- "auto": Pearson's chi-squared test when every expected count of the variable's table is at least 5, Fisher's exact test otherwise, the rule of gtsummary's add_p(). The choice is made per variable, and the note names the test of each.
- "chisq": Pearson's chi-squared test for every variable, the behavior before 0.14.0.
- "fisher": Fisher's exact test for every variable.
- "midp": the Fisher mid-p test for a 2x2 table.
- "n1": the N-1 chi-squared test for a 2x2 table whose expected counts are all at least 1, Fisher's exact test for one with a smaller count (Campbell's rule).

Under "midp" and "n1", a variable whose table is not 2x2 follows "auto". correct and simulate_p apply to the Pearson chi-squared test only. When the exact enumeration of a large sparse table fails, Fisher's p-value is its Monte Carlo estimate over simulate_B tables, and the note says so; call set.seed() first to reproduce it. See *Tests*. Ignored when by is not used or when p_value = FALSE.

p_value — Logical or NULL. If TRUE and by is used, adds the *p*-value column of the test chosen by test. When NULL (the default), the *p*-value is shown automatically whenever by is supplied, and hidden otherwise. Pass p_value = FALSE to remove the test from every output: the p column, and the test_type, statistic, df and p.value columns of "long"; test, correct and simulate_p are then

ignored, with a warning. The association measure keeps its own switch, so `p_value = FALSE, assoc_measure = "none"` gives a purely descriptive table. Ignored when by is not used.

`p_adjust` Multiple-comparison adjustment of the p column, applied to the family of every test in the table (one p-value per variable). One of `"none"` (default), `"holm"`, `"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"` / `"fdr"`, or `"BY"`, as in `table_regression()`; delegated to `stats::p.adjust()`. Needs by and the p column. See *Multiple-comparison adjustment*.

`percent` The direction of the percentages under by, as in `cross_tab()`: `"column"` (the default), the distribution of each group over the levels of the variable, summing to 100 down each group; or `"row"`, the share of each level falling in each group, summing to 100 across the groups, so that the `Total` column shows 100 on every level row. Unique abbreviations are accepted (`"c"`, `"r"`). Only the displayed shares change: the counts, the test, the association measure and the SMD are the same in both directions. The note says the direction in both cases ("Percentages are column percentages.", "Percentages are row percentages."). Without by, each variable has one distribution to show, and `percent = "row"` is ignored with a warning (class `spicy_ignored_arg`).

`percent_base` The base of the percentages of a variable whose missing values the `"(Missing)"` row shows (`drop_na = FALSE`). `"valid"` (the default) computes them on the observed values of the variable, as gtsummary does: the levels sum to 100, and the `"(Missing)"` row shows its count and no percentage (a blank cell, `NA` in the data frames). `"total"` computes them on every row, the `"(Missing)"` row included, as SPSS does and as this function did before 0.14.0. Under by, the base is the observed values of each column, the `Total` column and the column of missing by values included. Row percentages keep each row as its base, the `"(Missing)"` row included, which keeps its row percentages; the Total row of `total_row = TRUE` gives the share of each group on the observed values under `"valid"`. The test, the association measure and the SMD run on the observed values whatever the base. Without missing values to show (`drop_na = TRUE`, or a variable with none) the two bases give the same percentages. Unique abbreviations are accepted (`"v"`, `"t"`).

`percent_digits` Number of digits for percentages in report outputs. Defaults to `1`. A whole number: a fraction is an error, as for every `*_digits` argument.

`p_digits` Integer >= 1. Number of decimal places used to render *p*-values in the p column (default: `3`, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: `p_digits = 3` prints `.045` and `<.001`; `p_digits = 4` prints `.0451` and `<.0001`. Leading zeros are always stripped, following APA convention.

`effect_size_digits` Number of digits for the association measure, its confidence interval and the SMD, as for the effect sizes of the other table functions. Defaults to `2`.

`assoc_measure` Which association measure to report alongside the *p*-value. Accepts four input shapes:

- `"none"` – drop the column entirely.

- "auto" (the default) – pick a measure per row variable based on the variable type: a 2x2 table (binary row variable vs. binary by) uses phi, a pair of ordered factors uses tau_b, every other case uses cramer_v.
- a single string from c("cramer_v", "phi", "gamma", "tau_b", "tau_c", "somers_d", "lambda") – applied uniformly to every row variable.
- a character vector with one entry per row variable. Both **named** (c(smoking = "phi", health = "tau_b"), recommended; unnamed variables fall back to "auto") and **unnamed** positional (c("phi", "tau_b", "auto"), paired up with select) are accepted. Named is more robust to reordering of select.

When a single measure is used for every row, its symbol heads the column and names it in the data frames, as the effect size of table_continuous_lm() is headed: V (cramer_v), ϕ (phi), τ_b (tau_b) or γ (gamma), the three others keeping their name ("Lambda"). The note defines the symbol as it defines any abbreviation, under the abbreviations rule of spicy_style(): "V = Cramér's V" by default, the Greek letters only under "all" and in the "jama" and "lancet" themes. When multiple measures are used (typically with "auto" on a heterogeneous select), the header collapses to "Effect size" and the table note names the measure used for each variable.

phi requires a 2x2 table; if explicitly requested for a non-2x2 variable, an error is raised so the user can choose another measure or fall back to "auto".

assoc_ci — Passed to cross_tab(). If TRUE, includes the 95\ and in the "data.frame", "excel" and "clipboard" outputs, two extra columns headed 95% CI LL / 95% CI UL are added, the level stated as the other tables state it; in the long output ("long") the bounds fill es_ci_lower / es_ci_upper. In rendered formats ("gt", "tinytable", "flextable", "word"), the CI is shown inline (e.g., .14 [.08, .19]) under a header that states its level (V [95% CI]). Defaults to FALSE.

smd — Logical. If TRUE, adds an SMD column holding the standardized mean difference between the two groups of by, the balance diagnostic of the Table 1 literature. Requires exactly two groups; the value sits on the variable row beside p, never on a level row. Signed for a two-category variable (group 1 minus group 2 on the second category), unsigned for three or more, where it is a distance. No confidence interval and no p-value, by design. Rounded with effect_size_digits. See the "Standardized mean difference" section below. Defaults to FALSE.

decimal_mark — The decimal mark, a single character: "." (the default) or "," in most styles; the Lancet style of spicy_style() uses the middle dot.

align — Horizontal alignment of numeric columns in the printed ASCII table and in the tinytable, gt, flextable, word, and clipboard outputs. The first column (Variable) is always left-aligned. One of:

- "decimal" (default): align numeric columns on the decimal mark, the standard scientific-publication convention used by SPSS, SAS, and LaTeX siunitx. Numeric cells are pre-padded with figure-spaces (U+2007, digit-width) so every string in a column has the same width with the decimal mark at the same internal position; centering those uniform-width strings then stacks the decimal points vertically. The same pad-then-center strategy

is applied on every rendering engine (`gt`, `tinytable`, `flextable`, `word`, ASCII print) for a homogeneous rendering, matching `table_regression()` and `table_continuous_lm()`. The `clipboard` output is delimited text meant to be parsed rather than read at a fixed width, so its cells travel unpadded (a padded number pastes as text next to an unpadded number).

- `"center"`: center-align all numeric columns.
- `"right"`: right-align all numeric columns.

In the `excel` output, `"center"` centers the numeric columns and `"right"` is the same rendering as the default: cell-string padding does not align decimals under a proportional font, so `"decimal"` right-aligns instead, which combined with the per-column `numfmt` already produces dot-aligned columns. Same default and same three values as `table_continuous()` / `table_continuous_lm()`, whose workbooks resolve `"decimal"` differently: `table_continuous()` right-aligns only the counts and the *p*-value there, and `table_continuous_lm()` applies that convention at every `align`.

`note` The note under the table. `NULL` (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the `jama` and `lancet` themes; see `note_label` in `spicy_style()`), then, in a `table_regression()` with `stars`, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks (`"\n"`) included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (`note = "Note. Data weighted."`), and the legend of the stars stays below it. `""` keeps that legend alone, and no note at all without stars. `FALSE` removes the note from every output, the legend included.

`note_add` Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in `attr(x, "note")`, and the sentences the table writes stay. With a string in `note`, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An `NA` or an empty string is an error, and so is `note_add` with `note = FALSE`, which leaves no note to add to.

`output` Output format. One of:

- `"default"` (an ASCII table object, printed when the call is bare)
- `"data.frame"` (the displayed table, numeric)
- `"long"` (a long numeric `data.frame`)
- `"tinytable"` (requires `tinytable`)
- `"gt"` (requires `gt`)
- `"flextable"` (requires `flextable`)
- `"excel"` (requires `openxlsx2`)
- `"clipboard"` (requires `clipr`)
- `"word"` (requires `flextable` and `officer`)

`indent_text` Prefix used for modality labels in report table building. Defaults to `" "` (two spaces).

indent_text_excel_clipboard
: Stronger indentation used in Excel and clipboard exports. Defaults to six non-breaking spaces.

add_multilevel_header
: Logical. If TRUE (the default), merges top headers in Excel export. Only consulted for output = "excel" on a grouped table (by supplied): typed with another output, it is ignored with a warning (class spicy_ignored_arg), like the other arguments of one output (excel_path, excel_sheet, clipboard_delim, word_path, ...), and a one-way table, which has a single header row to begin with, does not use it.

blank_na_wide
: Logical. If FALSE (the default), NA values are kept as-is in output = "data.frame". If TRUE, the empty cells of its statistic columns become empty strings (the columns become text).

excel_path
: Path for output = "excel", ending in ".xlsx", in a directory that exists. Defaults to NULL.

excel_sheet
: Sheet name for Excel export. NULL (the default) uses "Categorical".

clipboard_delim
: Delimiter for clipboard text export. Defaults to "\t". A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path
: File path for output = "word", ending in ".docx", in a directory that exists. Defaults to NULL. Before 0.13.0, supplying it with output = "flextable" also wrote a .docx as a side effect; it is now used exclusively by output = "word" (the contract shared with the rest of the table family) and is ignored, with a warning, under any other output.

user_na
: Logical. If TRUE (the default), declared missing values in the row variables and in by are treated as missing: they join the "(Missing)" level under drop_na = FALSE, and under drop_na = TRUE they are removed with a dedicated disclosure line (Declared missing values removed: ...). If FALSE, the declared codes stay valid categories. See the "Declared missing values" section of freq().

style
: A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set options(spicy.style =) for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; ?spicy_style lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

v_digits
: Defunct. Renamed effect_size_digits in spicy 0.14.0, as in the other table functions; supplying v_digits is an error (class spicy_defunct).

Value

Depends on output:

- "default": a data.frame of class "spicy_categorical_table" carrying the rendering metadata as attributes. Its columns are internal and can change without notice: read the numbers from "data.frame", "long", broom::tidy(), broom::glance() or as_structured().

The object is returned visibly, so a bare `table_categorical(...)` call auto-prints the styled ASCII table at the console while `t <- table_categorical(...)` stays silent (print `t` to display the table).

- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per printed row – the header row of each variable, which carries its *p*-value and its association measure, then one row per level. The key columns are `variable` (the column name), `label` (the label the rows display for it, see `labels`), `level` and `.row_role` (`"factor_header"`, `"level"`, `"missing"` for the row of missing values the table adds, or `"total"` for the Total row of `total_row = TRUE`), then the printed columns: one pair `<group> n` / `<group> %` per group level (plus `Total` when `include_total = TRUE`), `p` (absent with `p_value = FALSE`), the association measure (named by its header: the symbol of the measure, `V`, its name for a measure without one, or `Effect size` when the row variables do not share one), its bounds `95% CI LL` / `95% CI UL` with `assoc_ci = TRUE`, and `SMD` with `smd = TRUE`. When `by = NULL`, the printed columns are `n` and `%`.
- `"long"`: a long `data.frame`, one row per variable, level and group, with columns `variable`, `label`, `level`, `.row_role` (the role of `level`: `"level"`, `"missing"` for the row of missing values, or `"total"` for the Total row of `total_row = TRUE`), then, when `by` is used, `group`, `.group_role` (the role of `group`: `"group"`, `"missing"` for the column of missing `by` values, or `"total"` for the margin) and `total` (`TRUE` on the rows of the margin and of the Total row), then `n`, `pct` (the percentage the table displays, in the direction of `percent`), then, with `by`, the test of the variable unless `p_value = FALSE`: `test_type` (`"chi_squared"` for Pearson's, `"n1_chi_squared"` for the N-1 chi-squared, `"fisher_exact"`, `"fisher_midp"` for the Fisher mid-p test, or `"fisher_exact_mc"` for the Monte Carlo estimate of Fisher's exact test), `statistic` and `df` (the chi-squared, Pearson's or the N-1, `NA` for a variable tested by an exact test), `p.value` (and `p_unadjusted` under `p_adjust`); and the association measure, whichever measure it is: `es_type` names it per row (`"cramer_v"`, `"phi"`, ...), `es_value` holds it, `es_ci_lower` / `es_ci_upper` its interval (`NA` unless `assoc_ci = TRUE`), and `smd_type` (`"binary"` or `"multinomial"`, the kernel the value came from) / `smd_value` the standardized mean difference. The measure and SMD columns are there whether or not they were requested, `NA` when not, so the schema does not move with an argument. The statistics of a variable sit on every row of its block.
- `"tinytable"`: a `tinytable` object.
- `"gt"`: a `gt_tbl` object.
- `"flextable"`: a `flextable` object.
- `"excel"` / `"word"`: writes to disk and returns the file path invisibly.
- `"clipboard"`: copies the table and returns the copied text invisibly.

The note opens with the sample size, "N = 159." or, with `by`, "N = 159: Women (n = 80), Men (n = 79).": the rows of the data the table describes, and those of each group column in column order, the column of missing `by` values included, so that the `n` add up to N. A row missing `by` under `drop_na = TRUE` and a row with a missing weight leave the table and are not counted; the note says how many. Under `weights` the cells hold weighted counts and N counts rows: "N = 60 (unweighted).". One N per table, the missing values of a variable in its `"(Missing)"` row or in the note. With `by`, the direction of the percentages follows, then the weights. Their base takes no sentence: the table shows it (see `percent_base`).

The note, the `drop_na = TRUE` disclosure included, travels with the table on every route, not just the console: `"default"` prints it under the ASCII table, `"tinytable"` / `"gt"` / `"flextable"` /

"word" carry it as a table note, "excel" writes it below the body, and the default object and the data frames of "data.frame" / "long" keep the whole note as printed, its lead-in included, in attr(x, "note") (NULL when there is nothing to say), so a pipeline that renders the numbers itself can still state what left the table. The default object also keeps the title the console prints, in attr(x, "title"). The same three objects record the direction of their percentages in attr(x, "percent"): "column" or "row" (see percent), and "column" for a table without by; and their base in attr(x, "percent_base"), "valid" or "total". The percentage columns of "data.frame", pct in "long" and the proportion of broom::tidy() hold the shares the table displays, in that direction and on that base: NA on the "(Missing)" row under "valid". On the "tinytable" route the note is set one size down; options(spicy.note_style) governs that (see table_regression()).

The Excel sheet carries the same title the console prints on its first row; the table itself starts on row 3.

Tests

When by is used, each selected variable is cross-tabulated against the grouping variable with cross_tab() and tested on the complete cases of its observed levels, a "(Missing)" level and a declared level nobody holds left out: one *p*-value per variable, in the p column. A table left with a single row or column (a variable observed at one level, or in one group) is uninformative and gets no test and no association measure, whatever test and assoc_measure say: its p and measure cells show an en dash, the undefined cell, and the note names no measure for it. A blank cell only means a statistic that was not requested or does not apply.

The chi-squared approximation is unreliable when expected counts are small. Under test = "auto", the default, a variable takes Pearson's chi-squared test (stats::chisq.test()) when every expected count of its table is at least 5, and Fisher's exact test (stats::fisher.test()) otherwise: the rule gtsummary's add_p() applies, with the same p-values. Fisher's exact test of a table larger than 2x2 is the Freeman-Halton extension. When its exact enumeration fails, which happens on large sparse tables, the p-value is its Monte Carlo estimate over simulate_B tables drawn with the observed margins: it is random, so call set.seed() first to reproduce it. test = "chisq" tests every variable by the chi-squared, the behavior before 0.14.0, and test = "fisher" every variable by the exact test.

Fisher's exact test is conservative: its actual level is often well below the nominal one, at a cost in power. test = "midp" gives a 2x2 table (a two-level variable by two groups) the Fisher mid-p test, as Fagerland, Lydersen and Laake (2017, equation 1.12, p. 28, and section 4.4.6, p. 97) and Lydersen, Fagerland and Laake (2009, equation 12, p. 1165) define it: the tables with the observed margins are ordered by their hypergeometric probability, and the *p*-value adds the probabilities of the tables less probable than the observed one to half the probabilities of the tables as probable, the observed one included, ties being read within the relative tolerance of stats::fisher.test(). The mid-p test can exceed the nominal level, usually by little, and Fagerland, Lydersen and Laake (2017, p. 175) recommend it for all sample sizes. A variable whose table is not 2x2 follows "auto", and the note names the test of each variable. The central variant, twice the smaller one-sided mid-p (the exact2x2 and epitools packages; Agresti, 2019, p. 49), coincides with this one when the row totals or the column totals are equal, except that it gives 1 for the most probable table; it differs otherwise and is not computed.

test = "n1" applies the rule of Campbell (2007, pp. 18-19) to a 2x2 table. When every expected count is at least 1, the table takes the N-1 chi-squared test: Pearson's statistic, with no continuity

correction, multiplied by (N - 1)/N and referred to a chi-squared distribution with 1 degree of freedom, which is the Mantel-Haenszel statistic of a single 2x2 table; `statistic` and `df` hold that statistic in the long output. Otherwise it takes Fisher's exact test, two-sided by Irwin's rule as `stats::fisher.test()` computes it. Campbell found the N-1 test close to its nominal level under that restriction, not without it. A variable whose table is not 2x2 follows `"auto"`, and the note names the test of each variable.

The note names the test behind the *p*-values, "P-values: Pearson chi-squared test (sex, education); Fisher exact test (smoking).", or the one test when it covers them all, "P-value:" when the table shows a single *p*-value. `test_type` in the long output gives the test of each variable. See `@param correct` / `simulate_p` to switch on Yates' continuity correction or Monte Carlo *p*-values for the chi-squared test, and `@param assoc_measure` for the per-row dispatch table used by `"auto"` (2x2 -> Phi, both ordered -> Kendall's Tau-b, otherwise Cramér's V). Without `by`, the table reports the marginal frequency distribution of each variable with no inferential statistics.

The exact tests condition on counts. Under `weights`, they run on the weighted table when its cells are whole numbers (frequency weights); otherwise a variable that a rule sends to Fisher's exact test keeps the chi-squared test, Pearson's under `test = "auto"` and the N-1 under Campbell's rule, with a warning (`spicy_fallback`), and `test = "fisher"`, or `test = "midp"` on a 2x2 table, is an error. For survey weights, `table_categorical_svy()` gives the design-based test.

The test and the measure are two decisions with two arguments: `p_value = FALSE` removes the test, `assoc_measure = "none"` the measure. Together they give a purely descriptive table, which keeps the group columns and their headers; `smd` stays available.

For model-based comparisons (cluster-robust SE, weighted contrasts, fitted means) on continuous outcomes, see `table_continuous_lm()`. For descriptive (empirical) comparisons on continuous outcomes, see `table_continuous()`.

Groups with no data

A variable with no observed value in a group of `by` that holds rows, typically a question not asked in that group, has no data there, which is not a missing answer. Its cells in that group show an en dash (`NA` in the data frames, in `tidy()` and in `as_structured()`), and its rows there are left out of everything computed for that variable: the missing values the note counts, the `Total` column, the `"(Missing)"` row, the test, the association measure and the SMD. A variable left with fewer than two groups with data is not compared: its `p` and measure cells show the dash, and the note names no test and no measure for it. The note says "No data for X in G.", in one sentence for the variables with no data in the same groups ("No data for X and Y in G."). Without `by`, a variable with no observed value at all keeps its declared levels (its one row when it declares none) with the dash, and the note says "No data for X." The column of missing `by` values is never such a group, and a declared level of `by` that holds no row keeps its zero column.

Multiple-comparison adjustment

The p-values of a descriptive table describe: they test no prespecified hypothesis. STROBE (explanation of item 14) and the SAMPL guidelines advise against significance tests in descriptive and baseline tables, hence the default `p_adjust = "none"`.

When a journal or a protocol asks for adjusted p-values anyway, `p_adjust` applies `stats::p.adjust()` to the family of every test in the table: one p-value per variable (one per block in `table_outcome()`),

never the rows of one variable, which share one test. `"holm"` suits a confirmatory set of comparisons, `"BH"` a screening one; `table_regression()` lists the methods. A test that returned no p-value stays out of the family.

The adjustment runs before any formatting, so the `p` column and its `<.001` threshold show the adjusted values, and the table note gives the number of tests and the method. The raw p-values stay in the object: a `p_unadjusted` column next to the p-value column of the returned data frames, and `col_meta$p$p_unadjusted` (with the method in `p_adjust` and the family size in `p_adjust_m`) in `as_structured()`. Without `by`, or with `p_value = FALSE`, the table shows no p-value to adjust, and a `p_adjust` other than `"none"` is an error.

Standardized mean difference

`smd = TRUE` adds an `SMD` column with the balance diagnostic of the Table 1 literature, on the variable row beside `p`. For a two-category variable it is the Bernoulli form,

$$\mathrm{SMD} = \frac{p_1 - p_2}{\sqrt{(p_1(1-p_1) + p_2(1-p_2))/2}}$$

with p the proportion of the SECOND category, **signed**, group 1 minus group 2 in the order the table displays them. Note the denominator: the Bernoulli variance $p(1-p)$ at *n*, not `var()` at $n-1$, which would be 19% off on a small table.

"Second category" is the order the table shows, which is worth knowing for a **logical** variable: spicy displays `TRUE` then `FALSE`, so the sign is taken on `FALSE`, where `tableone` and `cobalt` coerce with `factor()` (`FALSE`, `TRUE`) and take it on `TRUE` – the same magnitude with the opposite sign. Convert to a factor with the level order you want if the direction matters.

For three or more categories it is the multivariate form of Yang and Dalton (2012, SAS Global Forum 335-2012),

$$\mathrm{SMD} = \sqrt{T' S^{-} T}$$

with T the difference of the two profiles of proportions (first category dropped) and S the mean of their multinomial covariance matrices. This is a Mahalanobis distance: it is **unsigned**, it is **not bounded by 1**, and S^{-} is a pseudo-inverse, because a declared-but-unobserved category makes S singular and `solve()` would abort where the pseudo-inverse returns exactly the value that category's absence implies. Which kernel a row took is published as `smd_type` in the `"long"` output, and the unsigned reading is stated in the table note whenever a variable has more than two categories. The `MASS` package is needed for this arm only.

Two profiles with **no category in common** have an infinite standardized distance. The pseudo-inverse would quietly publish a finite number there, so the cell is an en-dash and a classed warning says why. The same applies when each group is constant on a different category, where the naive route publishes `0` – "perfectly balanced" for the most imbalanced variable possible.

Conventions shared with `table_continuous()`: exactly two groups (three or more are refused, not averaged over pairs); complete cases on the observed groups, so a `drop_na = FALSE` "(Missing)" level is displayed and never enters the diagnostic; no confidence interval and no p-value, by design. Under `weights` the profiles are the weighted proportions, which makes this column agree with both the frequency and the survey-design readings – a profile of proportions is invariant to a global

rescaling of the weights, so `rescale` cannot move it. (Only the continuous arm has a convention to choose there.)

The `SMD` cell keeps its leading zero where the association cell drops it: the APA strip belongs to a bounded measure, and this one is not bounded. The two columns therefore print `0.45` and `.45` side by side, on purpose.

A balance table usually shows no test: `p_value = FALSE` removes the `p` column here as it does in `table_continuous()`, so a table mixing continuous and categorical variables can drop both. `inline()` quotes the cell from the variable row (`inline(tbl, smoking, column = "smd")`).

Display conventions

Decimal alignment, *p*-value formatting, and required suggested packages per output engine are documented under `@param align`, `@param p_digits`, and `@param output` respectively.

Counts are displayed as integers: weighted counts are rounded (ties half to even, the R convention) at display time only, in cells and margins alike – the SPSS Crosstabs convention. Cells and margins are rounded independently, so small display discrepancies are possible (e.g. two cells of exactly 0.5 each display as `0` while their `Total` of 1.0 displays as `1`). The machine outputs (`"data.frame"`, `"long"`) carry the exact weighted counts and full-precision percentages.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

References

Agresti, A. (2019). *An Introduction to Categorical Data Analysis* (3rd ed.). Wiley.

Campbell, I. (2007). Chi-squared and Fisher-Irwin tests of two-by-two tables with small sample recommendations. *Statistics in Medicine*, 26(19), 3661-3675. doi:10.1002/sim.2832

Fagerland, M. W., Lydersen, S., & Laake, P. (2017). *Statistical Analysis of Contingency Tables*. Chapman and Hall/CRC. doi:10.1201/9781315374116

Lydersen, S., Fagerland, M. W., & Laake, P. (2009). Recommended tests for association in 2 x 2 tables. *Statistics in Medicine*, 28(7), 1159-1175. doi:10.1002/sim.3531

See Also

`table_continuous()` for empirical comparisons on continuous outcomes; `table_continuous_lm()` for the model-based companion (heteroskedasticity-consistent / cluster-robust / bootstrap / jackknife SE, fitted means, weighted contrasts); `cross_tab()` for two-way cross-tabulations; `freq()` for one-way frequency tables.

Other spicy tables: `table_continuous()`, `table_continuous_lm()`

Examples

```
# --- Basic usage ------------------------------------------------------

# Default: ASCII console table grouped by sex.
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = sex
)

# One-way frequency-style table (no `by`).
table_categorical(
  sochealth,
  select = c(smoking, physical_activity)
)

# Purely descriptive grouped table: no test, no association measure.
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = sex,
  p_value = FALSE,
  assoc_measure = "none"
)

# A small sample: `smoking` has an expected count below 5 and takes
# Fisher's exact test; the note names the test of each variable.
table_categorical(
  sochealth[1:40, ],
  select = c(smoking, physical_activity),
  by = sex
)

# Pretty labels keyed by column name.
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = education,
  labels = c(
    smoking           = "Current smoker",
```

```
    physical_activity = "Physical activity"
  )
)

# Survey weights with rescaling.
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = education,
  weights = "weight",
  rescale = TRUE
)

# Confidence interval for the association measure.
table_categorical(
  sochealth,
  select = smoking,
  by = education,
  assoc_ci = TRUE
)

# --- Per-variable association measure ---------------------------------

# Default (`assoc_measure = "auto"`): one measure per row variable based on
# the variable type (2x2 -> Phi, both ordered factors -> Kendall's Tau-b,
# otherwise Cramer's V). When the chosen measures differ across rows, the
# column header collapses to `"Effect size"` and the note names the
# measure used for each variable.
table_categorical(
  sochealth,
  select = c(smoking, education),
  by = sex
)

# Force a uniform measure across all row variables: its symbol, V,
# heads the column, and the note defines it.
table_categorical(
  sochealth,
  select = c(smoking, education),
  by = sex,
  assoc_measure = "cramer_v"
)

# Per-variable override (recommended named form).
table_categorical(
  sochealth,
  select = c(smoking, education, self_rated_health),
  by = sex,
  assoc_measure = c(
    smoking           = "phi",        # binary x binary
    education         = "cramer_v",   # multi-category nominal
    self_rated_health = "tau_b"       # ordinal x binary, Tau-b
  )
```

```
)

# --- Output formats ------------------------------------------------------

# The rendered outputs below all wrap the same call:
#   table_categorical(sochealth,
#                     select = c(smoking, physical_activity),
#                     by = sex)
# only `output` changes. Assign each result to a variable -- some
# engines auto-print as a console-friendly text fallback inside
# the `?` help viewer.

# Wide data.frame (one row per modality).
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = sex,
  output = "data.frame"
)

# Long data.frame (one row per (modality x group)).
table_categorical(
  sochealth,
  select = c(smoking, physical_activity),
  by = sex,
  output = "long"
)


# Rendered HTML / docx objects -- best viewed inside a
# Quarto / R Markdown document or a pkgdown article.
if (requireNamespace("tinytable", quietly = TRUE)) {
  tt <- table_categorical(
    sochealth, select = c(smoking, physical_activity), by = sex,
    output = "tinytable"
  )
}
if (requireNamespace("gt", quietly = TRUE)) {
  tbl <- table_categorical(
    sochealth, select = c(smoking, physical_activity), by = sex,
    output = "gt"
  )
}
if (requireNamespace("flextable", quietly = TRUE)) {
  ft <- table_categorical(
    sochealth, select = c(smoking, physical_activity), by = sex,
    output = "flextable"
  )
}

# Excel and Word: write to a temporary file.
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  tmp <- tempfile(fileext = ".xlsx")
```

```
  table_categorical(
    sochealth, select = c(smoking, physical_activity), by = sex,
    output = "excel", excel_path = tmp
  )
  unlink(tmp)
}
if (
  requireNamespace("flextable", quietly = TRUE) &&
    requireNamespace("officer", quietly = TRUE)
) {
  tmp <- tempfile(fileext = ".docx")
  table_categorical(
    sochealth, select = c(smoking, physical_activity), by = sex,
    output = "word", word_path = tmp
  )
  unlink(tmp)
}


## Not run:
# Clipboard: writes to the system clipboard.
table_categorical(
  sochealth, select = c(smoking, physical_activity), by = sex,
  output = "clipboard"
)

## End(Not run)
```

`table_categorical_svy` *Categorical summary table from a survey design*

Description

The design twin of `table_categorical()`: counts and estimated percentages of categorical variables computed from a `survey::svydesign()` or `survey::as.svrepdesign()` object instead of a data frame.

Every statistic is survey's. `survey::svymean()` estimates the percentages and their design effects, `survey::svyciprop()` their confidence intervals, and `survey::svychisq()` tests the association, Rao-Scott corrected and referred to the design degrees of freedom.

Usage

```
table_categorical_svy(
  design,
  select = tidyselect::everything(),
  by = NULL,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
```

```
  levels_keep = NULL,
  include_total = TRUE,
  drop_na = FALSE,
  proportion_ci = FALSE,
 ci_method = c("logit", "likelihood", "asin", "beta", "mean", "xlogit", "wilson"),
  ci_level = 0.95,
  chisq_statistic = c("F", "Chisq", "Wald", "adjWald", "saddlepoint"),
  deff = FALSE,
  df = NULL,
  p_value = NULL,
  p_adjust = "none",
  percent_digits = 1,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  indent_text = "  ",
  indent_text_excel_clipboard = strrep(" ", 6),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  user_na = TRUE,
  style = NULL
)
```

Arguments

design	A survey design: `survey::svydesign()` or `survey::as.svrepdesign()`.
select	Columns to tabulate, as a tidyselect expression on the design's variables. When omitted, every categorical column of the design (factor, character, logical, and labelled columns), as in `table_categorical()`; an explicit `select` is taken verbatim, so a numeric-coded variable can still be tabulated by naming it. The design's own columns are not variables to describe: its sampling weights, replicate weights, cluster identifiers (every stage, unless each cluster of a one-stage design is a single observation) and finite population corrections. `everything()` and the other selection helpers skip them, and naming one, or giving its position, is an error; `table_categorical()` describes it from the data. Strata stay variables, since a stratification variable such as a region is often one of interest. These columns are read from the design object. A design made by `survey::as.svrepdesign()` records none of them, and one made by `survey::svrepdesign()` names its sampling weights only in its call, as a formula, until `subset()`, `update()`, `postStratify()`, `calibrate()`, `rake()` or `trimWeights()` replaces the call: leave such a column out with a negation (`select = !w`).
by	A single grouping column: one column block per level, in the order of `table_categorical()`

(a factor's levels, any other column sorted: text in the C locale, FALSE before TRUE, numbers ascending), the order the levels of each variable follow too.

`labels` How each variable is named in the table: in the rows, in the title, and in the notes. Four forms:

- `labels = FALSE` shows the column name.
- `labels = TRUE` shows the column's `label` attribute (as set by haven, labelled, or `label_from_names()`) when it is present, not NA, and not empty, and the column name otherwise.
- `labels = c(bmi = "Body mass index")`, a named character vector, gives the columns it names these labels. Every other column follows `options(spicy.var_labels)`: its name when the option is unset or FALSE, its `label` attribute when it is TRUE.
- `labels = var_labels(bmi = "BMI (kg/m2)")` gives the columns it names these labels, and every other column its `label` attribute, whatever the option says. `var_labels()` alone is `labels = TRUE`.

A name that matches no column of the table warns (class `spicy_unknown_name`) and is left out, so one vector can still serve every table of a project. The default reads `options(spicy.var_labels)`, and a TRUE or FALSE typed in the call wins over it. Any other value, NULL included, is an error. A variable label documents the data file; a published table carries the wording you set here. `note_labels` gives a variable a shorter label in the title and the notes.

`note_labels` The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one `labels` gives it: a named character vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and NULL (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`levels_keep` Levels to keep, as a character vector (all variables) or a named list (per variable).

`include_total` Add a `Total` column block with the whole design's percentages (default TRUE, only with by). A group displayed with the label of that block is an error, as in `table_categorical()`.

`drop_na` Drop missing values. A table that counts shows them by default, the rule of `table_categorical()`: FALSE (the default) shows them as a `(Missing)` level, and a level displayed with that label beside them is an error, as there. Shown or dropped, they never enter the test: the p-value is computed on the complete cases either way, which is the convention `table_categorical()` applies. Dropped, they are counted in the note, by last: a missing by takes the observation out of every variable of the table, a missing value only out of its own variable. Under `drop_na = TRUE`, a by that no observation holds is an error (class `spicy_invalid_data`).

`proportion_ci` Add the confidence interval of each percentage.

ci_method — Interval method passed to survey::svyciprop(): "logit" (default), "likelihood", "asin", "beta", "mean", "xlogit" or "wilson".

ci_level — Confidence level of the intervals of proportion_ci (default 0.95): a number strictly between 0 and 1.

chisq_statistic — Statistic for survey::svychisq(): "F" (default), "Chisq", "Wald", "adjWald" or "saddlepoint". "saddlepoint" is refused on a replicate-weights design: survey computes its p-value without the denominator degrees of freedom there, so it comes out too small.

deff — Show the design effect of each percentage: FALSE (default), TRUE or "replace".

df — Degrees of freedom for the intervals. NULL (default) uses survey::degf() on each domain. It does not reach the test: survey::svychisq() has no df argument, so the Rao-Scott reference distribution keeps the design's own degrees of freedom and the note says so.

p_value — Show the p-value column (defaults to TRUE with by). A variable whose complete cases include negatively weighted rows is not tested: the Rao-Scott correction is a function of the design variance, which is not defined when the weights change sign. Its percentages are still reported, the note says which tests were withheld, and the call warns (spicy_negative_weights_no_test).

p_adjust — Multiple-comparison adjustment of the p-value column, applied to the family of every test in the table (one design-based p-value per variable; a withheld test stays out). One of "none" (default), "holm", "hochberg", "hommel", "bonferroni", "BH" / "fdr", or "BY", as in table_regression(); delegated to stats::p.adjust(). Needs by and the p-value column. See *Multiple-comparison adjustment*.

percent_digits — Number of digits for percentages in report outputs. Defaults to 1. A whole number: a fraction is an error, as for every *_digits argument.

p_digits — Integer >= 1. Number of decimal places used to render *p*-values in the p column (default: 3, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: p_digits = 3 prints .045 and <.001; p_digits = 4 prints .0451 and <.0001. Leading zeros are always stripped, following APA convention.

decimal_mark — The decimal mark, a single character: "." (the default) or "," in most styles; the Lancet style of spicy_style() uses the middle dot.

align — Horizontal alignment of the numeric columns: "decimal" (the default), "center" or "right", as in table_continuous(), whose rendering engines this table shares, its Excel output included.

note — The note under the table. NULL (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the jama and lancet themes; see note_label in spicy_style()), then, in a table_regression() with stars, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks ("\n") included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (note = "Note. Data weighted."), and the legend of the stars stays below it.

"" keeps that legend alone, and no note at all without stars. FALSE removes the note from every output, the legend included.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an empty string is an error, and so is note_add with note = FALSE, which leaves no note to add to.

output — One of "default", "data.frame", "long", or a rendering engine: "tinytable", "gt", "flextable", "excel", "clipboard", "word". "data.frame" is the displayed table with its numbers, "long" one row per variable, level and group, as in table_categorical(); see *Value*.

indent_text — Prefix used for modality labels in report table building. Defaults to " " (two spaces).

indent_text_excel_clipboard — Stronger indentation used in Excel and clipboard exports. Defaults to six non-breaking spaces.

excel_path — Path for output = "excel", ending in ".xlsx", in a directory that exists. Defaults to NULL.

excel_sheet — Sheet name for Excel export. NULL (the default) uses "Categorical".

clipboard_delim — Delimiter for clipboard text export. Defaults to "\t". A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path — File path for output = "word", ending in ".docx", in a directory that exists. Defaults to NULL.

user_na — Logical. If TRUE (the default), declared missing values in the row variables and in by are treated as missing: they join the "(Missing)" level under drop_na = FALSE, and under drop_na = TRUE they are removed with a dedicated disclosure line (Declared missing values removed: ...). If FALSE, the declared codes stay valid categories. See the "Declared missing values" section of freq().

style — A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set options(spicy.style =) for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; ?spicy_style lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

Depends on output:

- "default": a spicy_categorical_svy_table (a data.frame subclass, with the parent class "spicy_table"), printed when the call is bare; its columns are the table's compute

frame, internal and liable to change without notice, and the display frame and the typed view are attached.

- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per printed row (the header row of each variable, which carries its *p*-value, then its levels), the key columns `variable` (the column name), `label` (the label displayed), `level` and `.row_role` (`"factor_header"`, `"level"`, or `"missing"` on the row of missing values the table adds), then the printed columns in their order: per block of by (and the `Total` margin) `<group> n`, `<group> %`, with `proportion_ci` the interval bounds `<group> 95% CI LL` and `<group> 95% CI UL` (the level of `ci_level`, as printed), with `deff` `<group> DEff`, and last p. Without by, the columns of the one block are n, %, ...
- `"long"`: one row per variable, level and group, in the order of the table, with `variable`, `label`, `level`, `.row_role` (the role of `level`: `"level"`, or `"missing"`), with by `group`, `.group_role` (the role of `group`: `"group"`, `"missing"` for the column of missing by values, or `"total"`) and `total` (`TRUE` on the rows of the margin), then n (observed), `pct` (the estimated percentage), `ci_lower`, `ci_upper` (its interval, `NA` without `proportion_ci`), `deff` (`NA` without `deff`), and, with by and the *p*-value, `test_type` (the statistic of `survey::svychisq()`, `chisq_statistic`), `p.value` and, under `p_adjust`, `p_unadjusted`.
- `"tinytable"`, `"gt"`, `"flextable"`: the rendered table.
- `"excel"` / `"word"`: writes the file and returns its path invisibly.
- `"clipboard"`: copies the table and returns the display `data.frame` invisibly.

The default object and the two data frames carry the whole note as printed, its lead-in included, in `attr(x, "note")`. The default object also keeps the title the console prints, in `attr(x, "title")`. tidy() reports the proportions between 0 and 1.

What the columns are

n is the *observed* count – the number of rows in the sample, not an estimated population size. % is the estimated percentage *within* its column: without by it is the distribution of the variable in the population, with by the distribution inside that domain. The table note gives the sample size and the estimated population together, because neither alone tells the reader what they are looking at.

`proportion_ci = TRUE` adds the interval of each percentage. `ci_method` chooses among the seven `survey::svyciprop()` offers; the default `"logit"` is bounded inside 0 to 100, which the Wald interval (`"mean"`) is not. The percentage itself always comes from `survey::svymean()`, so it does not move when `ci_method` does.

A percentage of 0 or 100 has no design-based interval: its estimated variance is zero. Whatever `ci_method`, such a cell gets the exact binomial (Clopper-Pearson) interval on the unweighted count of its column, with no degrees-of-freedom factor. This is the rule of Korn and Graubard (1998, p. 194) for an estimate of 0, which the NCHS data presentation standards for proportions apply to its complement as well (Parker et al. 2017, pp. 2-3). The note says so when such a cell is shown.

The test

`svychisq()` with `chisq_statistic = "F"` (the default) is the Pearson chi-square with the Rao-Scott second-order correction, referred to F(ndf, `survey::degf(design)`). It is survey's own default and the one Stata's `svy: tabulate` reports. The design-based test has no exact counterpart,

so the switch to Fisher's exact test that `test` makes in `table_categorical()` when an expected count is below 5 has no equivalent here.

It runs on the *complete cases* of the two variables, and on their observed levels: a `(Missing)` row and a declared-but-unobserved level are descriptive, and neither belongs to the null hypothesis. The p-value is therefore the same whether `drop_na` shows those rows or removes them, and the intervals beside it describe the same domain – the two families test the same table.

`"Chisq"` shows the p-value only: survey adjusts the statistic in the `"F"` branch and only the p-value in the `"Chisq"` one, so the statistic there is not the one the p-value came from. `"Wald"`, `"adjWald"` and `"saddlepoint"` are available; `"lincom"` and `"wls-score"` are refused, the first because its integration is documented as failing in the far tail (`?pchisqsum`), the second because it has no reporting convention here.

Groups with no data

As in `table_categorical()`, a variable with no observed value in a group of `by` that holds sampled rows, typically a question not asked in that group, has no data there, which is not a missing answer. Its cells in that group show an en dash (`NA` in the data frames, in tidy() and in `as_structured()`), and its rows there are left out of everything computed for that variable: the missing values the note counts, the `Total` block, the `(Missing)` row and the test. A variable left with fewer than two groups with data, or observed at one level, is not tested: its `p` cell shows the dash, the undefined cell. The note says "No data for X in G.", in one sentence for the variables with no data in the same groups ("No data for X and Y in G."). Without `by`, a variable with no observed value at all keeps its declared levels (its one row when it declares none) with the dash, and the note says "No data for X."

Stability

This function is **stabilizing** in the sense `?spicy` defines: the names of its design-specific arguments may still be tightened before 1.0 – with a `NEWS.md` entry – but the behavior does not change silently. It was experimental through 0.13; the shape of the table has held across the cycle, so it now moves on the parent family's clock rather than its own. The numbers themselves are survey's and do not move with it.

What is absent, and why

`weights` and `rescale` (the weighting *is* the design). `correct` (Yates), `simulate_p` and `simulate_B`, which have no meaning once the reference distribution is Rao-Scott's. `percent`, not yet: a row percentage is estimated within its level, a domain of its own, so every level of every variable would bring its own degrees of freedom, intervals and design effects. And the association measures: Cramér's V, phi, tau-b/c, gamma, Somers' D and lambda have no established design-based variance, and the intervals `table_categorical()` gives them assume simple random sampling. The design-based measure of association here is the Rao-Scott test in the `p` column; for an effect size, model it with `table_regression(survey::svyglm(...))`.

Multiple-comparison adjustment

The p-values of a descriptive table describe: they test no prespecified hypothesis. STROBE (explanation of item 14) and the SAMPL guidelines advise against significance tests in descriptive and baseline tables, hence the default `p_adjust = "none"`.

When a journal or a protocol asks for adjusted p-values anyway, `p_adjust` applies `stats::p.adjust()` to the family of every test in the table: one p-value per variable (one per block in `table_outcome()`), never the rows of one variable, which share one test. `"holm"` suits a confirmatory set of comparisons, `"BH"` a screening one; `table_regression()` lists the methods. A test that returned no p-value stays out of the family.

The adjustment runs before any formatting, so the `p` column and its `<.001` threshold show the adjusted values, and the table note gives the number of tests and the method. The raw p-values stay in the object: a `p_unadjusted` column next to the p-value column of the returned data frames, and `col_meta$p$p_unadjusted` (with the method in `p_adjust` and the family size in `p_adjust_m`) in `as_structured()`. Without `by`, or with `p_value = FALSE`, the table shows no p-value to adjust, and a `p_adjust` other than `"none"` is an error.

References

Korn, E. L., & Graubard, B. I. (1998). Confidence intervals for proportions with small expected number of positive counts estimated from survey data. *Survey Methodology*, 24(2), 193-201.

Parker, J. D., Talih, M., Malec, D. J., et al. (2017). *National Center for Health Statistics data presentation standards for proportions*. Vital and Health Statistics, Series 2, No. 175.

See Also

`table_categorical()` for the data-frame sibling, `table_continuous_svy()` for continuous variables.

Examples

```
data(api, package = "survey")
dclus1 <- survey::svydesign(
  id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc
)
table_categorical_svy(dclus1, select = c(stype, awards))
table_categorical_svy(dclus1, select = stype, by = sch.wide)
table_categorical_svy(
  dclus1,
  select = stype,
  proportion_ci = TRUE,
  deff = TRUE
)
```

`table_continuous` *Continuous summary table*

Description

Computes descriptive statistics (mean, SD, min, max, confidence interval of the mean, *n*) for one or many continuous variables selected with tidyselect syntax.

With by, produces grouped summaries and reports a group-comparison *p*-value by default (Welch test; change via `test`). Additional inferential output is opt-in: test statistics (`statistic`) and effect sizes (`effect_size` / `effect_size_ci`). Set `p_value = FALSE` to suppress the *p*-value column. Without by, produces one-way descriptive summaries.

Multiple output formats are available via `output`: a printed ASCII table (`"default"`), a plain `data.frame` (`"data.frame"`, the displayed table with its numbers, or `"long"`, every statistic of each variable and group; see *Value*), or publication-ready tables (`"tinytable"`, `"gt"`, `"flextable"`, `"excel"`, `"clipboard"`, `"word"`).

This is the descriptive companion to `table_continuous_lm()`. The two functions share their layout, alignment, and reporting precision so descriptive and model-based analyses of the same data look uniform side by side – with one exception, documented under `align`: only `table_continuous()` carries `align` into the `excel` output. Use `table_continuous_lm()` when you need robust SE, weighted contrasts, fitted means, or covariate adjustment.

Usage

```
table_continuous(
  data,
  select = tidyselect::everything(),
  by = NULL,
  exclude = NULL,
  regex = FALSE,
  drop_na = TRUE,
  weights = NULL,
  rescale = getOption("spicy.rescale", FALSE),
  test = c("welch", "student", "nonparametric"),
  p_value = NULL,
  p_adjust = "none",
  statistic = FALSE,
  show_n = TRUE,
  show_columns = NULL,
  effect_size = c("none", "auto", "hedges_g", "eta2", "r_rb", "epsilon2"),
  effect_size_ci = FALSE,
  smd = FALSE,
  ci = TRUE,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  ci_level = 0.95,
  digits = 2,
  effect_size_digits = 2,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
```

```
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  verbose = FALSE,
  user_na = TRUE,
  style = NULL
)
```

Arguments

data	A data.frame.
select	Columns to include. If regex = FALSE, use tidyselect syntax or a character vector of column names (default: tidyselect::everything()). If regex = TRUE, provide a regular expression pattern (character string). The by and weights columns are not part of the selection: everything() and the other helpers skip them, and naming one in select, or giving its position, is an error. A name that is not a column of data is an error (class spicy_missing_column), and so is a selection with no numeric column.
by	Optional grouping column. Accepts an unquoted column name or a single character column name. Coerced to factor for grouping; non-numeric grouping columns (factor, character, logical) are supported as-is. Factor levels keep their declared order, which is the way to choose another order; any other by is sorted, as in table_categorical(): text in the C locale (capitals before lower case, the same on every machine), FALSE before TRUE, numbers ascending, whatever the order of the rows. The group of missing values comes last. A haven labelled by names its groups by their value labels, as the rows of a labelled variable of table_categorical() name its levels ("[1] Women"), in the order of its labels attribute when every observed value carries a label and by ascending code otherwise; declared missing values follow user_na as usual.
exclude	Columns to exclude. Supports tidyselect syntax and character vectors of column names.
regex	Logical. If FALSE (the default), uses tidyselect helpers. If TRUE, the select argument is treated as a regular expression.
drop_na	Logical. Controls how missing values in the by column are handled, under the rule of the family: a table that counts shows the missing values by default (a frequency table reports how many answers are missing, as a "(Missing)" row and, under by, a "(Missing)" column: table_categorical()); a table that computes removes them by default and counts them in the note. This table computes: a mean is computed on the observed values, so the NAs of each summarized variable are always excluded from that variable's statistics, and the exclusion is disclosed in a table note ("Missing values removed: ...") rather than silent. If TRUE (the default), rows with NA in by are removed from the grouped summaries and counted in the same note, by after the variables ("Missing values

removed: age (3), sex (2)."): a missing by takes the row out of the whole table, a missing value only out of its own variable's statistics. If FALSE, rows with NA in by form a dedicated "(Missing)" group – the field convention for descriptive tables (gtsummary's "Unknown" row; see the Epidemiologist R Handbook, Descriptive tables) – while the group-comparison test and effect size are still computed on the observed groups only (show the missing, test the observed, matching table_categorical()); a level of by displayed with the label of that group is an error, as there. Under drop_na = TRUE, a by that no observation holds is an error (class spicy_invalid_data). Ignored (with a warning) when by is not used.

weights — Optional case weights: an unquoted column name, a character column name, or a numeric vector of length nrow(data). Weights must be non-negative and finite; rows with NA or zero weight leave every statistic (including Min / Max) and NA weights are disclosed in the table note. The weighted formulas follow the frequency-expansion convention – see the **Weights** section. The weighted table names its weights in the note ("Statistics weighted by ..."). Group tests and effect sizes are not computed under weights: use table_continuous_lm() for weighted comparisons.

rescale — Logical. If TRUE, weights are first normalized so that they sum to the number of observations used for each variable – the same rescale grammar as table_categorical(). This is the sampling-weights reading: results become invariant to the scale of the weights, and the weighted SD then equals Stata's [aweight] / survey::svyvar() value exactly. FALSE uses the weights as given (frequency reading). The default reads options(spicy.rescale), FALSE when it is unset.

test — Character. Statistical test to use when comparing groups. One of "welch" (default), "student", or "nonparametric".

- "welch": Welch *t*-test (2 groups) or Welch one-way ANOVA (3+ groups). Does not assume equal variances.
- "student": Student *t*-test (2 groups) or classic one-way ANOVA (3+ groups). Assumes equal variances.
- "nonparametric": Wilcoxon rank-sum / Mann–Whitney *U* (2 groups) or Kruskal–Wallis *H* (3+ groups).

Used whenever by is supplied (since p_value defaults to TRUE in that case) or when statistic = TRUE or an effect_size is requested. Ignored when by is not used, or when all three display toggles are turned off. A variable with no variance has no test, whichever is chosen, with a warning (class spicy_undefined_stat); so has a variable whose groups are each constant on their own value (a residual variance of zero) under "welch" and "student", while the rank tests, which read the order of the values, run. Its test cells (test, p-value, effect size) show the undefined mark, as do those of any comparison that cannot run (too few observations in a group, a test that fails, a non-finite effect size).

p_value — Logical or NULL. If TRUE and by is used, adds a *p*-value column from the test specified by test. When NULL (the default), the *p*-value is shown automatically whenever by is supplied, and hidden otherwise. Pass p_value = FALSE to suppress the column explicitly. Ignored when by is not used.

`p_adjust` Multiple-comparison adjustment of the *p*-value column, applied to the family of every test in the table (one *p*-value per variable). One of `"none"` (default), `"holm"`, `"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"` / `"fdr"`, or `"BY"`, as in `table_regression()`; delegated to `stats::p.adjust()`. Needs `by` and the *p*-value column. See *Multiple-comparison adjustment*.

`statistic` Logical. If `TRUE` and `by` is used, the test statistic is shown in an additional column (e.g., `t(df) = ...`, `F(df1, df2) = ...`, `W = ...`, or `H(df) = ...`). Both `p_value` and `statistic` are independent; either or both can be enabled. Defaults to `FALSE`. Ignored when `by` is not used.

`show_n` Logical. If `TRUE`, includes an unweighted `n` column in the printed ASCII table and in every rendered output (`tinytable`, `gt`, `flextable`, `word`, `excel`, `clipboard`). Set to `FALSE` to drop the `n` column structurally from those outputs (no empty placeholder, no spanner). The `n` column is always present in `output = "long"` for downstream programmatic access. Defaults to `TRUE`. Ignored (with a warning) when `show_columns` is supplied.

`show_columns` Statistics to display, as a character vector of tokens or a named list of such vectors (one per variable). `NULL` (the default) keeps the historical display: mean, SD, min, max, the mean CI (see `ci`) and `n` (see `show_n`). See the "Choosing the statistics" section for the token vocabulary, the per-variable form, and the test that follows a median.

`effect_size` Effect-size measure to include in the rendered outputs. One of:

- `"none"` (default): no effect-size column.
- `"auto"`: auto-select the canonical measure for the chosen `test` and group count – Hedges's g (parametric, 2 groups), eta squared (parametric, 3+ groups), rank-biserial r (nonparametric, 2 groups), epsilon squared (nonparametric, 3+ groups).
- `"hedges_g"`: Hedges's g (bias-corrected standardized mean difference, 2 groups, parametric). CI via the Hedges & Olkin normal approximation.
- `"eta2"`: Eta squared (η^2, parametric ANOVA-style `SS_between / SS_total`). CI via inversion of the noncentral F distribution.
- `"r_rb"`: Rank-biserial r from the Wilcoxon / Mann-Whitney statistic (2 groups, nonparametric). CI via Fisher z-transform.
- `"epsilon2"`: Epsilon squared (ε^2) from the Kruskal-Wallis statistic (3+ groups, nonparametric). CI via percentile bootstrap (2 000 replicates).

A squared measure ends in 2, as in `table_continuous_lm()` and `table_regression()`: `"eta_sq"` and `"epsilon_sq"`, the spellings before 0.14.0, are an error (class `spicy_defunct`) that gives the new one. A logical value is an error of the same class: use `"auto"` for `TRUE` and `"none"` for `FALSE`. Explicit choices are validated against the active `test` and the number of groups; an incompatible request (e.g. `"eta2"` with two groups, or `"hedges_g"` with `test = "nonparametric"`) triggers an actionable error. Ignored when `by` is not used.

`effect_size_ci` Logical. If `TRUE`, appends the confidence interval of the effect size in brackets (e.g., `g = 0.45 [0.22, 0.68]`), under a header that states its level (`ES [95% CI]`). Implies a non-`"none"` effect size: if left at the default `effect_size = "none"`, the function warns and promotes `effect_size` to `"auto"` so the requested CI can be shown. Defaults to `FALSE`.

smd — Logical. If TRUE, adds an SMD column holding the standardized mean difference between the two groups of by, the balance diagnostic of the Table 1 literature. Requires exactly two groups; signed, group 1 minus group 2 in the order the table displays them; no confidence interval and no p-value, by design. It is independent of p_value and of effect_size: turning it on turns nothing else off. Rounded with effect_size_digits. See the "Standardized mean difference" section below. Defaults to FALSE.

ci — Logical. If TRUE, includes the mean confidence interval columns (<level>% CI LL / <level>% CI UL) and their spanner in the printed ASCII table and in every rendered output (tinytable, gt, flextable, word, excel, clipboard). Set to FALSE to drop both columns and the CI spanner structurally from those outputs (no empty placeholders, no border lines under an empty header). The CI bounds are always present as ci_lower / ci_upper in output = "long" for downstream programmatic access. Defaults to TRUE. The CI level is taken from ci_level. Ignored (with a warning) when show_columns is supplied.

labels — How each variable is named in the table: in the rows, in the title, and in the notes. Four forms:

- labels = FALSE shows the column name.
- labels = TRUE shows the column's label attribute (as set by haven, labelled, or label_from_names()) when it is present, not NA, and not empty, and the column name otherwise.
- labels = c(bmi = "Body mass index"), a named character vector, gives the columns it names these labels. Every other column follows options(spicy.var_labels): its name when the option is unset or FALSE, its label attribute when it is TRUE.
- labels = var_labels(bmi = "BMI (kg/m2)") gives the columns it names these labels, and every other column its label attribute, whatever the option says. var_labels() alone is labels = TRUE.

A name that matches no column of the table warns (class spicy_unknown_name) and is left out, so one vector can still serve every table of a project. The default reads options(spicy.var_labels), and a TRUE or FALSE typed in the call wins over it. Any other value, NULL included, is an error. A variable label documents the data file; a published table carries the wording you set here. note_labels gives a variable a shorter label in the title and the notes.

note_labels — The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one labels gives it: a named character vector, note_labels = c(sex = "sex at birth"), or a var_labels() object, which reads as one. labels can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of labels, and so do the title and the notes for a variable note_labels does not name. A name that matches no column warns and is left out, as in labels, and NULL (the default) renames nothing. A note label that another variable of the table shows is an error (class spicy_label_collision): choose another one, or relabel the other variable.

ci_level — Confidence level for the mean confidence interval (default: 0.95): a number strictly between 0 and 1.

`digits` Number of decimal places of the statistics on each variable's own scale: the mean, SD, median, IQR, quartiles, minimum and maximum, and the confidence intervals of the mean and of the median. One non-negative whole number sets them for every variable (default: `2`), and the weighted *n* with them. A vector named by column, not by label, gives each variable it names its own, `digits = c(height = 0, bmi = 1)`; the other variables and the weighted *n* keep the default, the `digits` of a `spicy_style()` in force included. A name that matches no variable warns and is left out, as in `labels`. A fraction is an error, as for every `*_digits` argument. The other numbers keep their own decimals: *p*-values (`p_digits`), effect sizes and the SMD (`effect_size_digits`), test statistics (two decimals) and counts (none).

`effect_size_digits` Number of decimal places for effect-size values in formatted displays (default: `2`).

`p_digits` Integer >= 1. Number of decimal places used to render *p*-values in the `p` column (default: `3`, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: `p_digits = 3` prints `.045` and `<.001`; `p_digits = 4` prints `.0451` and `<.0001`; `p_digits = 2` prints `.05` and `<.01`. Useful for genomics / GWAS contexts with very small *p*-values, or for journals using a coarser convention. Leading zeros are always stripped, following APA convention.

`decimal_mark` The decimal mark, a single character: `"."` (the default) or `","` in most styles; the Lancet style of `spicy_style()` uses the middle dot.

`align` Horizontal alignment of numeric columns in the printed ASCII table and in the `tinytable`, `gt`, `flextable`, `word`, `excel`, and `clipboard` outputs. The first column (`Variable`) and `Group` (when present) are always left-aligned. One of:

- `"decimal"` (default): align numeric columns on the decimal mark, the standard scientific-publication convention used by SPSS, SAS, and LaTeX `siunitx`. Numeric cells are pre-padded with figure-spaces (U+2007, digit-width) so every string in a column has the same width with the decimal mark at the same internal position; centering those uniform-width strings then stacks the decimal points vertically. The same pad-then-center strategy is applied on every rendering engine (`gt`, `tinytable`, `flextable`, `word`, ASCII print) for a homogeneous rendering, matching `table_regression()` and `table_continuous_lm()`. The `clipboard` output is delimited text meant to be parsed rather than read at a fixed width, so its cells travel unpadded (a padded number pastes as text next to an unpadded number).
- `"center"`: center-align all numeric columns.
- `"right"`: right-align all numeric columns.

`"center"` and `"right"` reach the `excel` output too. `"decimal"` does not: Excel cells are written unpadded, because cell-string padding does not align decimals under a proportional font, so the workbook keeps the engine's own convention instead – counts and the *p*-value right-aligned, the other numeric columns centered. Same default and same three values as `table_continuous_lm()`, whose `excel` output still uses that convention at every `align`.

`note` The note under the table. `NULL` (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under

the jama and lancet themes; see note_label in spicy_style()), then, in a table_regression() with stars, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks ("\n") included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (note = "Note. Data weighted."), and the legend of the stars stays below it. "" keeps that legend alone, and no note at all without stars. FALSE removes the note from every output, the legend included.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an empty string is an error, and so is note_add with note = FALSE, which leaves no note to add to.

output — Output format. One of:

- "default": an ASCII table object, printed when the call is bare.
- "data.frame": the displayed table with its numbers, one row per printed row.
- "long": one row per (variable x group) (or one row per variable when by is not used), with every statistic the table computes.
- "tinytable" (requires tinytable)
- "gt" (requires gt)
- "flextable" (requires flextable)
- "excel" (requires openxlsx2)
- "clipboard" (requires clipr)
- "word" (requires flextable and officer)

excel_path — File path for output = "excel", ending in ".xlsx", in a directory that exists.

excel_sheet — Sheet name for output = "excel". NULL (the default) uses "Descriptives".

clipboard_delim — Delimiter for output = "clipboard" (default: "\t"). A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path — File path for output = "word", ending in ".docx", in a directory that exists.

verbose — Logical. If TRUE, a message (class spicy_info) names the non-numeric columns left out (default: FALSE).

user_na — Logical. If TRUE (the default), declared missing values never reach the numeric summaries: they are excluded like NA and disclosed in the table note (Declared missing values removed: ...); declared-missing by values form no group. If FALSE, the declared codes are summarized as ordinary numbers. See the "Declared missing values" section of freq().

style — A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes

DEFAULTS – any argument you pass explicitly wins over it. Set `options(spicy.style = )` for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; `?spicy_style` lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

Depends on `output`:

- `"default"`: a `data.frame` of class `"spicy_continuous_table"` / `"spicy_table"` carrying the rendering metadata as attributes. Its columns are internal and can change without notice: read the numbers from `"data.frame"`, `"long"`, `broom::tidy()`, `broom::glance()` or `as_structured()`. The object is returned visibly, so a bare `table_continuous(...)` call auto-prints the styled ASCII table at the console while `t <- table_continuous(...)` stays silent (print `t` to display the table). The object can be re-coerced via `as.data.frame.spicy_continuous_table()` or piped into `broom::tidy()` / `broom::glance()`.
- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per printed row, the key columns `variable` (the column name), `label` (the label displayed for it) and, with `by`, `group` and `.row_role` (`"group"`, or `"missing"` for the group of missing `by` values), then the printed columns under their names (`"M"`, `"SD"`, `"Min"`, `"Max"`, `"95% CI LL"`, `"95% CI UL"`, `"n"`, `"Test"`, `"p"`, `"ES"`, `"SMD"`, the columns `show_columns` asks for). A cell that prints several numbers keeps the first in its column and the others beside it under their names in the long output: `df` and `df.residual` after `"Test"`, `es_ci_lower` and `es_ci_upper` after `"ES"` when its interval is printed, `q1` and `q3` after `"Mdn [Q1, Q3]"`.
- `"long"`: a plain `data.frame` with one row per `(variable x group)` and columns `variable`, `label`, `group` and `.group_role` (the role of `group`: `"group"`, or `"missing"` for the group of missing `by` values) when `by` is used, `mean`, `sd`, `min`, `max`, `ci_lower`, `ci_upper`, `median`, `q1`, `q3`, `iqr`, `med_ci_lower`, `med_ci_upper`, `n`, `weighted_n`. Every statistic is computed whatever `show_columns` displays. When `by` is used, the comparison columns follow (populated on the first row of each variable block only):
  - `test_type` – test identifier (e.g., `"welch_t"`, `"welch_anova"`, `"student_t"`, `"anova"`, `"wilcoxon"`, `"kruskal"`), with `statistic`, `df` and `df.residual` (the degrees of freedom of a t statistic in `df`, those of an F statistic in `df` and `df.residual`, as `broom::glance()` names them), when the test ran (for `p_value`, `statistic` or an effect size).
  - `p.value` – with the *p*-value only, and `p_unadjusted` under `p_adjust`.
  - `es_type` – effect-size identifier (`"hedges_g"`, `"eta2"`, `"r_rb"`, or `"epsilon2"`), `es_value`, `es_ci_lower`, `es_ci_upper` – the effect size and its interval.
  - `smd_type`, `smd_value` – the standardized mean difference; `smd_type` names the kernel the value came from: `"continuous"` here.

  The effect-size and SMD columns are there whether or not they were requested, `NA` when not, so the schema a pipeline indexes into does not move with an argument (the `weighted_n` rule); the test columns follow `p_value`.
- `"tinytable"`: a `tinytable` object.
- `"gt"`: a `gt_tbl` object.
- `"flextable"`: a `flextable` object.

- "excel" / "word": writes to disk and returns the file path invisibly.
- "clipboard": copies the table and returns the display data.frame invisibly.

The note opens with the sample size, as in table_categorical(): "N = 159." or, with by, "N = 159: Women (n = 80), Men (n = 79).", the rows the table describes and those of each group (the group of missing by values included), a row missing by under drop_na = TRUE or a weight left out, rows rather than weights under weights ("N = 60 (unweighted)."), the complete cases of each variable in its n column.

The missing-value disclosure (values excluded from the summaries, and rows removed for a missing by value under drop_na = TRUE) travels with the table on every route, not just the console: "default" prints it under the ASCII table, "tinytable" / "gt" / "flextable" / "word" carry it as a table note, "excel" writes it below the body, and the default object and the data frames of "data.frame" / "long" keep the whole note as printed, its lead-in included, in attr(x, "note") (NULL when there is nothing to say), so a pipeline that renders the numbers itself can still state what left the table. The default object also keeps the title it prints in attr(x, "title"). On the "tinytable" route the note is set one size down; options(spicy.note_style) governs that (see table_regression()).

The Excel sheet carries the same title the console prints on its first row; the table itself starts on row 3.

Choosing the statistics

show_columns selects which statistics the table displays. The tokens, and the column each one produces:

Token	Column	Statistic
"m"	M	mean
"sd"	SD	standard deviation
"med"	Mdn	median (stats::median())
"iqr"	IQR	interquartile *width*, Q3 - Q1
"med_iqr"	Mdn [Q1, Q3]	median and the interquartile *interval*, in one compact column
"q1" / "q3"	Q1 / Q3	first / third quartile
"min" / "max"	Min / Max	extremes
"ci"	<level>% CI LL / UL	*t* confidence interval of the mean
"med_ci"	Mdn <level>% CI LL / UL	exact confidence interval of the median
"n"	n	valid observations
"weighted_n"	Weighted n	sum of weights (requires weights)

The column is the header the table prints. In output = "data.frame", as.data.frame() and as_structured(), the columns are named by their keys, the same in every display language and style: the default header in English (M; Mdn, Mdn [Q1, Q3], Mdn <level>% CI LL / UL for the median), where the "jama" and "lancet" styles print Mean and Median. The note under the table defines the abbreviations these headers show, as the style in force asks (see abbreviations in spicy_style()).

Quartiles use stats::quantile()'s default type 7. "iqr" is the width (one number, the rank mirror of SD); "med_iqr" shows the interval with its bounds. Columns appear in the canonical order of the table above, whatever order they were written in.

Weights

With `weights`, every displayed statistic uses the **frequency-expansion** convention: for integer weights each statistic equals its unweighted version computed on the data with every row repeated `w` times (`rep(x, w)`), exactly; with all weights equal to 1 every statistic equals its unweighted sibling. The formulas, with $W = \sum w_i$:

- mean: $\sum w_i x_i / W$;
- SD: $\sqrt{\sum w_i (x_i - \bar{x}_w)^2 / (W - 1)}$;
- quantiles: type-7 positions on the cumulative-weight scale (the `Hmisc::wtd.quantile()` default algorithm);
- CI of the mean: $\bar{x}_w \pm t_{W-1}\, s_w / \sqrt{W}$;
- `n` counts the rows used; `"weighted_n"` reports W.

These are the conventions of `Hmisc::wtd.mean()` / `wtd.var()` / `wtd.quantile()` (defaults), `matrixStats::weightedSd()` and `DescTools::Quantile()`, and – for integer weights – of Stata's `[fweight]` and SPSS's `WEIGHT BY`. With `rescale = TRUE` the weights are normalized to sum to the number of observations first, which makes every result invariant to the scale of the weights and makes the SD equal Stata's `[aweight]` / `survey::svyvar()` value – the reading appropriate for sampling weights. Weighted-quantile conventions genuinely differ across software (Stata interpolates nowhere, SAS refuses analytic-weighted quantiles, the survey package offers twelve rules); spicy states its rule here rather than leaving it implicit.

Two deliberate refusals: the `"med_ci"` token (an order-statistic interval with no weighted version) and, under `by`, the group tests and effect sizes – a *t*-test printed next to weighted descriptives would silently be unweighted. Set `p_value = FALSE` for weighted descriptives by group, or use `table_continuous_lm()` with `weights` for weighted comparisons. Note that `table_continuous_lm()`'s residual SD answers a different question (model-based, precision-weight convention) and is not expected to match the descriptive SD here.

A named list applies a different selection to each variable, with `.default` covering the variables it does not name – the case of a table where a skewed variable must be reported as a median while the others keep the mean:

```
show_columns = list(
  mvpa    = c("med_iqr", "n"),
  sitting = c("med_iqr", "n"),
  .default = c("m", "sd", "n")
)
```

A name that is no variable of the table warns and is left out, as in `labels`.

The table's columns are the union of the requested tokens; a cell of a column the variable did not ask for is left blank (structurally empty, not an en dash, which is reserved for an undefined statistic).

The table tests what it shows. A variable displaying a median without a mean takes the rank-based test – Wilcoxon rank-sum for two groups, Kruskal-Wallis beyond – and the rank effect size (rank-biserial *r*, ε^2) when `effect_size` is `"auto"`. The switch is per variable, so a mixed table carries a rank test on its median rows and Welch on its mean rows, and the table note names which test each variable carries. An explicit `test` is sovereign: it applies to every variable, with a warning naming the ones displayed as medians.

"med_ci" is the exact order-statistic (sign-test) confidence interval: the tightest interval $[x_{(k)}, x_{(n-k+1)}]$ whose binomial coverage still reaches ci_level. It is distribution-free and deterministic – no bootstrap, no seed – and its coverage is at least nominal, the same convention as SAS PROC UNIVARIATE (CIPCTLDF) and DescTools::MedianCI(method = "exact"). Below about six observations no interval reaches the requested level; the cells then show an en dash rather than a false interval.

"ci" is the confidence interval *of the mean*: requested without "m" it is dropped with a warning pointing at "med_ci", and "med_ci" without a displayed median is dropped likewise. When show_columns is supplied it decides the n and CI columns on its own, and a contradictory show_n / ci is reported.

Tests

The omnibus test is computed only when by is supplied and at least two groups remain after dropping NAs, with every group contributing at least two observations. Choice of test family is driven by test (see the @param entry for the full dispatch and the underlying stats:: functions called). Whenever the table shows *p*-values, its note names the test behind them ("P-values: Welch t-test.", "P-value:" when the table shows one), each variable beside its own test when they differ. A test shown by its statistic alone (p_value = FALSE, statistic = TRUE) is named as the group comparison ("Group comparison: Welch t-test.").

No data in a group

A variable with no observed value in a group of by (every value missing there, typically a question not asked in that group) has no data in it, which is not a missing answer. Its cells in that group show an en dash, *n* included (NA in output = "long" and the typed view), its rows there are not counted in the missing values of the note, the comparison and the effect size use the groups with data (with fewer than two, there is no test and its cell is left blank), and the note says so ("No data for X and Y in G.", X and Y the labels of the variables with no data in group G). Without by, a variable with no observed value at all is shown the same way ("No data for X."). A declared level of by that no row holds keeps its count of zero, and the group of missing by values is never such a group.

For model-based contrasts (heteroskedasticity-consistent SE, cluster-robust SE, weighted contrasts, fitted means, covariate adjustment), use table_continuous_lm().

Effect sizes

See @param effect_size for the dispatch table (canonical measure for each (test, n_groups) combination) and the validation rules applied to explicit requests.

Confidence intervals (enabled with effect_size_ci = TRUE) use noncentral *F* inversion for η^2, the Hedges-Olkin normal approximation for g, the Fisher z-transform for r, and percentile bootstrap (2,000 replicates) for ε^2. The bootstrap bounds depend on the random number generator state: call set.seed() before the table for reproducible ε^2 intervals (the other three CIs are closed-form and deterministic).

For Cohen's d, Hays's ω^2, and Cohen's f^2 (derived from a fitted, possibly weighted lm()), use the model-based companion table_continuous_lm().

Standardized mean difference

smd = TRUE adds an SMD column with the balance diagnostic of the Table 1 literature, in Austin's form (Austin 2009, *Stat Med* 28:3083-3107; Austin 2011, *Multivar Behav Res* 46:399-424):

$$\text{SMD} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(s_1^2 + s_2^2)/2}}$$

The denominator is the root **mean of the two group variances**, each at $n - 1$, not the degrees-of-freedom pooled SD. At **equal** group sizes those two denominators are the same, so the SMD is exactly Cohen's *d*; at unequal sizes they part company (on a 4-versus-3 split the SMD is -0.51 and *d* is -0.54).

`effect_size = "hedges_g"`, two columns to the left, is a third number: *g* applies the small-sample correction *J* on top of *d*, so it **never** equals the SMD. At equal group sizes the ratio g/SMD is exactly *J* – 0.80 at *n* = 3 per group, 0.96 at *n* = 10 – approaching 1 only as the sample grows. Read each for what it is; do not recompute one from the other. (The divergence is nameable upstream: `cobalt::col_w_smd(s.d.denom = "pooled")` reproduces this column, `s.d.denom = "hedges"` reproduces `hedges_g`.)

Conventions, all deliberate:

- **Signed**, group 1 minus group 2 in the order the table displays the groups – the two groups sit side by side, so a bare magnitude would make the reader re-derive a direction the row already gives. (`tableone` publishes the magnitude; `cobalt` and `arsenal` sign it the other way, guessing the second level as "treated".) The threshold in the table note is read on $|\text{SMD}|$; the column keeps the sign. No conditional formatting: spicy never highlights a threshold.
- **No confidence interval and no p-value, ever.** The SMD is a descriptive diagnostic; attaching an interval to it reintroduces the test reasoning the balance literature asks the reader to drop. This is not a missing feature.
- **Exactly two groups.** A `by` with three or more is refused rather than averaged over pairs: an average has no published reading, and it can sit under the usual threshold while one pair sits well over it.
- **Complete cases on the observed groups.** A `drop_na = FALSE` "(Missing)" group is displayed and never enters the diagnostic, exactly as the test and the effect size behave.
- **Independent of `p_value`.** Turning the SMD on turns nothing else off. The balance-table idiom is `smd = TRUE, p_value = FALSE`; you have to write both.

Under `weights`, the means and variances are the weighted ones the `M` and `SD` columns already display – the frequency convention of the *Weights* section, from the same producer, so the column cannot contradict its neighbors. One consequence follows and is intended: a frequency weight is a number of copies, so the weighted SMD is **not invariant to the scale of the weights** (multiplying every weight by ten moves it, as it moves the `SD` column). `rescale = TRUE` normalizes the weights to sum to *n*, restores scale invariance, and is the form to use for sampling weights until the dedicated survey-design functions land.

A cell is an en-dash when the diagnostic applies but cannot be estimated: both groups constant at different values (an infinite standardized distance, disclosed by a warning), or a group with too little data to have a variance (silent – the `SD` cell beside it already says so). Two groups constant at the *same* value are perfectly balanced and print `0.00`.

Display conventions

Decimal alignment, *p*-value formatting, and required suggested packages per output engine are documented under `@param align`, `@param p_digits`, and `@param output` respectively.

Non-numeric columns are silently dropped (set verbose = TRUE to see which columns were excluded). When a constant column is passed, its statistics are reported exactly: SD is 0.00 and the CI degenerates to [m, m]. An en-dash cell appears only when a statistic is undefined (fewer than two valid observations).

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as 8 = Don't know or 9 = Refused that the source file marks as missing while keeping them distinct from a plain NA. Two kinds of declaration exist: na_values / na_range metadata on haven::labelled_spss() vectors, and tagged missing values created by haven::tagged_na() (the Stata .a, .b, ... convention). haven::read_sav() keeps the na_values / na_range declaration only with user_na = TRUE: its default turns the declared codes into NA on import.

spicy honors the declaration by default (user_na = TRUE): declared missing values are excluded from every statistic exactly like NA – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. freq() lists each observed declared value as its own row of the Missing block, with its value label; cross_tab(), table_categorical(), and table_continuous() disclose the exclusion in the table note (Declared missing values remo varlist() counts them as missing in N_valid / NAs / N_distinct while still listing the declared codes in Values; code_book() counts them in n_declared_missing and flags them in its values table.

Every function involved offers the same escape hatch: set user_na = FALSE to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine NAs either way; for them, user_na = FALSE only collapses the per-tag breakdown back into the regular NA count.

Multiple-comparison adjustment

The p-values of a descriptive table describe: they test no prespecified hypothesis. STROBE (explanation of item 14) and the SAMPL guidelines advise against significance tests in descriptive and baseline tables, hence the default p_adjust = "none".

When a journal or a protocol asks for adjusted p-values anyway, p_adjust applies stats::p.adjust() to the family of every test in the table: one p-value per variable (one per block in table_outcome()), never the rows of one variable, which share one test. "holm" suits a confirmatory set of comparisons, "BH" a screening one; table_regression() lists the methods. A test that returned no p-value stays out of the family.

The adjustment runs before any formatting, so the p column and its <.001 threshold show the adjusted values, and the table note gives the number of tests and the method. The raw p-values stay in the object: a p_unadjusted column next to the p-value column of the returned data frames, and col_meta$p$p_unadjusted (with the method in p_adjust and the family size in p_adjust_m) in as_structured(). Without by, or with p_value = FALSE, the table shows no p-value to adjust, and a p_adjust other than "none" is an error.

See Also

table_outcome() for the transposed shape – ONE continuous outcome across the levels of SEVERAL groupings, one block of rows per grouping. Several outcomes across one grouping is this function; one outcome across one or more groupings is that one; table_continuous_lm() for the

model-based companion (heteroskedasticity-consistent SE, cluster-robust SE, weighted contrasts, fitted means); table_categorical() for categorical variables; freq() for one-way frequency tables; cross_tab() for two-way cross-tabulations.

Other spicy tables: table_categorical(), table_continuous_lm()

Examples

```
# --- Basic usage ------------------------------------------------------

# Default: ASCII console table.
table_continuous(
  sochealth,
  select = c(bmi, wellbeing_score)
)

# Grouped by education (Welch p-value added by default).
table_continuous(
  sochealth,
  select = c(bmi, wellbeing_score),
  by = education
)

# Test statistic alongside the p-value.
table_continuous(
  sochealth,
  select = c(bmi, wellbeing_score),
  by = education,
  statistic = TRUE
)

# --- Choosing the statistics ------------------------------------------

# Median and interquartile range instead of mean and SD.
table_continuous(
  sochealth,
  select = c(bmi, wellbeing_score),
  show_columns = c("med_iqr", "n")
)

# Median with its exact (order-statistic) confidence interval.
table_continuous(
  sochealth,
  select = bmi,
  show_columns = c("med", "iqr", "med_ci", "n")
)

# One selection per variable. A skewed variable that a scoring
# protocol requires in median and IQR (the IPAQ case) sits next to
# variables kept in mean and SD; each row is tested the way it is
# displayed, and the note says so.
table_continuous(
  sochealth,
```

```
  select = c(bmi, life_sat_health, wellbeing_score),
  by = sex,
  show_columns = list(
    life_sat_health = c("med_iqr", "n"),
    .default = c("m", "sd", "n")
  )
)

# --- Effect sizes ------------------------------------------------------

# Auto-selected effect size with confidence interval (Hedges's g for
# binary `by`, eta squared for k > 2).
table_continuous(
  sochealth,
  select = wellbeing_score,
  by = sex,
  effect_size = "auto",
  effect_size_ci = TRUE
)

# Explicit effect-size measure.
table_continuous(
  sochealth,
  select = wellbeing_score,
  by = education,
  effect_size = "eta2",
  effect_size_ci = TRUE,
  effect_size_digits = 3
)

# --- Selection helpers -------------------------------------------------

# Regex selection.
table_continuous(
  sochealth,
  select = "^life_sat",
  regex = TRUE
)

# Pretty labels keyed by column name.
table_continuous(
  sochealth,
  select = c(bmi, life_sat_health),
  labels = c(
    bmi = "Body mass index",
    life_sat_health = "Satisfaction with health"
  )
)

# --- Output formats ----------------------------------------------------

# The rendered outputs below all wrap the same call:
#   table_continuous(sochealth,
```

```
#                    select = c(bmi, wellbeing_score),
#                    by = sex)
# only `output` changes. Assign each result to a variable -- some
# engines auto-print as a console-friendly text fallback inside
# the `?` help viewer.

# Wide / long data.frame (synonyms): one row per (variable x group).
table_continuous(
  sochealth,
  select = c(bmi, wellbeing_score),
  by = sex,
  output = "data.frame"
)


# Rendered HTML / docx objects -- best viewed inside a
# Quarto / R Markdown document or a pkgdown article.
if (requireNamespace("tinytable", quietly = TRUE)) {
  tt <- table_continuous(
    sochealth, select = c(bmi, wellbeing_score), by = sex,
    output = "tinytable"
  )
}
if (requireNamespace("gt", quietly = TRUE)) {
  tbl <- table_continuous(
    sochealth, select = c(bmi, wellbeing_score), by = sex,
    output = "gt"
  )
}
if (requireNamespace("flextable", quietly = TRUE)) {
  ft <- table_continuous(
    sochealth, select = c(bmi, wellbeing_score), by = sex,
    output = "flextable"
  )
}

# Excel and Word: write to a temporary file.
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  tmp <- tempfile(fileext = ".xlsx")
  table_continuous(
    sochealth, select = c(bmi, wellbeing_score), by = sex,
    output = "excel", excel_path = tmp
  )
  unlink(tmp)
}
if (
  requireNamespace("flextable", quietly = TRUE) &&
    requireNamespace("officer", quietly = TRUE)
) {
  tmp <- tempfile(fileext = ".docx")
  table_continuous(
    sochealth, select = c(bmi, wellbeing_score), by = sex,
    output = "word", word_path = tmp
```

```
  )
  unlink(tmp)
}


## Not run:
# Clipboard: writes to the system clipboard.
table_continuous(
  sochealth, select = c(bmi, wellbeing_score), by = sex,
  output = "clipboard"
)

## End(Not run)
```

`table_continuous_lm` *Continuous-outcome linear-model table*

Description

Builds publication-ready summary tables from a series of linear models for one or many continuous outcomes selected with tidyselect syntax.

A single focal predictor is supplied with `by`; each selected numeric outcome is fit as `lm(outcome ~ by, ...)`, optionally extended with additive covariates via `covariates` and case weights via `weights`. Categorical `by` produces model-based estimated marginal means by level (covariate-adjusted via `adjustment` when covariates are present), plus an optional single difference for dichotomous predictors. Numeric `by` produces the slope and its confidence interval.

Inference adapts via `vcov`: classical OLS, `"HC0"`-`"HC5"` (heteroscedasticity-consistent), `"CR0"`-`"CR3"` (cluster-robust, requires `cluster`), or `"bootstrap"` / `"jackknife"` resampling. Effect sizes (Cohen's *d* `"cohens_d"`, Hedges's *g* `"hedges_g"`, Hays's `"omega2"`, Cohen's `"f2"`) are reported with optional noncentral *t* / *F* confidence intervals via `effect_size_ci`, and adapt under covariate adjustment (see `effect_size`).

Multiple output formats are available via `output`: a printed ASCII table (`"default"`), a plain wide `data.frame` (`"data.frame"`), a raw long `data.frame` (`"long"`), or rendered outputs (`"tinytable"`, `"gt"`, `"flextable"`, `"excel"`, `"clipboard"`, `"word"`).

Usage

```
table_continuous_lm(
  data,
  select = tidyselect::everything(),
  by,
  covariates = NULL,
  adjustment = c("proportional", "balanced"),
  exclude = NULL,
  regex = FALSE,
  weights = NULL,
```

```
  vcov = c("classical", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "CR0", "CR1",
    "CR2", "CR3", "bootstrap", "jackknife"),
  cluster = NULL,
  boot_n = 1000,
  contrast = c("auto", "none"),
  statistic = FALSE,
  p_value = NULL,
  show_n = TRUE,
  show_weighted_n = FALSE,
  effect_size = c("none", "f2", "cohens_d", "hedges_g", "omega2"),
  effect_size_ci = FALSE,
  r2 = c("r2", "adj_r2", "none"),
  ci = TRUE,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  ci_level = 0.95,
  digits = 2,
  fit_digits = 2,
  effect_size_digits = 2,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
  note_add = NULL,
  output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  verbose = FALSE,
  user_na = TRUE,
  style = NULL
)
```

Arguments

data	A data.frame.
select	Outcome columns to include. If regex = FALSE, use tidyselect syntax or a character vector of column names (default: tidyselect::everything()). If regex = TRUE, provide a regular expression pattern (character string). The by column and the covariates are not part of the selection: everything() and the other helpers skip them, and naming one in select is an error. A name that is not a column of data is an error (class spicy_missing_column), and so is a selection with no numeric column.
by	A single predictor column. Accepts an unquoted column name or a single character column name. The predictor can be:

- **numeric** (continuous): treated as a continuous regressor. The table reports the slope of by and its CI from `lm(y ~ by, ...)`.
- **factor** or **ordered factor**: treated as categorical. Level order is preserved as declared; the **first level** is the reference for the displayed contrast (R's default treatment-contrast convention). An ordered factor is refit as an unordered factor with the same level order, so the model uses treatment contrasts – not the polynomial contrasts (`contr.poly`) `lm()` would apply by default. The ordering only determines the display order and the reference level; per-level means, the displayed difference, and its CI are identical to those of an unordered factor with the same levels.
- **character**: coerced to factor with its values sorted in the C locale (capitals before lower case, the same on every machine), the order of the groups in the whole family (`table_categorical()`, `table_continuous()`, ...): the first value in that order is the reference level, whatever the order of the rows. To control the reference level, supply by as an explicit factor with the desired level ordering (e.g. via `forcats::fct_relevel()` or `factor(..., levels = ...)`).
- **logical**: coerced to factor with levels `"FALSE"`, `"TRUE"` (in that order, since `FALSE < TRUE`). The reference level is `"FALSE"`, so the binary contrast is `TRUE` minus `FALSE`.
- **haven labelled with value labels** (an SPSS / Stata import): treated as categorical, one level per observed code named by its value label (`"[1] Women"`, as the rows of a labelled variable of `table_categorical()` are), in the order of its `labels` attribute when every observed value carries a label and by ascending code otherwise (the first level is the reference; the model needs a fixed level order), the order in which `table_continuous()` and `table_categorical()` display the same groups. A labelled vector without value labels is treated as a continuous regressor. Declared missing values follow `user_na` as usual.

Rows with `NA` in by are excluded from the analytic sample for each outcome (NAs in `y` and `weights` are also excluded; see Details), the rule of a table that computes (see `drop_na` in `table_continuous()`); this table has no `drop_na`. The note lists by after the outcomes and covariates ("Missing values removed: age (3), sex (2)."): a missing by or covariate takes the row out of the model of every outcome, a missing outcome only out of its own.

An outcome with no observed value in a level of a categorical by (typically a question not asked in that group) has no data there: it is fit on the other levels, the cells of that level show a dash, its rows are not counted as missing values, and the note says so. With fewer than two levels with data, the outcome has no model and no test.

`covariates` Optional additive covariates to adjust each per-outcome linear model for. Accepts a tidyselect expression (e.g. `covariates = c(age, sex)`, `covariates = tidyselect::all_of(cov_vec)`, `covariates = tidyselect::starts_with("control_")`) or a literal character vector of column names. Each covariate must be numeric, integer, logical, factor, or character. A variable cannot be both outcome and adjustment: the covariates are left out of the outcome selection (`select = everything()` skips them), and naming one in `select`, or giving its position,

raises an error; a covariate that equals by raises an error too (a variable cannot be both predictor and adjustment).

When non-empty, each model is fitted as `lm(y ~ by + cov1 + cov2 + ...)` and the reported estimate / SE / p-value / CI on by are covariate-adjusted via the focal coefficient. For categorical by, the displayed `emmean` is the covariate-adjusted estimated marginal mean – see `adjustment` for the choice of estimand (G-computation by default vs. equal-weight averaging). The omnibus test of by is the Wald *F* restricted to the focal coefficients (computed via `sandwich` / `clubSandwich` for HC* / CR* mode), so adding covariates does not contaminate the omnibus statistic with covariate contributions. Effect sizes adapt automatically – see `effect_size`.

v1 supports additive covariates only. Formula syntax with interactions or transforms (`covariates = ~ age * sex`, `covariates = ~ I(age^2)`) is reserved for a future release; passing a formula raises a `spicy_unsupported` error with a migration hint.

Rows with `NA` in any covariate are dropped from the analytic sample for each outcome (complete-cases per outcome, matching the existing by / `weights` NA handling).

`adjustment` How the covariate-adjusted estimated marginal means (the `emmean` / `emmean_se` / `emmean_ci_*` columns) are computed when `covariates` is non-empty (typed without covariates, it is ignored with a warning, class `spicy_ignored_arg`). One of:

- `"proportional"` (the default; matches Stata `margins` and `marginaleffects::avg_predictions()` G-computation on the observed sample. For each focal level of by, the model predicts at every observation with by set to that level (covariates kept at their observed values), and the predictions are averaged. Population-weighted by construction – the empirical joint distribution of covariates is the reference. Under `weights`, the averaging uses the case weights (the Stata `margins` convention after a weighted regression; equivalent to `marginaleffects::avg_predi = )`), so the reference distribution is the weighted one. Best when the goal is "what is the predicted mean in *this* population if everyone had by = `lvl`".
- `"balanced"` (matches `emmeans::emmeans()` default and the SPSS UNIANOVA EMMEANS / SAS LSMEANS conventions): synthetic grid of factor-covariate level combinations x numeric covariates fixed at their sample mean, with each grid cell weighted equally. Treats the design as if covariates were balanced – the "marginal mean assuming a balanced design" estimand. Best when the goal is to report a covariate-purified comparison independent of the empirical covariate distribution.

Both methods reduce to the same linear-contrast formula `emmean = avg_row %*% beta` and inherit the spicy variance pipeline (HC* / CR* / bootstrap / jackknife). They give the same answer when there are no covariates, and also when all covariates are numeric (fixed at their means either way). The two estimands diverge when a factor, character, or logical covariate has non-uniform observed proportions: those are levels to balance, and `"balanced"` weights them equally (a logical is a two-level factor to `lm()`, balanced FALSE/TRUE – the `emmeans` convention).

`exclude` Columns to exclude from `select`. Supports tidyselect syntax and character vectors of column names.

`regex` Logical. If `FALSE` (the default), uses tidyselect helpers. If `TRUE`, the `select` argument is treated as a regular expression.

`weights` Optional case weights. Accepts:

- `NULL` (default): an ordinary unweighted `lm()` is fit.
- an **unquoted numeric column name** present in `data`.
- a **single character column name** present in `data`.
- a **numeric vector of length** `nrow(data)` evaluated in the calling environment.

Validation: weights must be finite, non-negative, and contain at least one positive value (otherwise the function errors). Rows with `NA` in `weights` are excluded from the analytic sample for each outcome, alongside rows with `NA` in `y` or `by`. A weights column is not an outcome: `select = everything()` skips it, and naming it in `select`, or giving its position, is an error. When supplied, weights are passed to `lm(..., weights = ...)`, so coefficients become weighted least-squares estimates and R^2, adjusted R^2, and the four effect sizes are computed from the corresponding weighted sums of squares (see the *Weights* section in Details).

`vcov` Variance estimator used for standard errors, confidence intervals, and Wald test statistics. One of:

- `"classical"` (default): the ordinary OLS/WLS variance from `vcov(lm)`, which assumes homoscedastic errors.
- `"HC0"`: the original Eicker–White heteroskedasticity-consistent sandwich estimator (White 1980), with no finite-sample correction.
- `"HC1"`: HC0 multiplied by `n / (n - p)` (MacKinnon and White 1985). Matches Stata's `, robust` default.
- `"HC2"`: residuals divided by `sqrt(1 - h_ii)` (MacKinnon and White 1985).
- `"HC3"`: residuals divided by `(1 - h_ii)` (MacKinnon and White 1985). A common default for small to moderate samples (Long and Ervin 2000).
- `"HC4"`: leverage-adaptive variant designed for influential observations (Cribari-Neto 2004).
- `"HC4m"`: refinement of HC4 with a modified leverage exponent (Cribari-Neto and da Silva 2011).
- `"HC5"`: alternative leverage-adaptive variant designed for leveraged data (Cribari-Neto, Souza and Vasconcellos 2007).
- `"CR0"`, `"CR1"`, `"CR2"`, `"CR3"`: cluster-robust sandwich estimators for non-independent observations (Liang and Zeger 1986); requires `cluster`. `"CR2"` is the modern default (Bell and McCaffrey 2002; Pustejovsky and Tipton 2018), with Satterthwaite degrees of freedom for inference; the fractional df is reported in the long output (in `df` beside a *t*, in `df.residual` beside an *F*) and in the `t(df)` / `F(df1, df2)` test header. `"CR1"` applies the G/(G-1) correction only; Stata's `, vce(cluster id)` uses the larger G(N-1)/((G-1)(N-p)) factor with t(G-1) inference (clubSandwich's `"CR1S"`; exposed for `lm` fits in `table_regression()`, not here), so `"CR1"` does not reproduce Stata.

Cluster-robust variants are dispatched to `clubSandwich::vcovCR()` and inference uses `clubSandwich::coef_test()` / `clubSandwich::Wald_test()`; install `clubSandwich` to use them.

- `"bootstrap"`: nonparametric (resampling cases) or cluster bootstrap variance, depending on whether `cluster` is supplied (Davison and Hinkley 1997; Cameron, Gelbach and Miller 2008). The number of replicates is set by `boot_n`. Inference is asymptotic (`z` for single contrasts, `chi^2(q)` for the global Wald test); CIs are Wald-type around the point estimate.
- `"jackknife"`: leave-one-out variance, or leave-one-cluster-out when `cluster` is supplied (Quenouille 1956; MacKinnon and White 1985). Inference is asymptotic (`z` / `chi^2(q)`).

The `HC*` variants are computed via `sandwich::vcovHC()`. Coefficients (means, contrasts, slopes), R^2, and the standardized effect sizes (`f2`, `cohens_d`, `hedges_g`, `omega2`) are point estimates from the OLS/WLS fit and are not affected by `vcov`; only their standard errors, CIs, and the test statistic of the contrast change.

`cluster` Cluster identifier for cluster-aware variance estimators. Required when `vcov` is one of the `CR*` variants; optional and triggers a cluster bootstrap or leave-one-cluster-out jackknife when `vcov` is `"bootstrap"` / `"jackknife"`; forbidden for the other (independent-observation) variants. Accepts:

- `NULL` (default): no cluster structure.
- an unquoted column name in `data`.
- a single character column name in `data`.
- a one-sided formula naming a column in `data` (`~ region`), the `sandwich` / `fixest` convention shared with `table_regression()`.
- an atomic vector of length `nrow(data)` evaluated in the calling environment (factor, character, integer, etc.).

Rows with `NA` in `cluster` are excluded from the analytic sample for each outcome (alongside rows with `NA` in `y`, `by`, or `weights`). A cluster column is not an outcome: `select = everything()` skips it, and naming it in `select`, or giving its position, is an error. At least two distinct non-missing cluster values are required. Multi-way clustering (a list / data.frame of multiple cluster vectors) is not supported; use `sandwich::vcovCL()` or `clubSandwich::vcovCR()` directly on the fitted model for that case.

`boot_n` Number of bootstrap replicates used when `vcov = "bootstrap"`: a positive whole number, as in `table_regression()` and `table_multiresponse()` (a fraction is an error). Defaults to `1000`, the default of `table_regression()`, which computes the same bootstrap variance. That variance needs at least 10 valid replicates: a smaller `boot_n` is an error. Larger values reduce Monte-Carlo error in the bootstrap variance; typical values for inference are `500`-`2000`. Typed with another `vcov`, it is ignored with a warning (class `spicy_ignored_arg`).

`contrast` Contrast display for categorical predictors (typed with a numeric `by`, it is ignored with a warning, class `spicy_ignored_arg`). One of:

- `"auto"` (default): show a single reference contrast, the second level minus the first, only when `by` has exactly two non-empty levels. The reference level is the **first level of the factor** (R's default treatment-contrast convention, `getOption("contrasts")[1]`). To change which level acts as the

reference, re-level by upstream (for example with `forcats::fct_relevel()` or `stats::relevel()`).

- `"none"`: suppress the contrast column for categorical predictors. Level-specific means are still displayed.

`statistic` Logical. If `TRUE`, includes a test-statistic column in the wide and rendered outputs. Defaults to `FALSE`.

`p_value` Logical or `NULL`. If `TRUE`, includes a `p` column in the wide and rendered outputs. `NULL` (the default) shows it whenever `by` is supplied, as in the rest of the family; `by` is required here, so `NULL` means `TRUE`. Pass `p_value = FALSE` to suppress it: `output = "long"` then drops `p.value`, and `test_type`, `statistic`, `df` and `df.residual` too unless `statistic = TRUE`; `tidy()` and `glance()` keep theirs, `NA`.

`show_n` Logical. If `TRUE`, includes an unweighted `n` column in the wide and rendered outputs. Defaults to `TRUE`.

`show_weighted_n`

Logical. If `TRUE` and `weights` is supplied, includes a `Weighted n` column equal to the sum of case weights in the analytic sample. Defaults to `FALSE`.

`effect_size` Character. Effect-size column to include in the wide and rendered outputs (a logical is an error, class `spicy_defunct`). An estimator named after its author carries the name, as `"hedges_g"` does in `table_continuous()`: `"d"` and `"g"`, the spellings before 0.14.0, are an error of the same class that gives the new one. One of:

- `"none"` (the default): no effect-size column.
- `"f2"`: Cohen's `f^2 = R^2 / (1 - R^2)`. Defined for any predictor type. Familiar from Cohen (1988); standard input for a-priori power analysis. Note that for a single-predictor model, f^2 is a monotone transform of R^2 and adds no information beyond it.
- `"cohens_d"`: Cohen's `d = beta_hat / sigma_hat`, where `beta_hat` is the model coefficient (the displayed difference) and `sigma_hat` is the residual standard deviation from the fitted model. Defined only when `by` has exactly two non-empty levels; otherwise the function errors. The sign matches the displayed difference, the second level minus the first.
- `"hedges_g"`: Hedges's `g = J * d` with the small-sample correction `J = 1 - 3 / (4 * df_resid - 1)`. Same domain as `"cohens_d"`.
- `"omega2"`: Hays's omega squared, a bias-corrected estimator of the population variance explained, less optimistic than R^2 for small samples. Defined for any predictor type and truncated at 0.

When `weights` is supplied, `"cohens_d"`, `"hedges_g"`, and `"omega2"` are derived from the weighted least-squares fit (using weighted sums of squares and the model's weighted residual standard deviation), keeping them consistent with the weighted contrast and its CI shown in the table. All effect sizes are point estimates derived from the OLS/WLS fit and are **not** affected by `vcov`.

Under covariate adjustment (`covariates` non-empty):

- `"f2"` and `"omega2"` become the **partial** f^2 / partial ω^2, derived from the partial F of `by` (the Type-II test of `by` after all covariates, equal to `stats::drop1()`

in this additive setting) – the correctly-defined effect size when the model is adjusted. For numeric by, partial f^2 equals the squared partial correlation of by with the outcome, divided by `(1 - r^2_partial)`.

- `"cohens_d"` and `"hedges_g"` raise a `spicy_unsupported` error: Cohen's d and Hedges's g have no canonical extension to adjusted models (the pooled SD is undefined under adjustment). Use `"f2"` or `"omega2"` instead – both generalize via partial F.

`effect_size_ci` Logical. If `TRUE` and `effect_size != "none"`, adds a confidence interval for the effect size derived from inversion of the appropriate noncentral distribution (noncentral t for `"cohens_d"` / `"hedges_g"`; noncentral F for `"omega2"` / `"f2"`). The CI level is taken from `ci_level`. In the long output (`output = "long"`), the bounds are always present in `es_ci_lower` / `es_ci_upper` (numeric). In the wide raw output (`output = "data.frame"`), the bounds appear under the same names, `es_ci_lower` / `es_ci_upper` (numeric). In the printed ASCII table and rendered outputs (`"tinytable"`, `"gt"`, `"flextable"`, `"word"`, `"excel"`, `"clipboard"`), the effect-size column shows the value followed by the CI in brackets (e.g. `0.18 [0.07, 0.30]`), under a header that states its level (`d [95% CI]`). Defaults to `FALSE`. When `effect_size = "none"`, this argument is ignored with a warning.

`r2` Character. Fit statistic to include in the wide and rendered outputs. One of:

- `"r2"` (default): the model R^2 (`summary(lm)$r.squared`).
- `"adj_r2"`: adjusted R^2, penalizing for `df_effect` relative to the residual degrees of freedom.
- `"none"`: omit the fit-statistic column.

When `weights` is supplied, R^2 and adjusted R^2 are the weighted least-squares versions reported by `summary(lm(..., weights = ...))`.

`ci` Logical. If `TRUE`, includes contrast confidence-interval columns in the wide and rendered outputs when a single contrast is shown. Defaults to `TRUE`.

`labels` How each variable is named in the table: in the rows, in the title, and in the notes, the covariates included. Four forms:

- `labels = FALSE` shows the column name.
- `labels = TRUE` shows the column's `label` attribute (as set by `haven`, `labelled`, or `label_from_names()`) when it is present, not `NA`, and not empty, and the column name otherwise.
- `labels = c(bmi = "Body mass index")`, a named character vector, gives the columns it names these labels. Every other column follows `options(spicy.var_labels)`: its name when the option is unset or `FALSE`, its `label` attribute when it is `TRUE`.
- `labels = var_labels(bmi = "BMI (kg/m2)")` gives the columns it names these labels, and every other column its `label` attribute, whatever the option says. `var_labels()` alone is `labels = TRUE`.

A name that matches no column of the table warns (class `spicy_unknown_name`) and is left out, so one vector can still serve every table of a project. The default reads `options(spicy.var_labels)`, and a `TRUE` or `FALSE` typed in the call wins over it. Any other value, `NULL` included, is an error. A variable label documents

the data file; a published table carries the wording you set here. `note_labels` gives a variable a shorter label in the title and the notes.

`note_labels` The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one `labels` gives it: a named character vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and `NULL` (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`ci_level` Confidence level for coefficient and model-based mean intervals (default: `0.95`): a number strictly between 0 and 1.

`digits` Number of decimal places of the numbers on each outcome's own scale: the means of the groups, adjusted or not, the difference or the slope, and its confidence interval. One non-negative whole number sets them for every outcome (default: `2`), and the test statistic and the weighted *n* with them. A vector named by column, not by label, gives each outcome it names its own, `digits = c(height = 0, bmi = 1)`; the other outcomes, the test statistic and the weighted *n* keep the default, the `digits` of a `spicy_style()` in force included. A name that matches no outcome warns and is left out, as in `labels`. A fraction (`2.9`) is an error, as for `fit_digits` and `effect_size_digits`. The *p*-value (`p_digits`), the R^2 (`fit_digits`), the effect size (`effect_size_digits`) and the counts keep their own decimals.

`fit_digits` Number of decimal places for model-fit columns (R^2 or adjusted R^2) in wide and rendered outputs (default: `2`).

`effect_size_digits`
Number of decimal places for the effect-size column (`f2`, `d`, `g`, or `omega2`) in wide and rendered outputs (default: `2`).

`p_digits` Integer >= 1. Number of decimal places used to render *p*-values in the `p` column (default: `3`, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: `p_digits = 3` prints `.045` and `<.001`; `p_digits = 4` prints `.0451` and `<.0001`; `p_digits = 2` prints `.05` and `<.01`. Useful for genomics / GWAS contexts where adjusted *p*-values can be very small, or for journals using a coarser convention. Leading zeros are always stripped, following APA convention. Values below `1` and fractions (`3.7`) raise an error.

`decimal_mark` The decimal mark, a single character: `"."` (the default) or `","` in most styles; the Lancet style of `spicy_style()` uses the middle dot.

`align` Horizontal alignment of numeric columns in the printed ASCII table and in the `tinytable`, `gt`, `flextable`, `word`, and `clipboard` outputs. The first column (`Variable`) is always left-aligned. One of:

- `"decimal"` (default): align numeric columns on the decimal mark, the standard scientific-publication convention used by SPSS, SAS, and LaTeX

siunitx. Numeric cells are pre-padded with figure-spaces (U+2007, digit-width) so every string in a column has the same width with the decimal mark at the same internal position; centering those uniform-width strings then stacks the decimal points vertically. The same pad-then-center strategy is applied on every rendering engine (gt, tinytable, flextable, word, ASCII print) for a homogeneous rendering, matching table_regression(). The clipboard output is delimited text meant to be parsed rather than read at a fixed width, so its cells travel unpadded (a padded number pastes as text next to an unpadded number).

- "center": center-align all numeric columns.
- "right": right-align all numeric columns.

The excel output uses the engine's default alignment in any case: cell-string padding does not align decimals under proportional fonts, and writing raw numbers with a numeric format would require a separate refactor.

note — The note under the table. NULL (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the jama and lancet themes; see note_label in spicy_style()), then, in a table_regression() with stars, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks ("\n") included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (note = "Note. Data weighted."), and the legend of the stars stays below it. "" keeps that legend alone, and no note at all without stars. FALSE removes the note from every output, the legend included.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an empty string is an error, and so is note_add with note = FALSE, which leaves no note to add to.

output — Output format. One of:

- "default": an ASCII table object, printed when the call is bare
- "data.frame": the displayed table with its numbers, one row per outcome
- "long": one row per level of by and outcome (one row per outcome with a numeric by), with every estimate
- "tinytable" (requires tinytable)
- "gt" (requires gt)
- "flextable" (requires flextable)
- "excel" (requires openxlsx2)
- "clipboard" (requires clipr)
- "word" (requires flextable and officer)

excel_path — File path for output = "excel", ending in ".xlsx", in a directory that exists.

excel_sheet — Sheet name for output = "excel". NULL (the default) uses "Linear models".

clipboard_delim
: Delimiter for output = "clipboard" (default: "\t"). A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path
: File path for output = "word", ending in ".docx", in a directory that exists.

verbose
: Logical. If TRUE, a message (class spicy_info) names the non-numeric columns left out of the outcomes (default: FALSE).

user_na
: Logical. If TRUE (the default), declared missing values in the outcomes, in by, and in covariates are treated as missing and excluded from the fitted models (reflected in the per-group n). If FALSE, the declared codes enter the fits as ordinary values. See the "Declared missing values" section of freq().

style
: A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set options(spicy.style =) for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; ?spicy_style lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

Depends on output:

- "default": the compute frame with class "spicy_continuous_lm_table" / "spicy_table", carrying the rendering metadata as attributes (the by column and its label as group_var and group_label, the title as printed as title). The object is returned visibly, so a bare table_continuous_lm(...) call auto-prints the styled ASCII table at the console while t <- table_continuous_lm(...) stays silent (print t to display the table).
- "data.frame": the displayed table with its numbers, what as.data.frame() of the default object returns: one row per outcome, the key columns variable (the column name of the outcome) and label (the label displayed), then the printed columns under their names – the means (categorical by) or the slope (numeric by), the contrast and its interval bounds, the test statistic, p, the fit statistic, the effect size (with es_ci_lower / es_ci_upper beside it when effect_size_ci = TRUE), n, and Weighted n.
- "long": one block per outcome, one row per level of by (one row, estimate_type "slope" and level NA, with a numeric by), with every estimate: identification (variable, label, predictor_type, predictor_label, level, reference), fitted means and their CI (emmean, emmean_se, emmean_ci_lower, emmean_ci_upper), contrast or slope estimates and CI (estimate_type, estimate, estimate_se, estimate_ci_lower, estimate_ci_upper), inferential output (test_type, statistic, df, df.residual, p.value: the *F* of a factor has both degrees of freedom, the *t* of a contrast or a slope has its own in df and none in df.residual; see p_value), effect size with its CI (es_type, es_value, es_ci_lower, es_ci_upper), fit (r.squared, adj.r.squared, on the first row of an outcome), the size of the model (nobs, weighted_nobs, repeated on each row of an outcome), and boot_n_valid, the number of bootstrap replicates whose refit gave every coefficient (NA without vcov = "bootstrap").
- "tinytable": a tinytable object.
- "gt": a gt_tbl object.

- `"flextable"`: a `flextable` object.
- `"excel"` / `"word"`: writes to disk and returns the file path invisibly.
- `"clipboard"`: copies the wide table and returns it invisibly.

The Excel sheet carries the same title the console prints on its first row; the table itself starts on row 3, and the note lines sit below the body. The default object and the data frames of `"data.frame"` and `"long"` carry the whole note as printed, its lead-in included, in `attr(x, "note")` (`NULL` when there is nothing to say).

If no numeric outcome columns remain after applying `select`, `exclude`, and `regex`, the function stops with an error (class `spicy_invalid_input`), as the other tables of the family do.

Model and outputs

`table_continuous_lm()` is designed for article-style reporting around a single focal predictor: one model per selected continuous outcome, fitted as `lm(outcome ~ by, ...)` and optionally extended with case `weights` and additive `covariates` (`lm(outcome ~ by + cov1 + ...)`). For categorical `by`, the reported means are model-based fitted means (or covariate-adjusted estimated marginal means; see `adjustment`) for each level, and contrasts come from the same fitted linear model. For an unweighted `lm(y ~ factor)` with classical variance and no covariates, the fitted means coincide numerically with empirical subgroup means; the *model-based* qualifier matters because (a) under `weights` the means become weighted least-squares estimates, (b) their CIs derive from the model `vcov` (classical, `HC*`, `CR*`, bootstrap or jackknife), (c) under `covariates` they become adjusted marginal means, and (d) tests, *p*-values and effect sizes all come from the same fitted model, keeping the table internally consistent.

Compared with `table_continuous()`, this function is the model-based companion: choose it when you want heteroskedasticity-consistent standard errors (`vcov = "HC*"`), model fit statistics, or case weights via `lm(..., weights = ...)`. Because the function exists to report a fitted model, its inferential output is on by default: the *p*-value is shown (`p_value = NULL`, the default, shows it with `by`, which is required here) and `r2 = "r2"`; set `p_value = FALSE` or `r2 = "none"` to suppress them.

An outcome with no variance in its fitted sample (every observation the same value) leaves nothing to infer, as in `table_continuous()`: its test, its interval, its effect size and its R^2 are undefined cells (`NA` in the long output and in `glance()`), its means and its contrast or slope (0) are shown, and a warning (class `spicy_undefined_stat`) names it. So does a residual variance of zero, an outcome constant within each group of a categorical `by` or an exact linear function of `by` and the covariates: its means (adjusted for the covariates), its contrast or slope and its R^2 of 1 are shown, and its test, interval and effect size are undefined. Every cell of an outcome applies to it, so a cell with no value shows the undefined mark, never a blank: a saturated fit (one observation per group) has no interval, test or effect size, and a model that could not be fit has no cell at all.

Effect sizes

Effect size is selected explicitly via `effect_size` (defaults to `"none"`). All variants are derived from the same fitted model as the displayed coefficients, R^2, and CIs, so the effect size stays internally consistent with the rest of the table.

- `"f2"`: Cohen's `f^2 = R^2 / (1 - R^2)` (Cohen 1988). Defined for any predictor type. For a single-predictor model, f^2 is a monotone transform of R^2 and adds no information beyond it; its primary use is in *a priori* power analysis (e.g. G*Power).

- "cohens_d", "hedges_g": standardized mean difference (Cohen's d or Hedges's g), defined only when by has exactly two non-empty levels. d = beta_hat / sigma_hat with sigma_hat = summary(fit)$sigma (the pooled within-group SD for the unweighted two-group case); g = J * d with J = 1 - 3 / (4 * df_resid - 1) (Hedges and Olkin 1985). The sign matches the displayed difference, the second level minus the first. For published reports of two-group comparisons, g is the convention recommended by Hedges and Olkin (1985).
- "omega2": Hays's ω^2, computed from weighted sums of squares as (SS_effect - df_effect * MSE) / (SS_total + MSE) and truncated at 0 for small or null effects (Hays 1963; Olejnik and Algina 2003). Less biased than η^2 (which equals R^2 in this single-predictor design) and recommended for reporting variance explained in ANOVA-style designs (Olejnik and Algina 2003).

All four effect sizes are point estimates derived from the OLS/WLS fit and are **invariant to** vcov: choosing HC* changes the SE, CI, and test statistic of the contrast but not the standardized magnitude itself.

Under covariate adjustment (covariates non-empty), "f2" and "omega2" become the partial f^2 / partial ω^2 of by, derived from the partial F restricted to the focal term (the Type-II test of by after all covariates, equal to stats::drop1() in this additive setting). "cohens_d" and "hedges_g" raise a spicy_unsupported error: the pooled standard deviation has no canonical extension under adjustment, so Cohen's d and Hedges's g are undefined for adjusted models. See effect_size for the full dispatch.

Confidence intervals for the effect size are available via effect_size_ci = TRUE and use the modern noncentral-distribution inversion approach, the consensus standard in commercial statistical software (Stata esize / estat esize, SAS PROC TTEST and PROC GLM EFFECTSIZE 14.2+) and in mainstream R packages (effectsize, MOTE, TOSTER, effsize):

- "cohens_d", "hedges_g": noncentral t inversion (Steiger and Fouladi 1997; Goulet-Pelletier and Cousineau 2018). Empirical coverage is nominal across sample sizes (Fitts 2021), unlike the older Hedges-Olkin normal approximation which is biased for small samples. For Hedges's g the bounds inherit the J small-sample correction.
- "omega2", "f2": noncentral F inversion (Steiger 2004). Without covariates (model-level effect sizes), bounds are converted from the noncentrality parameter using omega^2 = ncp / (ncp + N) and f^2 = ncp / N respectively, with N = df1 + df2 + 1 (total sample size). Under covariate adjustment (partial effect sizes), the bounds use the partial transforms instead: partial f^2 = ncp / df2, and for partial omega^2 the inversion runs at the F-value equivalent of the omega squared point estimate with bounds ncp / (ncp + df2) – the effectsize::omega_squared() partial = TRUE convention, which is narrower than the partial eta squared CI because omega squared shrinks the point estimate.

For the weighted case, the CI uses raw (unweighted) group counts and df.residual(fit) = n - p, consistent with the WLS reporting convention (DuMouchel and Duncan 1983). For propensity-score balance assessment or complex-survey designs, dedicated packages (cobalt::bal.tab() for the Austin and Stuart 2015 formulation; survey for design-based effect sizes) are more appropriate.

Robust standard errors

When vcov is one of the HC* variants, the standard errors, CIs, and Wald test statistics use a heteroskedasticity-consistent sandwich estimator computed via sandwich::vcovHC() (Zeileis 2004), the canonical R implementation. For a brief guide:

- "HC0" is the original White (1980) form; "HC1" adds the n / (n - p) correction (MacKinnon and White 1985), Stata's , robust default.
- "HC2" and "HC3" use leverage-based residual rescalings (MacKinnon and White 1985); "HC3" is the sandwich::vcovHC() default for small to moderate samples (Long and Ervin 2000).
- "HC4" adapts the leverage exponent for influential observations (Cribari-Neto 2004); "HC4m" is a modified-exponent refinement (Cribari-Neto and da Silva 2011); "HC5" is an alternative leverage-adaptive variant (Cribari-Neto, Souza and Vasconcellos 2007).

When observations are not independent (repeated measurements per individual, students nested in classes, patients in hospitals, country-year panels), classical and HC* standard errors are biased downward. Use the CR* variants together with cluster = id_var to get cluster-robust inference (Liang and Zeger 1986). The implementation dispatches to clubSandwich::vcovCR() for the variance and to clubSandwich::coef_test() (single-coefficient, Satterthwaite *t*) and clubSandwich::Wald_test() (multi-coefficient Hotelling-T-squared with Satterthwaite df, "HTZ") for inference. "CR2" (Bell and McCaffrey 2002; Pustejovsky and Tipton 2018) is the modern recommended default; it generally produces fractional Satterthwaite degrees of freedom (in df beside a *t*, in df.residual beside an *F*), which the displayed t(df) / F(df1, df2) header renders to one decimal. "CR1" applies the G/(G-1) correction only – Stata's , vce(cluster id) uses the larger G(N-1)/((G-1)(N-p)) factor with t(G-1) inference (clubSandwich's "CR1S"; exposed for lm fits in table_regression(), not here), so "CR1" does not reproduce Stata. Effect sizes remain invariant to vcov (including CR*); only the SE, CI, test statistic, and degrees of freedom of the contrast change.

Two resampling-based estimators are also available without adding any dependency: vcov = "bootstrap" (nonparametric resampling-cases bootstrap; Davison and Hinkley 1997) and vcov = "jackknife" (leave-one-out delete-1; Quenouille 1956; MacKinnon and White 1985). Supplying cluster switches both to their cluster-aware variants (cluster bootstrap, Cameron, Gelbach and Miller 2008; leave-one-cluster-out jackknife). The number of bootstrap replicates is controlled by boot_n (default 1000); replicates that fail to fit on rank-deficient resamples are dropped, with an explicit warning if more than half fail. Fewer than 10 valid bootstrap replicates (or fewer than 2 jackknife leave-outs) raises spicy_resampling_failed rather than silently reporting a different variance estimator. Inference for both estimators is asymptotic (z for single-coefficient contrasts, chi^2(q) for the multi-coefficient global Wald test on k > 2 categorical predictors), reflected in the displayed test header. Use the bootstrap when the residual distribution is non-standard or the sample is small; use the jackknife as a closed-form, deterministic alternative.

R^2, adjusted R^2, and the effect sizes remain ordinary least-squares (or weighted least-squares) statistics regardless of vcov.

Weights

When weights is supplied, table_continuous_lm() fits weighted linear models via lm(..., weights = ...). Means become weighted least-squares estimates and contrasts and slopes are weighted. The fit statistics R^2 and adjusted R^2, as well as Hays's omega^2 and Cohen's f^2, use the corresponding **weighted sums of squares** from the WLS fit. Cohen's d and Hedges's g use the **WLS coefficient and the model's weighted residual standard deviation** (summary(fit)$sigma), which is the standard convention for case-weighted regression-style reporting (DuMouchel and Duncan 1983); the noncentral *t* CI for d / g uses the raw (unweighted) group counts and the residual degrees of freedom of the WLS fit (n - p). This case-weighted workflow is appropriate for weighted article tables, but is **not** a substitute for a full complex-survey design (see e.g. the survey package),

nor for propensity-score balance assessment under the Austin and Stuart (2015) convention (see e.g. `cobalt::bal.tab()`).

The `n` column always reports the unweighted analytic sample size for each outcome. When `show_weighted_n = TRUE`, an additional `Weighted n` column reports the sum of case weights in the same analytic sample.

Display conventions

With a categorical `by`, the headers have two levels (decked heads, APA Publication Manual, 7th ed., section 7.12): the mean symbol `M` spans the columns of the fitted means, each headed by its level alone, in reference-level order, in the console as in every rendering engine. For a dichotomous `by`, the contrast column is headed Δ alone, and a sentence of the note, after those on the test and the standard errors, gives the order of the subtraction: the second level minus the first, written with the minus sign. Its confidence interval keeps its spanner, `95% CI` over `LL` and `UL`, which the console folds into the headers `95% CI LL` and `95% CI UL`, as in the other tables. In `output = "data.frame"`, a mean keeps both levels in its name, `M (<level>)`, and the contrast is named Δ (the Greek capital delta, U+0394). For categorical predictors with more than two levels, no single contrast or contrast CI is shown in the wide outputs; instead, the table reports level-specific means plus the overall `F` test when `statistic = TRUE` (or `F(df1, df2)` when the degrees of freedom are constant across outcomes).

The note opens with the sample size, as in `table_categorical()`: "N = 159." for a numeric `by`, "N = 159: Women (n = 80), Men (n = 79)." for a categorical one, the rows with a `by` value, a row with a missing weight left out, rows rather than weights under `weights` ("N = 60 (unweighted)."), the complete cases of each model in its `n` column.

When `covariates` is non-empty, the note then names the covariates and the chosen estimand, e.g. `Adjusted for age, education (proportional).` With p-values, the note also names the test behind them, read off each outcome's model: `P-values: t-test from the linear model.` (the F-test for a `by` with more than two levels, the t-test of the slope for a numeric `by`), in the singular, `P-value:`, when the table shows one. A test shown by its statistic alone (`p_value = FALSE, statistic = TRUE`) is named as the group comparison, `Group comparison: t-test from the linear model.`, or, for a numeric `by`, as the slope test: `Slope test: t-test from the linear model.`

The rendering engines carry that same footer as a table note. On the `"tinytable"` route the note is set one size down; `options(spicy.note_style)` governs that (see `table_regression()`).

Optional output engines require the corresponding suggested packages:

- **tinytable** for `output = "tinytable"`
- **gt** for `output = "gt"`
- **flextable** for `output = "flextable"`
- **flextable** + **officer** for `output = "word"`
- **openxlsx2** for `output = "excel"`
- **clipr** for `output = "clipboard"`

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`.

Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

References

Austin, P. C., & Stuart, E. A. (2015). Moving towards best practice when using inverse probability of treatment weighting (IPTW) using the propensity score to estimate causal treatment effects in observational studies. *Statistics in Medicine*, **34**(28), 3661–3679. doi:10.1002/sim.6607

Bell, R. M., & McCaffrey, D. F. (2002). Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology*, **28**(2), 169–181.

Cameron, A. C., Gelbach, J. B., & Miller, D. L. (2008). Bootstrap-based improvements for inference with clustered errors. *Review of Economics and Statistics*, **90**(3), 414–427. doi:10.1162/rest.90.3.414

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum.

Cribari-Neto, F. (2004). Asymptotic inference under heteroskedasticity of unknown form. *Computational Statistics & Data Analysis*, **45**(2), 215–233. doi:10.1016/S01679473(02)003663

Cribari-Neto, F., Souza, T. C., & Vasconcellos, K. L. P. (2007). Inference under heteroskedasticity and leveraged data. *Communications in Statistics – Theory and Methods*, **36**(10), 1877–1888. doi:10.1080/03610920601126589

Cribari-Neto, F., & da Silva, W. B. (2011). A new heteroskedasticity-consistent covariance matrix estimator for the linear regression model. *AStA Advances in Statistical Analysis*, **95**(2), 129–146. doi:10.1007/s1018201001412

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511802843

DuMouchel, W. H., & Duncan, G. J. (1983). Using sample survey weights in multiple regression analyses of stratified samples. *Journal of the American Statistical Association*, **78**(383), 535–543. doi:10.1080/01621459.1983.10478006

Fitts, D. A. (2021). Expected and empirical coverages of different methods for generating noncentral *t* confidence intervals for a standardized mean difference. *Behavior Research Methods*, **53**(6), 2412–2429. doi:10.3758/s13428021015504

Goulet-Pelletier, J.-C., & Cousineau, D. (2018). A review of effect sizes and their confidence intervals, Part I: The Cohen's *d* family. *The Quantitative Methods for Psychology*, **14**(4), 242–265. doi:10.20982/tqmp.14.4.p242

Hays, W. L. (1963). *Statistics for Psychologists*. New York: Holt, Rinehart and Winston.

Hedges, L. V., & Olkin, I. (1985). *Statistical Methods for Meta-Analysis*. Orlando, FL: Academic Press.

Long, J. S., & Ervin, L. H. (2000). Using heteroscedasticity consistent standard errors in the linear regression model. *The American Statistician*, **54**(3), 217–224. doi:10.1080/00031305.2000.10474549

Liang, K.-Y., & Zeger, S. L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika*, **73**(1), 13–22. doi:10.1093/biomet/73.1.13

MacKinnon, J. G., & White, H. (1985). Some heteroskedasticity-consistent covariance matrix estimators with improved finite sample properties. *Journal of Econometrics*, **29**(3), 305–325. doi:10.1016/03044076(85)901587

Olejnik, S., & Algina, J. (2003). Generalized eta and omega squared statistics: Measures of effect size for some common research designs. *Psychological Methods*, **8**(4), 434–447. doi:10.1037/1082989X.8.4.434

Pustejovsky, J. E., & Tipton, E. (2018). Small-sample methods for cluster-robust variance estimation and hypothesis testing in fixed effects models. *Journal of Business & Economic Statistics*, **36**(4), 672–683. doi:10.1080/07350015.2016.1247004

Quenouille, M. H. (1956). Notes on bias in estimation. *Biometrika*, **43**(3/4), 353–360. doi:10.1093/biomet/43.34.353

Steiger, J. H. (2004). Beyond the *F* test: Effect size confidence intervals and tests of close fit in the analysis of variance and contrast analysis. *Psychological Methods*, **9**(2), 164–182. doi:10.1037/1082989X.9.2.164

Steiger, J. H., & Fouladi, R. T. (1997). Noncentrality interval estimation and the evaluation of statistical models. In L. L. Harlow, S. A. Mulaik, & J. H. Steiger (Eds.), *What if there were no significance tests?* (pp. 221–257). Mahwah, NJ: Lawrence Erlbaum.

White, H. (1980). A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica*, **48**(4), 817–838. doi:10.2307/1912934

Zeileis, A. (2004). Econometric computing with HC and HAC covariance matrix estimators. *Journal of Statistical Software*, **11**(10), 1–17. doi:10.18637/jss.v011.i10

See Also

`table_continuous()`, `table_categorical()`. For broader workflows on the same statistical building blocks: `sandwich::vcovHC()` (the canonical R implementation of the `HC*` sandwich estimators, used internally for `vcov = "HC*"`); `clubSandwich::vcovCR()`, `clubSandwich::coef_test()` and `clubSandwich::Wald_test()` (the canonical R implementation of cluster-robust variance and Satterthwaite-style inference, used internally for `vcov = "CR*"`); `effectsize::cohens_d()`, `effectsize::hedges_g()`, and `effectsize::omega_squared()` (alternative effect-size computations and CIs); `cobalt::bal.tab()` for propensity-score covariate balance with weighted standardized mean differences (Austin and Stuart 2015); the `survey` package for design-based inference on complex-survey samples.

Other spicy tables: `table_categorical()`, `table_continuous()`

Examples

```
# --- Basic usage -----------------------------------------------------------

# Default: ASCII table with model-based means, p, and R-squared.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex
)

# --- Effect sizes ----------------------------------------------------------

# Cohen's d (binary by required).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  effect_size = "cohens_d"
)

# Hedges's g with weighted analysis and weighted n column.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  weights = weight,
  statistic = TRUE,
  effect_size = "hedges_g",
  show_weighted_n = TRUE
)

# Hedges's g with noncentral t confidence interval (bracket notation).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  effect_size = "hedges_g",
  effect_size_ci = TRUE
)

# Cohen's f-squared alongside R-squared (familiar power-analysis effect size).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  effect_size = "f2"
)

# Hays's omega squared for a 3-level predictor ("cohens_d" and
# "hedges_g" need two groups).
table_continuous_lm(
  sochealth,
```

```
  select = c(wellbeing_score, bmi),
  by = education,
  effect_size = "omega2"
)

# --- Robust SE for a numeric predictor ----------------------------------

# HC3 standard errors for the slope of a continuous predictor.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = age,
  vcov = "HC3",
  ci = FALSE
)

# Cluster-robust SE for repeated-measures data: the `sleep` dataset
# has 10 subjects measured twice (one observation per group).
if (requireNamespace("clubSandwich", quietly = TRUE)) {
  table_continuous_lm(
    sleep,
    select = extra,
    by = group,
    cluster = ID,
    vcov = "CR2"
  )
}

# --- Covariate adjustment -----------------------------------------------

# Adjust the comparison of `wellbeing_score` and `bmi` by `sex` for `age`
# and `education`. The footer surfaces the adjustment estimand
# ("proportional" by default = G-computation, matching Stata `margins`).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  covariates = c(age, education),
  vcov = "HC3"
)

# Same model with the emmeans / SPSS UNIANOVA convention (equal-weight
# marginal means on a synthetic covariate grid).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  covariates = c(age, education),
  adjustment = "balanced",
  vcov = "HC3"
)

# Effect sizes adjust automatically: f2 / omega2 become partial
```

```
# effect sizes via the partial F restricted to the focal `by`.
# d / g are undefined under adjustment and raise spicy_unsupported.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  covariates = c(age, education),
  effect_size = "f2",
  effect_size_ci = TRUE
)

# --- Article-style polish ------------------------------------------------

# Pretty outcome labels and adjusted R-squared.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  labels = c(
    wellbeing_score = "WHO-5 wellbeing (0-100)",
    bmi = "Body-mass index (kg/m^2)"
  ),
  r2 = "adj_r2"
)

# European decimal comma.
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  decimal_mark = ","
)

# Regex selection of all columns starting with "life_sat".
table_continuous_lm(
  sochealth,
  select = "^life_sat",
  by = sex,
  regex = TRUE
)

# --- Output formats ------------------------------------------------------

# The rendered outputs below all wrap the same call:
#   table_continuous_lm(sochealth,
#                       select = c(wellbeing_score, bmi),
#                       by = sex)
# only `output` changes. Assign to a variable to avoid the
# console-friendly text fallback that some engines fall back to
# when printed directly in `?` help.

# Wide data.frame (one row per outcome).
table_continuous_lm(
```

```
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  output = "data.frame"
)

# Raw long data.frame (one block per outcome).
table_continuous_lm(
  sochealth,
  select = c(wellbeing_score, bmi),
  by = sex,
  output = "long"
)


# Rendered HTML / docx objects -- best viewed inside a
# Quarto / R Markdown document or a pkgdown article.
if (requireNamespace("tinytable", quietly = TRUE)) {
  tt <- table_continuous_lm(
    sochealth, select = c(wellbeing_score, bmi), by = sex,
    output = "tinytable"
  )
}
if (requireNamespace("gt", quietly = TRUE)) {
  tbl <- table_continuous_lm(
    sochealth, select = c(wellbeing_score, bmi), by = sex,
    output = "gt"
  )
}
if (requireNamespace("flextable", quietly = TRUE)) {
  ft <- table_continuous_lm(
    sochealth, select = c(wellbeing_score, bmi), by = sex,
    output = "flextable"
  )
}

# Excel and Word: write to a temporary file.
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  tmp <- tempfile(fileext = ".xlsx")
  table_continuous_lm(
    sochealth, select = c(wellbeing_score, bmi), by = sex,
    output = "excel", excel_path = tmp
  )
  unlink(tmp)
}
if (
  requireNamespace("flextable", quietly = TRUE) &&
    requireNamespace("officer", quietly = TRUE)
) {
  tmp <- tempfile(fileext = ".docx")
  table_continuous_lm(
    sochealth, select = c(wellbeing_score, bmi), by = sex,
    output = "word", word_path = tmp
```

```
  )
  unlink(tmp)
}


## Not run:
# Clipboard: writes to the system clipboard.
table_continuous_lm(
  sochealth, select = c(wellbeing_score, bmi), by = sex,
  output = "clipboard"
)

## End(Not run)
```

`table_continuous_svy` *Descriptive statistics from a survey design*

Description

The design twin of `table_continuous()`: the same table of means, standard deviations, intervals and counts, computed from a `survey::svydesign()` or `survey::as.svrepdesign()` object instead of a data frame.

Not one statistic is computed here. `survey::svymean()` gives the mean, its standard error and its design effect, `survey::svyvar()` the standard deviation, `survey::svyquantile()` the quantiles, and `survey::svyttest()` / `survey::regTermTest()` / `survey::svyranktest()` the group comparison. Every interval and every test is referred to `survey::degf(design)`.

Usage

```
table_continuous_svy(
  design,
  select = tidyselect::everything(),
  by = NULL,
  exclude = NULL,
  regex = FALSE,
  drop_na = TRUE,
  deff = FALSE,
  qrule = "math",
  df = NULL,
  test = c("welch", "student", "nonparametric"),
  p_value = NULL,
  p_adjust = "none",
  statistic = FALSE,
  show_n = TRUE,
  show_columns = NULL,
  ci = TRUE,
  labels = getOption("spicy.var_labels", FALSE),
```

```
  note_labels = NULL,
  ci_level = 0.95,
  digits = 2,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  verbose = FALSE,
  user_na = TRUE,
  style = NULL
)
```

Arguments

design	A survey design: `survey::svydesign()` or `survey::as.svrepdesign()`. Two-phase, pps, database-backed and multiframe designs are refused with a classed error.
select	Columns to summarize, as a tidyselect expression on the design's variables. As in `table_continuous()`, the by column is not part of the selection (naming it, or giving its position, is an error), a name that is not a column is an error, and so is a selection with no numeric column. The design's own columns are not part of it either, since they are not variables to describe: its sampling weights, replicate weights, cluster identifiers (every stage, unless each cluster of a one-stage design is a single observation) and finite population corrections. `everything()`, the other selection helpers and `regex` skip them, and naming one, or giving its position, is an error; `table_continuous()` describes it from the data. Strata stay variables, since a stratification variable such as a region is often one of interest. These columns are read from the design object. A design made by `survey::as.svrepdesign()` records none of them, and one made by `survey::svrepdesign()` names its sampling weights only in its call, as a formula, until `subset()`, `update()`, `postStratify()`, `calibrate()`, `rake()` or `trimWeights()` replaces the call: leave such a column out with a negation (`select = !w`).
by	A single grouping column. One domain per level, in the order of `table_continuous()`: a factor's levels, any other column sorted (text in the C locale, `FALSE` before `TRUE`, numbers ascending). A by that no observation holds is an error under `drop_na = TRUE`.
exclude	Columns to exclude. Supports tidyselect syntax and character vectors of column names.
regex	Logical. If `FALSE` (the default), uses tidyselect helpers. If `TRUE`, the `select` argument is treated as a regular expression.

drop_na — Drop observations with a missing by value. A table that computes removes the missing values by default and counts them in the note, the rule of table_continuous(): TRUE (the default) removes them, and the missing values of each summarized variable are always removed. The note lists both, by last: a missing by takes the observation out of the whole table, a missing value only out of its own variable. With FALSE they form a (Missing) domain of their own – an ordinary subpopulation, with its own degrees of freedom – which is excluded from the group comparison. Ignored (with a warning) when by is not used.

deff — Show the design effect: FALSE (default), TRUE (against sampling without replacement) or "replace" (against sampling with replacement, ignoring the finite population correction).

qrule — Quantile rule: "math" (default), "spicy", or anything survey::svyquantile() accepts, including a function.

df — Degrees of freedom for the intervals. NULL (default) uses survey::degf() on each domain. It does not reach the group comparison: survey::svyttest() and survey::svyranktest() have no df argument, so the test keeps the design's own degrees of freedom and the note says so.

test — Group comparison: "welch" (default), "student" (warns; identical under a design) or "nonparametric". A variable whose complete cases include negatively weighted rows is not compared: a design-based test is not defined when the weights change sign. Its estimates are still reported, the note says which comparisons were withheld, and the call warns (spicy_negative_weights_no_test).

p_value — Logical or NULL. If TRUE and by is used, adds a *p*-value column from the test specified by test. When NULL (the default), the *p*-value is shown automatically whenever by is supplied, and hidden otherwise. Pass p_value = FALSE to suppress the column explicitly. Ignored when by is not used.

p_adjust — Multiple-comparison adjustment of the p-value column, applied to the family of every test in the table (one design-based p-value per variable; a withheld test stays out). One of "none" (default), "holm", "hochberg", "hommel", "bonferroni", "BH" / "fdr", or "BY", as in table_regression(); delegated to stats::p.adjust(). Needs by and the p-value column. See *Multiple-comparison adjustment*.

statistic — Logical. If TRUE and by is used, the test statistic is shown in an additional column (e.g., t(df) = ..., F(df1, df2) = ..., W = ..., or H(df) = ...). Both p_value and statistic are independent; either or both can be enabled. Defaults to FALSE. Ignored when by is not used.

show_n — Logical. If TRUE, includes an unweighted n column in the printed ASCII table and in every rendered output (tinytable, gt, flextable, word, excel, clipboard). Set to FALSE to drop the n column structurally from those outputs (no empty placeholder, no spanner). The n column is always present in output = "long" for downstream programmatic access. Defaults to TRUE. Ignored (with a warning) when show_columns is supplied.

show_columns — Character vector of statistic tokens; NULL keeps the default display.

ci — Logical. If TRUE, includes the mean confidence interval columns (<level>% CI LL / <level>% CI UL) and their spanner in the printed ASCII table and in every rendered output (tinytable, gt, flextable, word, excel, clipboard). Set to

FALSE to drop both columns and the CI spanner structurally from those outputs (no empty placeholders, no border lines under an empty header). The CI bounds are always present as ci_lower / ci_upper in output = "long" for downstream programmatic access. Defaults to TRUE. The CI level is taken from ci_level. Ignored (with a warning) when show_columns is supplied.

labels — How each variable is named in the table: in the rows, in the title, and in the notes. Four forms:

- labels = FALSE shows the column name.
- labels = TRUE shows the column's label attribute (as set by haven, labelled, or label_from_names()) when it is present, not NA, and not empty, and the column name otherwise.
- labels = c(bmi = "Body mass index"), a named character vector, gives the columns it names these labels. Every other column follows options(spicy.var_labels): its name when the option is unset or FALSE, its label attribute when it is TRUE.
- labels = var_labels(bmi = "BMI (kg/m2)") gives the columns it names these labels, and every other column its label attribute, whatever the option says. var_labels() alone is labels = TRUE.

A name that matches no column of the table warns (class spicy_unknown_name) and is left out, so one vector can still serve every table of a project. The default reads options(spicy.var_labels), and a TRUE or FALSE typed in the call wins over it. Any other value, NULL included, is an error. A variable label documents the data file; a published table carries the wording you set here. note_labels gives a variable a shorter label in the title and the notes.

note_labels — The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one labels gives it: a named character vector, note_labels = c(sex = "sex at birth"), or a var_labels() object, which reads as one. labels can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of labels, and so do the title and the notes for a variable note_labels does not name. A name that matches no column warns and is left out, as in labels, and NULL (the default) renames nothing. A note label that another variable of the table shows is an error (class spicy_label_collision): choose another one, or relabel the other variable.

ci_level — Confidence level for the mean confidence interval (default: 0.95): a number strictly between 0 and 1.

digits — Number of decimal places of the statistics on each variable's own scale: the mean, SD, standard error, median, IQR, quartiles, minimum and maximum, and the confidence interval of the mean. One non-negative whole number sets them for every variable (default: 2), and the weighted *n* and the design effect with them. A vector named by column, not by label, gives each variable it names its own, digits = c(height = 0, bmi = 1); the other variables, the weighted *n* and the design effect keep the default, the digits of a spicy_style() in force included. A name that matches no variable warns and is left out, as in labels. A fraction is an error, as for every *_digits argument. The *p*-values (p_digits), the test statistics (two decimals) and the counts keep their own decimals.

p_digits — Integer >= 1. Number of decimal places used to render *p*-values in the p column (default: 3, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: p_digits = 3 prints .045 and <.001; p_digits = 4 prints .0451 and <.0001; p_digits = 2 prints .05 and <.01. Useful for genomics / GWAS contexts with very small *p*-values, or for journals using a coarser convention. Leading zeros are always stripped, following APA convention.

decimal_mark — The decimal mark, a single character: "." (the default) or "," in most styles; the Lancet style of spicy_style() uses the middle dot.

align — Horizontal alignment of numeric columns in the printed ASCII table and in the tinytable, gt, flextable, word, excel, and clipboard outputs. The first column (Variable) and Group (when present) are always left-aligned. One of:

- "decimal" (default): align numeric columns on the decimal mark, the standard scientific-publication convention used by SPSS, SAS, and LaTeX siunitx. Numeric cells are pre-padded with figure-spaces (U+2007, digit-width) so every string in a column has the same width with the decimal mark at the same internal position; centering those uniform-width strings then stacks the decimal points vertically. The same pad-then-center strategy is applied on every rendering engine (gt, tinytable, flextable, word, ASCII print) for a homogeneous rendering, matching table_regression() and table_continuous_lm(). The clipboard output is delimited text meant to be parsed rather than read at a fixed width, so its cells travel unpadded (a padded number pastes as text next to an unpadded number).
- "center": center-align all numeric columns.
- "right": right-align all numeric columns.

"center" and "right" reach the excel output too. "decimal" does not: Excel cells are written unpadded, because cell-string padding does not align decimals under a proportional font, so the workbook keeps the engine's own convention instead – counts and the *p*-value right-aligned, the other numeric columns centered. Same default and same three values as table_continuous_lm(), whose excel output still uses that convention at every align.

note — The note under the table. NULL (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the jama and lancet themes; see note_label in spicy_style()), then, in a table_regression() with stars, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks ("\n") included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (note = "Note. Data weighted."), and the legend of the stars stays below it. "" keeps that legend alone, and no note at all without stars. FALSE removes the note from every output, the legend included.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an

	empty string is an error, and so is `note_add` with `note = FALSE`, which leaves no note to add to.
`output`	One of `"default"`, `"data.frame"`, `"long"`, or a rendering engine: `"tinytable"`, `"gt"`, `"flextable"`, `"excel"`, `"clipboard"`, `"word"`. `"data.frame"` is the displayed table with its numbers, `"long"` one row per `(variable x group)` with every statistic, as in `table_continuous()`; see *Value*.
`excel_path`	File path for `output = "excel"`, ending in `".xlsx"`, in a directory that exists.
`excel_sheet`	Sheet name for `output = "excel"`. `NULL` (the default) uses `"Descriptives"`.
`clipboard_delim`	Delimiter for `output = "clipboard"` (default: `"\t"`). A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.
`word_path`	File path for `output = "word"`, ending in `".docx"`, in a directory that exists.
`verbose`	Logical. If `TRUE`, a message (class `spicy_info`) names the non-numeric columns left out (default: `FALSE`).
`user_na`	Logical. If `TRUE` (the default), declared missing values never reach the numeric summaries: they are excluded like `NA` and disclosed in the table note (`Declared missing values removed: ...`); declared-missing by values form no group. If `FALSE`, the declared codes are summarized as ordinary numbers. See the "Declared missing values" section of `freq()`.
`style`	A journal style: a theme name (`"jama"`, `"nejm"`, `"lancet"`, `"annals"`, `"apa"`, `"aer"`), a `spicy_style()` object, or `NULL` (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set `options(spicy.style = )` for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; `?spicy_style` lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

Depends on `output`:

- `"default"`: a `spicy_continuous_svy_table` (a `data.frame` subclass, with the parent class `"spicy_table"`), printed when the call is bare; its columns are the table's compute frame, and the display frame and the typed view are attached.
- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per printed row, the key columns `variable`, `label` and, with by, `group` and `.row_role` (`"group"`, or `"missing"` for the group of missing by values), then the printed columns under their names (`"M"`, `"SD"`, `"SE"`, `"95% CI LL"`, ..., `"n"`, `"p"`), as in `table_continuous()`.
- `"long"`: one row per `(variable x group)` (per variable without by) with every statistic the table computes, whatever it displays: `variable`, `label`, `group` and `.group_role` (with by, as in `table_continuous()`), `mean`, `sd`, `se`, `min`, `max`, `ci_lower`, `ci_upper`, `median`, `q1`, `q3`, `iqr`, `med_ci_lower`, `med_ci_upper`, `n` (observed), `weighted_n` (the estimated population), `deff`, `degf` (the degrees of freedom of the domain), and, with by, the test on the first row of each variable: `test_type` (`"design_t"` or `"design_f"`), `statistic`, `df`, `df.residual`, `p.value` (with the *p*-value) and, under `p_adjust`, `p_unadjusted`.

- `"tinytable"`, `"gt"`, `"flextable"`: the rendered table.
- `"excel"` / `"word"`: writes the file and returns its path invisibly.
- `"clipboard"`: copies the table and returns the display `data.frame` invisibly.

The default object and the two data frames carry the whole note as printed, its lead-in included, in `attr(x, "note")`. The default object also keeps the title it prints in `attr(x, "title")`.

Which function do I need?

A data frame with a column of weights is `table_continuous()` (`weights = `). A survey design object – strata, clusters, finite population correction, calibration, replicate weights – is this function. Passing one to the other errors with the name of the right one; there is no silent coercion, because the design-based standard errors, degrees of freedom and tests cannot be recovered from the weights alone.

Two conventions, one bridge

`table_continuous(weights = )` implements the **frequency-expansion** convention: a weight is a number of copies, and `SD` has denominator `sum(w) - 1`. This function implements the **sampling-weight** convention: a weight is a number of units represented, and `SD` is `sqrt(survey::svyvar())`, whose denominator is `n - 1` on weights normalized to sum to `n`. These are two estimands, not two approximations of one.

`rescale = TRUE` is the bridge, and it is an identity rather than a coincidence. Writing `w' = w * n / sum(w)`, the rescaled weighted variance is `sum(w' (x - xbar)^2) / (sum(w') - 1) = n / (n - 1) * sum(w (x - xbar)^2) / sum(w`, which is what `survey::svyvar()` computes. So on a design that declares nothing but weights, `table_continuous(weights = w, rescale = TRUE)` and this function return the same mean and the same standard deviation. The default `rescale = FALSE` does not, and that is the estimand boundary, not a bug.

The mean is continuous across both regimes: `sum(w x) / sum(w)` does not move when the weights are rescaled.

Choosing the statistics

`show_columns` takes the tokens of `table_continuous()` with two additions and one removal:

- `"se"` – the design-based standard error of the mean;
- `"deff"` – the design effect (requires `deff = TRUE`);
- `"med_ci"` is refused. The exact interval of the sibling inverts a binomial sign test on independent observations, which a clustered or stratified sample is not.

Quantiles

`qrule = "math"` is the default and estimates `inf{x : F(x) >= p}`, the quantile of the *population*. `qrule = "spicy"` switches to the type-7 interpolation `table_continuous()` uses, for a reader who needs the two tables to agree cell for cell; any other value – including a function – is handed to `survey::svyquantile()` untouched. The note always says which rule produced the numbers.

Groups and degrees of freedom

`by =` cuts one domain per group with `[` on the design. survey recomputes the degrees of freedom on the primary sampling units and strata each domain retains, so a grouped table generally carries a *different* df per row; the note gives the span when they differ.

A group with a missing value is a domain like any other: `drop_na = FALSE` gives it a `(Missing)` row, with its own degrees of freedom. A domain reduced to one primary sampling unit has none, and its interval shows the undefined dash rather than an interval built on `qt(p, df = 0)`.

The comparison is a single test on the whole design, not a set of pairwise ones: `survey::svyttest()` with two observed groups, `survey::regTermTest()` on `survey::svyglm()` with three or more, or `survey::svyranktest()` under `test = "nonparametric"`. Under a design the Welch / Student distinction does not exist – the variance is the design's – so `test = "student"` warns and behaves like `"welch"`. A variable with no variance, or whose domains are each constant on their own value (a residual variance of zero), has no comparison, with a warning (class `spicy_undefined_stat`), where survey would still return a p-value. Under `test = "nonparametric"` too, unlike `table_continuous()`: `survey::svyranktest()` is a design-based test of the ranks, which are then constant within the domains as well. Its test cells show the undefined mark, as do those of any comparison that cannot run (too few observations in a domain, a test refused under weights that change sign). The note names the comparison of each variable.

A variable with no observed value in a domain (a question not asked in that group) has no data in it, as in `table_continuous()`: its cells there show an en dash, *n* included, those rows are not counted in the missing values of the note, the comparison uses the domains with data, and the note says so ("No data for X in G.").

Stability

This function is **stabilizing** in the sense `?spicy` defines: the names of its design-specific arguments may still be tightened before 1.0 – with a `NEWS.md` entry – but the behavior does not change silently. It was experimental through 0.13; the shape of the table has held across the cycle, so it now moves on the parent family's clock rather than its own. The numbers themselves are survey's and do not move with it.

What is absent, and why

`weights` and `rescale` (the weighting *is* the design), `effect_size` and `smd` (no established design-based variance), and `data`.

Multiple-comparison adjustment

The p-values of a descriptive table describe: they test no prespecified hypothesis. STROBE (explanation of item 14) and the SAMPL guidelines advise against significance tests in descriptive and baseline tables, hence the default `p_adjust = "none"`.

When a journal or a protocol asks for adjusted p-values anyway, `p_adjust` applies `stats::p.adjust()` to the family of every test in the table: one p-value per variable (one per block in `table_outcome()`), never the rows of one variable, which share one test. `"holm"` suits a confirmatory set of comparisons, `"BH"` a screening one; `table_regression()` lists the methods. A test that returned no p-value stays out of the family.

The adjustment runs before any formatting, so the `p` column and its `<.001` threshold show the adjusted values, and the table note gives the number of tests and the method. The raw p-values stay

in the object: a p_unadjusted column next to the p-value column of the returned data frames, and col_meta$p$p_unadjusted (with the method in p_adjust and the family size in p_adjust_m) in as_structured(). Without by, or with p_value = FALSE, the table shows no p-value to adjust, and a p_adjust other than "none" is an error.

See Also

table_continuous() for the data-frame sibling, table_categorical_svy() for categorical variables, table_regression() on a survey::svyglm() fit for a model.

Examples

```
data(api, package = "survey")
dclus1 <- survey::svydesign(
  id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc
)
table_continuous_svy(dclus1, select = c(api00, api99))
table_continuous_svy(dclus1, select = api00, by = stype)
table_continuous_svy(
  dclus1,
  select = api00,
  show_columns = c("m", "se", "ci", "deff", "n"),
  deff = TRUE
)
```

---

table_multiresponse *Multiple-response summary table*

Description

Tabulates a "check all that apply" question: one row per item, with the number of respondents who chose it and their percentage, a row for the respondents who chose none of the items when there are any (see none_row), then a row with the number of respondents the percentages are computed on. The items are the indicator columns of the question, selected with tidyselect syntax.

With by, the counts and percentages are given per group and in a Total margin, each item gets a test across the groups (the chi-squared, or Fisher's exact test when an expected count is below 5), and the question gets a global test of all its items together, valid although one respondent can choose several items.

Multiple output formats are available via output: a printed ASCII table ("default"), a wide or long numeric data.frame ("data.frame", "long"), or publication-ready tables ("tinytable", "gt", "flextable", "excel", "clipboard", "word").

Usage

```
table_multiresponse(
  data,
  select,
```

```
  by = NULL,
  counted = NULL,
  percent = c("cases", "responses", "both"),
  base = c("valid", "answered", "all"),
  none_row = TRUE,
  sort = c("", "+", "-", "name+", "name-"),
  test = c("auto", "chisq", "fisher", "midp", "n1"),
  p_value = NULL,
  global_test = c("rs2", "bootstrap"),
  boot_n = 1999L,
  p_adjust = "holm",
  statistic = FALSE,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  percent_digits = 1,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  title = NULL,
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  style = NULL
)
```

Arguments

data	A data frame, one row per respondent.
select	The indicator columns of the question, one per item, in tidyselect syntax or as a character vector of column names. Their order is the order of the rows, unless sort sorts them. The by column is not part of the selection: a selection helper skips it, and naming it, or giving its position, is an error. A name that is not a column of data is an error (class spicy_missing_column), and so is an empty selection.
by	Optional grouping column, unquoted or as a string. Factor levels keep their declared order, a declared level with no respondent included; any other column is sorted, as in table_categorical(): text in the C locale, FALSE before TRUE, numbers ascending, whatever the order of the rows, and a labelled column is named and ordered by its value labels as there. The missing answers to the question are counted (see base and none_row), as a table that counts does (see drop_na in table_categorical()), but a missing by has no group to be counted in: rows with a missing by are removed from the whole table, and the note lists by with their count ("Missing values removed: sex (2)."),

so this table has no drop_na; a by that no observation holds is an error (class spicy_invalid_data). A level displayed with the label of the Total margin is an error, as in table_categorical().

counted — The value that marks a choice. NULL (the default) reads logical columns (TRUE is chosen) and numeric columns coded 0 and 1 (1 is chosen) as they are; any other column needs counted, and the error names it. When supplied, counted applies to every selected column: a factor or character column is chosen where it equals counted (counted = "Yes"), a labelled column where its code or its value label does (counted = 1 or counted = "Yes"). Any other non-missing value is "not chosen", and a missing value stays missing. A counted that matches no value of any selected column is an error.

percent — The percentage shown beside each count: "cases" (the default), the percentage of respondents, which can sum to more than 100; "responses", the percentage of all the choices made in the group, which sums to 100; or "both", two columns.

base — The respondents the percentages of cases are computed on; the note names the base the table uses.

- "valid" (the default): the respondents with at least one non-missing indicator, so that a respondent who chose nothing counts and one who skipped the question does not, as expss counts its valid cases.
- "answered": the respondents who chose at least one item, the default of SPSS MULT RESPONSE (MISSING = TABLE) and of Stata's mrtab, whose include option gives "valid" (Jann, 2005, pp. 98 and 107-108), where choosing nothing cannot be told from skipping.
- "all": every row of the group, as questionr::multi.table() counts.

In a "check all that apply" question, an item left unchecked cannot tell "none applies" from "skipped": "valid" reads a respondent with no item chosen as "none applies", "answered" (the convention of SPSS and Stata) as nonresponse. "answered" suits a questionnaire with neither a "None of these" option nor a required answer. An export that leaves every unchecked box missing gives both bases the same respondents, and only "answered" describes them in the note. A filter code that is not declared missing (for example -1 "not applicable" kept as an ordinary value) counts as "not chosen" and keeps the filtered respondents in the base: recode it to NA first (SPSS user-missing values declared as such are already read as missing).

none_row — Logical. If TRUE (the default), the respondents of the base who chose no item get a row of their own, after the items and before the Base row: "(No item chosen)" under base = "valid" and "all", and, under "all", "(No data for the question)" for the respondents with every indicator missing. Each row appears only when its count is above 0. base = "answered" leaves both groups out of the base and shows neither row. A row gives the count and the percentage of cases of each group and of the margin, on the base of the column, as the items do. Its percentage of responses and its test cells show a dash: these respondents gave no answer, and the rows take no part in the tests or in sort. The note then no longer counts the respondents a row shows. FALSE gives these counts in the note only.

`sort` Sorting method for items, as in `freq()`:

- `""` - no sorting (default)
- `"+"` - increasing frequency
- `"-"` - decreasing frequency
- `"name+"` - alphabetical A-Z
- `"name-"` - alphabetical Z-A

With no sorting, the rows keep the order in which `select` delivers the columns: their order in `data` with a helper such as `starts_with()`, or the order written in `c()`. The function cannot know the questionnaire, so its order comes from the order of the columns or the order written in `select`. The frequency is the total count of each item, and items with the same count keep the order of `select`. The alphabetical sort reads the labels displayed in the rows (the column names, or the labels set by `labels`) in the C locale, the same on every platform: capitals before lower case, accented letters after `z`.

`test` The test of each item's p-value, as in `table_categorical()`: `"auto"` (the default) takes Pearson's chi-squared test when every expected count of the item's table is at least 5 and Fisher's exact test otherwise; `"chisq"` and `"fisher"` take one test for every item; `"midp"` takes the Fisher mid-p test for an item of a two-group table; `"n1"` takes the N-1 chi-squared test for an item of a two-group table whose expected counts are all at least 1, and Fisher's exact test for one with a smaller count. With more groups, `"midp"` and `"n1"` follow `"auto"`. The note names the test of each item. See *Tests*.

`p_value` Logical or `NULL`. With `by`, `NULL` (the default) and `TRUE` show the item `p` column and the global test in the note. `FALSE` removes both from every output, the `p` column of the data frame and the `p.value` column of the long output included. `p_value` and `statistic` are independent, as in `table_continuous()`. Ignored without `by`, with a warning when `TRUE`.

`global_test` The global test: `"rs2"` (the default), the sum of the item chi-squared statistics with the second-order Rao-Scott correction; or `"bootstrap"`, the same sum referred to its bootstrap distribution. See *Tests*.

`boot_n` Number of bootstrap resamples for `global_test = "bootstrap"`: a positive whole number, as in `table_regression()` and `table_continuous_lm()`. Defaults to `1999`, the default `B` of `MI.test()` in the `MRCV` package for the same bootstrap test (reference manual of `MRCV` 0.4-0). Typed with `global_test = "rs2"`, or in a table without a global test, it is ignored with a warning (class `spicy_ignored_arg`).

`p_adjust` Multiple-comparison adjustment of the item `p` column, applied to the family of the items tested: `"holm"` (the default), `"none"`, or another method of `stats::p.adjust()` (`"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"` / `"fdr"`, `"BY"`), as in `table_categorical()`. See *Multiple-comparison adjustment*. Typing it when the table has no test to adjust (no `by`, or `p_value = FALSE`) is an error.

`statistic` Logical. If `TRUE`, adds a `Test` column with each item's chi-squared statistic and its degrees of freedom, with or without the p-values of `p_value`; an item whose p-value comes from Fisher's exact test or its mid-p version has no statistic and shows a dash, and an item tested by the N-1 chi-squared shows that statistic. Defaults to `FALSE`.

`labels` How each variable is named in the table: in the rows, in the title, and in the notes. Four forms:

- `labels = FALSE` shows the column name.
- `labels = TRUE` shows the column's `label` attribute (as set by `haven`, `labelled`, or `label_from_names()`) when it is present, not `NA`, and not empty, and the column name otherwise.
- `labels = c(bmi = "Body mass index")`, a named character vector, gives the columns it names these labels. Every other column follows `options(spicy.var_labels)`: its name when the option is unset or `FALSE`, its `label` attribute when it is `TRUE`.
- `labels = var_labels(bmi = "BMI (kg/m2)")` gives the columns it names these labels, and every other column its `label` attribute, whatever the option says. `var_labels()` alone is `labels = TRUE`.

A name that matches no column of the table warns (class `spicy_unknown_name`) and is left out, so one vector can still serve every table of a project. The default reads `options(spicy.var_labels)`, and a `TRUE` or `FALSE` typed in the call wins over it. Any other value, `NULL` included, is an error. A variable label documents the data file; a published table carries the wording you set here. `note_labels` gives a variable a shorter label in the title and the notes.

`note_labels` The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one `labels` gives it: a named character vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and `NULL` (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`percent_digits` Number of digits for percentages in report outputs. Defaults to `1`. A whole number: a fraction is an error, as for every `*_digits` argument.

`p_digits` Integer >= 1. Number of decimal places used to render *p*-values in the `p` column (default: `3`, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: `p_digits = 3` prints `.045` and `<.001`; `p_digits = 4` prints `.0451` and `<.0001`. Leading zeros are always stripped, following APA convention.

`decimal_mark` The decimal mark, a single character: `"."` (the default) or `","` in most styles; the Lancet style of `spicy_style()` uses the middle dot.

`align` Horizontal alignment of the numeric columns in the console and in the rendered outputs: `"decimal"` (the default), `"center"`, or `"right"`, as in `table_categorical()`.

`title` The table title. `NULL` (the default) gives "Multiple responses", followed by "by" and the by variable under the `labels` contract (or its label in `note_labels`); a single string replaces it, and `FALSE` removes it, as in `table_regression()`.

`note` The table note. `NULL` (the default) builds it, opening with "Note.": what the percentages sum to, the base with the respondents it holds or leaves out, the

rows removed for a missing by and the items with no data in a group, then the tests, and the sentences of note_add. A single string replaces it, printed as typed after the lead-in ("Note.", unless the string opens with one), note_add after it, and FALSE removes it, as in every table function.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an empty string is an error, and so is note_add with note = FALSE, which leaves no note to add to.

output — Output format. One of:

- "default" (an ASCII table object, printed when the call is bare)
- "data.frame" (the displayed table, numeric)
- "long" (a long numeric data.frame)
- "tinytable" (requires tinytable)
- "gt" (requires gt)
- "flextable" (requires flextable)
- "excel" (requires openxlsx2)
- "clipboard" (requires clipr)
- "word" (requires flextable and officer)

excel_path — Path for output = "excel", ending in ".xlsx", in a directory that exists. Defaults to NULL.

excel_sheet — Sheet name for Excel export. NULL (the default) uses "Multiple responses".

clipboard_delim — Delimiter for clipboard text export. Defaults to "\t". A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path — File path for output = "word". Defaults to NULL.

style — A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set options(spicy.style =) for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; ?spicy_style lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

Depends on output:

- "default": a spicy_multiresponse_table (a data.frame subclass), with the rendering metadata as attributes, returned visibly, so a bare call prints the ASCII table.

- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per item, the rows of `none_row`, and a last row with the base. Columns: `variable` (the item's column, `"none"`, `"no_data"`, or `"nobs"` on the base row), `label` (the label the row displays), `.row_role` (what the row is: `"summary"` for an item, `"none"` for "(No item chosen)", `"missing"` for "(No data for the question)", `"fit_stat"` for the base, the roles of `as_structured()`), then the printed columns: the count and the percentage of each group of `by` and of the `Total` margin (`<group> n` and `<group> %`, or `<group> % of cases` and `<group> % of responses` under `percent = "both"`; `n` and `%` without `by`), `Test` (the chi-squared of the item, with `df` beside it) under `statistic = TRUE`, and `p` with `p_value`.
- `"long"`: one row per item and group, then one per row of `none_row` and group, with columns `variable` (the item's column, `"none"` or `"no_data"` on the rows of `none_row`), `label` (the label the row displays), `.row_role` as above, with by `group`, `.group_role` (the role of `group`: `"group"`, or `"total"` for the margin) and `total` (`TRUE` on the rows of the margin), `n`, `base` (the respondents of the group; in the margin, those of the groups with data for the item), the percentage named by its base: `pct_cases` (of the respondents) or `pct_responses` (of the choices made), as `percent` asks, both under `percent = "both"`, and, when the item tests ran (for `p_value` or for `statistic`), `test_type` (the test of the item, as in `table_categorical()`: `"chi_squared"`, `"n1_chi_squared"`, `"fisher_exact"`, `"fisher_midp"`, `"fisher_exact_mc"`), `statistic` and `df` (`NA` for an item tested by an exact test), `p.value` with `p_value`, and `p_unadjusted` under an adjustment. The rows of `none_row` are `NA` in the test columns.
- `"tinytable"`, `"gt"`, `"flextable"`: the rendered table object.
- `"excel"` / `"word"`: writes to disk and returns the file path invisibly.
- `"clipboard"`: copies the table and returns the display `data.frame` invisibly.

The default object and the data frames carry the note as printed in `attr(x, "note")` (its lead-in included, `NULL` under `note = FALSE`) and, when the table has a global test, `attr(x, "global_test")`: a one-row data frame with `test`, `statistic`, `df`, `p.value`, `boot_n`, `n_items` (the items tested), `n_groups`, `rs_mean` and `rs_cv` (the design-effect summary of `global_test = "rs2"`), and `boot_redraws` (the resamples drawn again under `global_test = "bootstrap"`); see *Tests*. `as_structured()` reads the default object, with the same `.row_role` and a `.variable` of `"none"` and `"no_data"` on the rows of `none_row`; tidy() reads the items only.

Tests

With `by`, the table tests multiple marginal independence: that the probability of choosing each item is the same in every group. Each item gets the Pearson chi-squared statistic of its $r \times 2$ table (group by chose the item) with $r-1$ degrees of freedom: the arithmetic of `chisq.test(correct = FALSE)`, with no continuity correction and no constant added to an empty cell (Agresti and Liu, 1999).

The item p-value comes from the test `test` chooses, as in `table_categorical()`. Under `"auto"`, the default, it is that chi-squared test when every expected count of the item's table is at least 5, and Fisher's exact test (`stats::fisher.test()`) otherwise; when the exact enumeration fails, the p-value is its Monte Carlo estimate over 2000 tables, which draws from the session's RNG. An item tested by Fisher's exact test has no statistic: its `Test` cell shows a dash, and `statistic` and `df` are `NA` in the long output. Without p-values (`p_value = FALSE`), `statistic = TRUE` shows the chi-squared statistic of every item. The global test sums the Pearson statistics of the items whatever `test` chooses for their p-values, since its correction and its bootstrap are defined for that sum.

With two groups, an item's table is 2x2. `test = "midp"` then gives it the Fisher mid-p test of Fagerland, Lydersen and Laake (2017, section 4.4.6), and `test = "n1"` the N-1 chi-squared test of

Campbell (2007) when its expected counts are all at least 1 and Fisher's exact test otherwise, both defined under *Tests* in `table_categorical()`. With more groups, the items follow `"auto"`. An item tested by the mid-p test has no statistic (a dash); one tested by the N-1 chi-squared shows that statistic, on 1 degree of freedom.

The global test sums the item statistics, $X^2_S = \sum_j X^2_j$, a value that does not depend on which answer of an item is coded as chosen (Agresti and Liu, 1999). One respondent can choose several items, so the items are correlated and X^2_S is not chi-squared under the null (Loughin and Scherer, 1998): its asymptotic distribution is a weighted sum of chi-squared variables, the weights being the eigenvalues of a design-effect matrix (Bilder, Loughin and Nettleton, 2000, Appendix A). `global_test = "rs2"` matches its first two moments, the second-order Rao-Scott correction of Thomas and Decady (2004, pp. 48 and 58-59):

$$X^2_{RS2} = \frac{c(r-1)\,X^2_S}{\sum_k \lambda_k^2}, \qquad \nu = \frac{c^2(r-1)^2}{\sum_k \lambda_k^2}$$

referred to a chi-squared distribution with ν degrees of freedom, usually fractional. Here c is the number of items tested and the λ_k are the eigenvalues of the design-effect matrix, in the second-order form of Bilder and Loughin (2004). The within-group covariance of the indicators is centered at the group means, the unrestricted estimate of Bilder and Loughin (2004) that the `MRCV` package also uses, whereas Thomas and Decady (2004) center it at the pooled proportions with divisor $n(n-1)$, which gives p = .068 instead of .063 on the Kansas farmer data of Loughin and Scherer (1998). The result is deterministic. `attr(x, "global_test")` reports `rs_mean`, the mean of the λ_k, which is 1 in the population under the null and which the correction takes as 1 (Thomas and Decady, 2004, p. 48), and `rs_cv`, the $\hat{a}$ of Thomas and Decady (2004, p. 59), defined by $1 + \hat{a}^2 = \sum_k \lambda_k^2/(c(r-1))$: the coefficient of variation of the λ_k when their mean is 1, and `NA` when that ratio falls below 1.

`global_test = "bootstrap"` refers X^2_S to its distribution over `boot_n` resamples of whole response vectors, drawn with replacement from the pooled respondents with the group sizes held fixed (Loughin and Scherer, 1998). The p-value is the share of resamples whose statistic is at least the observed one. A resample in which an item is chosen by every respondent or by none gives that item no statistic: it is discarded and drawn again, the practice of Bilder, Loughin and Nettleton (2000, p. 1298) for resamples whose marginal table has an empty row or column, and `attr(x, "global_test")$boot_redraws` counts the resamples drawn again. When 100 times `boot_n` draws do not yield `boot_n` usable resamples, the function stops with an error. The bootstrap held its level in most published settings, with exceptions reported by Bilder, Loughin and Nettleton (2000, p. 1305), and no study of multiple marginal independence compares it with the second-order test. It is random: call `set.seed()` first to reproduce it.

An item chosen by every respondent, or by none, has no test: its p cell shows a dash, it leaves the global test, and the note names it. Its row stays. An item with no data in a group (see *Base and missing values*) is tested on the groups with data for it, has no test with fewer than two (its p cell is then blank), and leaves the global test, which compares every group on the same items. With fewer than two groups holding respondents there is nothing to compare, and the table shows no test. The note names the global test with its statistic, degrees of freedom, and p-value.

Multiple-comparison adjustment

The items of one question form a natural family of tests, read together, so the item p column is adjusted by Holm's method by default. In `table_categorical()`, each test belongs to a different

variable and the default is "none". Holm's method controls the family-wise error rate whatever the dependence between the items, and its smallest adjusted p-value equals the Bonferroni global p-value of Agresti and Liu (1999). The raw p-values stay in the data frames as p_unadjusted, and in col_meta$p$p_unadjusted of as_structured(). The note names the method and the number of items adjusted together, in the sentence of table_categorical(). The global test is never adjusted.

Base and missing values

Within the base, a missing indicator counts as not chosen, for the percentages and for the tests alike, so the table and its tests describe the same respondents: those with at least one non-missing indicator under base = "valid", those who chose at least one item under base = "answered", every row under base = "all". The counts are the same under the three bases; the percentages of cases and the tests are not. The Base row gives the base of each group, and the note says which base the table uses, with two counts, all groups together: the respondents with no data for the question (every indicator missing) and those with no item chosen (data, but no item chosen). The base holds the second and leaves out the first under "valid", leaves out both under "answered", and holds both under "all"; a count of zero is not mentioned. Under none_row = TRUE, the default, the respondents of these two kinds that the base holds are rows of the table, "(No item chosen)" and "(No data for the question)", with their count and percentage in each group, and the note counts only those the base leaves out. An item displayed with the label of a row the table adds is an error (class spicy_label_collision): choose another label in labels, or give the row another label, for example options(spicy.labels = list(row_multiresponse_base = "N")). Declared missing values of a labelled column (na_values, na_range) are missing, as under the user_na = TRUE default of table_categorical().

An item missing for every respondent of a group (typically an item not asked there) has no data in that group: its cells there show a dash, its Total counts the other groups only, and the note says so, in one sentence for the items with no data in the same groups. Without by, an item with no observed value at all shows a dash. A group with no data for any item was not asked the question: all its cells, Base included, show a dash, and its respondents leave the base, the Total, the tests and the count of the respondents with no data for the question.

A percentage carries no more digits than its denominator, so a base of fewer than 100 respondents takes whole percentages (American Medical Association, 2020, section 4.1.8): percent_digits = 0 prints them.

References

Agresti, A., & Liu, I.-M. (1999). Modeling a categorical variable allowing arbitrarily many category choices. *Biometrics*, 55(3), 936-943. doi:10.1111/j.0006341x.1999.00936.x

American Medical Association. (2020). *AMA Manual of Style: A Guide for Authors and Editors* (11th ed.). Oxford University Press. doi:10.1093/jama/9780190246556.001.0001

Bilder, C. R., & Loughin, T. M. (2004). Testing for marginal independence between two categorical variables with multiple responses. *Biometrics*, 60(1), 241-248. doi:10.1111/j.0006341x.2004.00147.x

Bilder, C. R., Loughin, T. M., & Nettleton, D. (2000). Multiple marginal independence testing for pick any/c variables. *Communications in Statistics - Simulation and Computation*, 29(4), 1285-1316. doi:10.1080/03610910008813665

Campbell, I. (2007). Chi-squared and Fisher-Irwin tests of two-by-two tables with small sample recommendations. *Statistics in Medicine*, 26(19), 3661-3675. doi:10.1002/sim.2832

Fagerland, M. W., Lydersen, S., & Laake, P. (2017). *Statistical Analysis of Contingency Tables*. Chapman and Hall/CRC. doi:10.1201/9781315374116

Jann, B. (2005). Tabulation of multiple responses. *The Stata Journal*, 5(1), 92-122. doi:10.1177/1536867x0500500113

Loughin, T. M., & Scherer, P. N. (1998). Testing for association in contingency tables with multiple column responses. *Biometrics*, 54(2), 630-637. doi:10.2307/3109769

Thomas, D. R., & Decady, Y. J. (2004). Testing for association using multiple response survey data: Approximate procedures based on the Rao-Scott approach. *International Journal of Testing*, 4(1), 43-59. doi:10.1207/s15327574ijt0401_3

See Also

`table_categorical()` for single-response variables; the article Multiple-response questions.

Examples

```
# The five info_* items of sochealth answer one "check all that apply"
# question: the sources of health information used in the past 12 months.
table_multiresponse(sochealth, select = starts_with("info_"))

# The labels of the items, sorted by decreasing count.
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  labels = TRUE,
  sort = "-"
)

# By age group: a test per item and a global test of the question.
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  by = age_group
)

# A descriptive table by sex, on the respondents who chose at least one
# source: percentages of cases and of responses, no tests.
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  by = sex,
  percent = "both",
  base = "answered",
  p_value = FALSE
)

# Items coded "Yes" and "No" need `counted`, the value that marks a choice.
yes_no <- as.data.frame(lapply(
```

```
  sochealth[startsWith(names(sochealth), "info_")],
  function(x) factor(x, levels = c(FALSE, TRUE), labels = c("No", "Yes"))
))
table_multiresponse(yes_no, select = everything(), counted = "Yes")

# Bootstrap global test: set a seed to reproduce its p-value.
set.seed(1)
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  by = sex,
  global_test = "bootstrap",
  boot_n = 999
)

# Wide data frame.
table_multiresponse(
  sochealth,
  select = starts_with("info_"),
  by = sex,
  output = "data.frame"
)


if (requireNamespace("tinytable", quietly = TRUE)) {
  tt <- table_multiresponse(
    sochealth,
    select = starts_with("info_"),
    by = sex,
    labels = TRUE,
    output = "tinytable"
  )
}
```

`table_outcome` *Describe one continuous outcome across several groupings*

Description

Summarizes *one* continuous outcome across the levels of *several* categorical variables, one block of rows per variable. It is the inverse layout of `table_continuous()`, which puts several outcomes in rows and one grouping in columns.

Each block reports the outcome's statistics level by level, plus its own group comparison on the block's header row, and an `Overall` row gives the marginal summary of the whole analytic sample.

Usage

```
table_outcome(
  data,
  outcome,
  select,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  overall = TRUE,
  drop_na = FALSE,
  weights = NULL,
  rescale = getOption("spicy.rescale", FALSE),
  test = c("welch", "student", "nonparametric"),
  p_value = NULL,
  p_adjust = "none",
  statistic = FALSE,
  show_n = TRUE,
  show_columns = NULL,
  effect_size = c("none", "auto", "hedges_g", "eta2", "r_rb", "epsilon2"),
  effect_size_ci = FALSE,
  ci = TRUE,
  ci_level = 0.95,
  digits = 2,
  effect_size_digits = 2,
  p_digits = 3,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  indent_text = "  ",
  indent_text_excel_clipboard = strrep(" ", 6),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  user_na = TRUE,
  style = NULL,
  by
)
```

Arguments

data	A data frame.
outcome	The continuous outcome, unquoted or as a string. Exactly one column.
select	The grouping characteristics to describe the outcome across, as a tidyselect expression or a character vector of column names. One block of rows per variable, in the order given, and in each block one row per level: the levels of

a factor in their order, any other variable sorted, as the groups of by are in table_continuous() (text in the C locale, FALSE before TRUE, numbers ascending, a labelled column by its value labels), whatever the order of the rows. As everywhere in the family, select is what structures the rows. The outcome and the weights column are not part of the selection: a selection helper skips them, and naming one in select, or giving its position, is an error. A name that is not a column of data is an error (class spicy_missing_column), and so is an empty selection.

labels

How each variable is named in the table: in the rows, in the title, and in the notes. The outcome follows the same rule as the select variables. Four forms:

- labels = FALSE shows the column name.
- labels = TRUE shows the column's label attribute (as set by haven, labelled, or label_from_names()) when it is present, not NA, and not empty, and the column name otherwise.
- labels = c(bmi = "Body mass index"), a named character vector, gives the columns it names these labels. Every other column follows options(spicy.var_labels): its name when the option is unset or FALSE, its label attribute when it is TRUE.
- labels = var_labels(bmi = "BMI (kg/m2)") gives the columns it names these labels, and every other column its label attribute, whatever the option says. var_labels() alone is labels = TRUE.

A name that matches no column of the table warns (class spicy_unknown_name) and is left out, so one vector can still serve every table of a project. The default reads options(spicy.var_labels), and a TRUE or FALSE typed in the call wins over it. Any other value, NULL included, is an error. A variable label documents the data file; a published table carries the wording you set here. note_labels gives a variable a shorter label in the title and the notes.

note_labels

The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one labels gives it: a named character vector, note_labels = c(sex = "sex at birth"), or a var_labels() object, which reads as one. labels can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of labels, and so do the title and the notes for a variable note_labels does not name. A name that matches no column warns and is left out, as in labels, and NULL (the default) renames nothing. A note label that another variable of the table shows is an error (class spicy_label_collision): choose another one, or relabel the other variable.

overall

Show the marginal Overall row (default TRUE).

drop_na

Drop rows with a missing select value from that block. The rule of the family: the categories a table counts keep their missing values as a row by default, as in table_categorical(); the values it computes on lose them, counted in the note, as in table_continuous(). The blocks of this table are categories, so FALSE (the default) shows the missing values of a select variable as a (Missing) level, excluded from the comparison, and a level displayed with that label is an error, as in table_categorical(); the missing values of the outcome are always removed and counted in the note. TRUE removes those of

a `select` variable from its block, and the note lists the variable after the outcome ("Missing values removed: age (3), sex (2)."): a missing outcome takes the row out of every block, a missing `select` value only out of its own block. Under `drop_na = TRUE`, a grouping that no observation holds is an error (class `spicy_invalid_data`).

`weights` Optional case weights: an unquoted column name, a character column name, or a numeric vector of length `nrow(data)`. Weights must be non-negative and finite; rows with `NA` or zero weight leave every statistic (including `Min` / `Max`) and `NA` weights are disclosed in the table note. The weighted formulas follow the frequency-expansion convention – see the **Weights** section. The weighted table names its weights in the note ("Statistics weighted by ..."). Group tests and effect sizes are not computed under weights: use `table_continuous_lm()` for weighted comparisons.

`rescale` Logical. If `TRUE`, weights are first normalized so that they sum to the number of observations used for each variable – the same `rescale` grammar as `table_categorical()`. This is the sampling-weights reading: results become invariant to the scale of the weights, and the weighted SD then equals Stata's `[aweight]` / `survey::svyvar()` value exactly. `FALSE` uses the weights as given (frequency reading). The default reads `options(spicy.rescale)`, `FALSE` when it is unset.

`test` Group comparison for every block: `"welch"` (default), `"student"` or `"nonparametric"`. An outcome with no variance in a block has no test there, whichever is chosen, with a warning (class `spicy_undefined_stat`); so has an outcome constant within each level of a block (a residual variance of zero) under `"welch"` and `"student"`, while the rank tests, which read the order of the values, run. The block's test cells (test, p-value, effect size) show the undefined mark, as do those of any block whose comparison cannot run (too few observations in a level, a test that fails, a non-finite effect size).

`p_value` Show the p-value column: `TRUE`, `FALSE`, or `NULL` (the default), which shows it, every block being a comparison.

`p_adjust` Multiple-comparison adjustment of the p-value column, applied to the family of every test in the table (one p-value per block). One of `"none"` (default), `"holm"`, `"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"` / `"fdr"`, or `"BY"`, as in `table_regression()`; delegated to `stats::p.adjust()`. Needs the p-value column. See *Multiple-comparison adjustment*.

`statistic` Logical. If `TRUE`, the test statistic of each block is shown in an additional column, with or without the *p*-value: `p_value` and `statistic` are independent. Defaults to `FALSE`.

`show_n` Logical. If `TRUE`, includes an unweighted `n` column in the printed ASCII table and in every rendered output (`tinytable`, `gt`, `flextable`, `word`, `excel`, `clipboard`). Set to `FALSE` to drop the `n` column structurally from those outputs (no empty placeholder, no spanner). The `n` column is always present in `output = "long"` for downstream programmatic access. Defaults to `TRUE`. Ignored (with a warning) when `show_columns` is supplied.

`show_columns` Statistics to display, as a character vector of the tokens of `table_continuous()` (see *Choosing the statistics*). `NULL` (the default) shows the mean, SD, min, max, the mean CI (see `ci`) and `n` (see `show_n`).

effect_size, effect_size_ci
: Effect size per block and its confidence interval, as in table_continuous() (a logical effect_size is an error there and here).

ci
: Logical. If TRUE, includes the mean confidence interval columns (<level>% CI LL / <level>% CI UL) and their spanner in the printed ASCII table and in every rendered output (tinytable, gt, flextable, word, excel, clipboard). Set to FALSE to drop both columns and the CI spanner structurally from those outputs (no empty placeholders, no border lines under an empty header). The CI bounds are always present as ci_lower / ci_upper in output = "long" for downstream programmatic access. Defaults to TRUE. The CI level is taken from ci_level. Ignored (with a warning) when show_columns is supplied.

ci_level
: Confidence level for the mean confidence interval (default: 0.95): a number strictly between 0 and 1.

digits
: Number of decimal places of the descriptive statistics (default: 2). A whole number: a fraction is an error, as for every *_digits argument. Test statistics keep two decimals.

effect_size_digits
: Number of decimal places for effect-size values in formatted displays (default: 2).

p_digits
: Integer >= 1. Number of decimal places used to render *p*-values in the p column (default: 3, the APA Publication Manual standard). Both the displayed precision and the small-*p* threshold derive from this argument: p_digits = 3 prints .045 and <.001; p_digits = 4 prints .0451 and <.0001; p_digits = 2 prints .05 and <.01. Useful for genomics / GWAS contexts with very small *p*-values, or for journals using a coarser convention. Leading zeros are always stripped, following APA convention.

decimal_mark
: The decimal mark, a single character: "." (the default) or "," in most styles; the Lancet style of spicy_style() uses the middle dot.

align
: Horizontal alignment of numeric columns in the printed ASCII table and in the tinytable, gt, flextable, word, excel, and clipboard outputs. The first column (Variable) and Group (when present) are always left-aligned. One of:

  - "decimal" (default): align numeric columns on the decimal mark, the standard scientific-publication convention used by SPSS, SAS, and LaTeX siunitx. Numeric cells are pre-padded with figure-spaces (U+2007, digit-width) so every string in a column has the same width with the decimal mark at the same internal position; centering those uniform-width strings then stacks the decimal points vertically. The same pad-then-center strategy is applied on every rendering engine (gt, tinytable, flextable, word, ASCII print) for a homogeneous rendering, matching table_regression() and table_continuous_lm(). The clipboard output is delimited text meant to be parsed rather than read at a fixed width, so its cells travel unpadded (a padded number pastes as text next to an unpadded number).
  - "center": center-align all numeric columns.
  - "right": right-align all numeric columns.

  "center" and "right" reach the excel output too. "decimal" does not: Excel cells are written unpadded, because cell-string padding does not align decimals

under a proportional font, so the workbook keeps the engine's own convention instead – counts and the *p*-value right-aligned, the other numeric columns centered. Same default and same three values as table_continuous_lm(), whose excel output still uses that convention at every align.

note — The note under the table. NULL (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the jama and lancet themes; see note_label in spicy_style()), then, in a table_regression() with stars, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks ("\n") included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (note = "Note. Data weighted."), and the legend of the stars stays below it. "" keeps that legend alone, and no note at all without stars. FALSE removes the note from every output, the legend included.

note_add — Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in attr(x, "note"), and the sentences the table writes stay. With a string in note, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An NA or an empty string is an error, and so is note_add with note = FALSE, which leaves no note to add to.

output — One of "default", "data.frame", "long", or a rendering engine: "tinytable", "gt", "flextable", "excel", "clipboard", "word". "data.frame" is the displayed table with its numbers, "long" the compute frame with every statistic, as in table_continuous(); see *Value*.

indent_text — Prefix used for modality labels in report table building. Defaults to " " (two spaces).

indent_text_excel_clipboard — Stronger indentation used in Excel and clipboard exports. Defaults to six non-breaking spaces.

excel_path — File path for output = "excel", ending in ".xlsx", in a directory that exists.

excel_sheet — Sheet name for output = "excel". NULL (the default) uses "Outcome".

clipboard_delim — Delimiter for output = "clipboard" (default: "\t"). A cell holding the delimiter itself, a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

word_path — File path for output = "word", ending in ".docx", in a directory that exists.

user_na — Logical. If TRUE (the default), declared missing values are treated as missing: those of the outcome leave its statistics and are disclosed in the table note (Declared missing values removed: ...), those of a select variable join its (Missing) level under drop_na = FALSE. If FALSE, the declared codes are ordinary values. See the "Declared missing values" section of freq().

style — A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes

DEFAULTS – any argument you pass explicitly wins over it. Set `options(spicy.style = )` for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; `?spicy_style` lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

`by` Defunct. The grouping characteristics are selected with `select` since spicy 0.13.0; supplying `by` is an error.

Value

Depends on `output`:

- `"default"`: a `spicy_outcome_table` (a `data.frame` subclass, with the parent class `"spicy_table"`), printed when the call is bare; its columns are the table's compute frame, and the display frame and the typed view are attached.
- `"data.frame"`: the displayed table with its numbers, what as.data.frame() of the default object returns: one row per printed row, the key columns `variable` (the outcome on the `Overall` row, the grouping variable elsewhere), `label`, `level` and `.row_role` (`"summary"` for the `Overall` row, `"factor_header"`, `"level"`, or `"missing"`), then the printed columns under their names (`"M"`, `"SD"`, ..., `"n"`, `"p"`), a cell that prints several numbers keeping the others beside it as in `table_continuous()`.
- `"long"`: the compute frame, one row per printed row with every statistic the table computes, whatever it displays: `variable`, `label`, `level`, `.row_role`, `mean`, `sd`, `min`, `max`, `ci_lower`, `ci_upper`, `median`, `q1`, `q3`, `iqr`, `med_ci_lower`, `med_ci_upper`, `n`, `weighted_n`, the test of each block on its header row (`test_type`, `statistic`, `df`, `df.residual`, `p.value`, and `p_unadjusted` under `p_adjust`) and its effect size (`es_type`, `es_value`, `es_ci_lower`, `es_ci_upper`, `NA` when none was requested).
- `"tinytable"`, `"gt"`, `"flextable"`: the rendered table.
- `"excel"` / `"word"`: writes the file and returns its path invisibly.
- `"clipboard"`: copies the table and returns the display `data.frame` invisibly.

The default object and the two data frames carry the whole note as printed, its lead-in included, in `attr(x, "note")` (`NULL` when there is nothing to say). The default object also keeps the title it prints in `attr(x, "title")`.

Which shape do I need?

Several continuous variables across *one* grouping is `table_continuous()` (`select = , by = `). One continuous variable across one or several groupings is this function. A single `select` is legitimate here – it is the natural way in when you know more groupings are coming – but with several outcomes and one grouping, the sibling is the table you want.

Choosing the statistics

`show_columns` takes the same tokens as `table_continuous()`, with the same meanings; see the *Choosing the statistics* section of `?table_continuous` for the vocabulary. Only the character-vector form is accepted here: there is one outcome, so a per-variable list would name nothing.

Weights

`weights` applies the frequency-expansion convention of the family: all weights 1 reproduces the unweighted table, and integer weights reproduce the STATISTICS of the data duplicated that many times – `n` stays the raw count of rows that carried the weights. Rows with a missing or zero weight leave the analytic sample; the note counts the missing ones.

`rescale` is the switch between the two readings of a weight: the frequency reading above, and the sampling-weight reading, where the weights are normalized to sum to the sample size. See the Weights section of `table_continuous()` for the choice in full.

`rescale = TRUE` normalizes the weights over the outcome's whole surviving sample, once, never per level – a per-level rescale would destroy the relative weights across levels, which is the entire information a sampling weight carries into this table. The means are unchanged by it; the standard deviations move, because their denominator is `sum(w) - 1`.

A weighted table refuses the group comparison. The estimates and their interval have no weighted version here, and a p-value or an effect size silently computed unweighted beside weighted descriptives is the one thing that must not happen: set `p_value = FALSE` (and `statistic = FALSE`, `effect_size = "none"`), or use `table_continuous_lm()` for a weighted comparison. The order-statistic median interval is refused for the same reason.

Blocks and the group comparison

Every block is a separate one-way comparison of the outcome across the levels of that variable. Nothing in this table adjusts one block for another, and the table note says so. Read the blocks as a set of bivariate descriptions, not as a model.

Each block chooses its test independently: with two observed levels `test = "welch"` is the Welch t-test, with three or more it is the Welch one-way ANOVA, and `test = "nonparametric"` is the Wilcoxon rank-sum or the Kruskal-Wallis test on the same rule. A block with fewer than two observed levels, or with a level holding a single observation, is not tested; its statistics stay empty and the other blocks are unaffected.

A level where the outcome has no observed value (typically a question not asked there) has no data, which is not a missing answer: its cells show an en dash, *n* included, the block compares the levels with data, the rows of that level are not counted in the outcome's missing values (whichever block they fall in), and the note says so ("No data for X in G.", X the label of the outcome and G the level). The `Overall` row stays the whole analytic sample.

The `Overall` row

`overall = TRUE` puts the marginal summary of the whole analytic sample on the first row. Under the default `drop_na = FALSE` the levels of every block partition that sample – the `(Missing)` display level included – so each block's counts add up to the `Overall` count exactly, which is what makes it a usable denominator.

The row reads **Overall**, not *Total*, and the distinction is deliberate. *Total* is the word of a COUNT margin: the column of `table_categorical()` where frequencies add up. This row is the whole analytic sample, where a mean is recomputed over every observation and nothing is added. A mean is not a total.

A `select` variable displayed as `"Overall"` is an error (class `spicy_label_collision`): choose another label in `labels`, or give the row another label with `options(spicy.labels = list(row_overall = "Whole sample"))`.

Choosing the `select` columns

The canonical form is `select = where(is.factor)`, or an explicit enumeration. Negation (`select = -c(x, y)`) is not recommended: it sweeps in every remaining column, and a numeric one becomes a block with one LEVEL per distinct value, in ascending order. A variable producing more than 20 levels raises a warning for that reason – an arbitrary threshold, but a sixty-row block where a reader expects a handful of categories is not a table.

A `haven_labelled` column used as `select` names its levels by their value labels (`"[1] Women"`), in the order of its `labels` attribute when every observed value carries a label and by ascending code otherwise, as a labelled by names its groups in `table_continuous()` and as the rows of a labelled variable of `table_categorical()` read.

The table note

One note sits under the table. It opens with the sample size, as in `table_categorical()`, "N = 159.": the rows of the data, a row with a missing weight left out, rows rather than weights under `weights` ("N = 60 (unweighted)."), the outcome's complete cases in the n of the `Overall` row. It then states what left the analytic sample, which group comparison ran in each block, what the displayed columns mean, and how the blocks and the `Overall` row are to be read. The rendering engines carry the same sentence as a table note. On the `"tinytable"` route it is set one size down; `options(spicy.note_style)` governs that (see `table_regression()`).

Multiple-comparison adjustment

The p-values of a descriptive table describe: they test no prespecified hypothesis. STROBE (explanation of item 14) and the SAMPL guidelines advise against significance tests in descriptive and baseline tables, hence the default `p_adjust = "none"`.

When a journal or a protocol asks for adjusted p-values anyway, `p_adjust` applies `stats::p.adjust()` to the family of every test in the table: one p-value per variable (one per block in `table_outcome()`), never the rows of one variable, which share one test. `"holm"` suits a confirmatory set of comparisons, `"BH"` a screening one; `table_regression()` lists the methods. A test that returned no p-value stays out of the family.

The adjustment runs before any formatting, so the `p` column and its `<.001` threshold show the adjusted values, and the table note gives the number of tests and the method. The raw p-values stay in the object: a `p_unadjusted` column next to the p-value column of the returned data frames, and `col_meta$p$p_unadjusted` (with the method in `p_adjust` and the family size in `p_adjust_m`) in `as_structured()`. Without `by`, or with `p_value = FALSE`, the table shows no p-value to adjust, and a `p_adjust` other than `"none"` is an error.

See Also

`table_continuous()` for the transposed shape, `table_categorical()` for categorical outcomes.

Examples

```
table_outcome(sochealth, bmi, select = c(sex, smoking))
table_outcome(sochealth, wellbeing_score, select = where(is.factor))
```

table_regression *Regression coefficient summary table*

Description

Publication-ready coefficient table from one or more fitted `lm` / `glm` models. Supports standardized coefficients (β), average marginal effects (AME), partial effect sizes (f^2 / η^2 / ω^2 for `lm`; partial χ^2 for `glm`), pseudo-R^2 (`glm`), and a full vocabulary of variance estimators (classical / HC* / cluster-robust with Satterthwaite-corrected df / bootstrap / jackknife). `glm` covers binomial / poisson / Gamma / inverse.gaussian / quasi families with any link.

Usage

```
table_regression(
  models,
  vcov = "classical",
  cluster = NULL,
  ci_level = 0.95,
  ci_method = c("wald", "profile", "boot_percentile", "hdi"),
  boot_n = 1000L,
  tau = NULL,
  at_time = NULL,
  standardized = c("none", "refit", "posthoc", "basic", "smart", "pseudo"),
  exponentiate = FALSE,
  p_adjust = "none",
  show_columns = NULL,
  keep = NULL,
  drop = NULL,
  show_intercept = TRUE,
  show_thresholds = TRUE,
  show_components = TRUE,
  intercept_position = c("first", "last"),
  factor_layout = c("grouped", "flat"),
  reference_style = c("row", "annotation", "footer", "none"),
  reference_label = NULL,
  show_fit_stats = NULL,
  fit_stats_layout = c("first_col", "merged"),
  show_re = TRUE,
  re_scale = c("sd", "variance"),
  re_columns = c("est", "se", "ci"),
  re_test = c("none", "lrt", "rlrt"),
  re_ci = c("wald", "profile"),
  model_labels = NULL,
  outcome_labels = NULL,
  stars = FALSE,
  nested = FALSE,
  digits = 2L,
```

```
  p_digits = 3L,
  effect_size_digits = 2L,
  fit_digits = 2L,
  ic_digits = 1L,
  decimal_mark = ".",
  align = c("decimal", "center", "right"),
  padding = 0L,
  labels = getOption("spicy.var_labels", FALSE),
  note_labels = NULL,
  title = NULL,
  note = NULL,
  note_add = NULL,
 output = c("default", "data.frame", "long", "tinytable", "gt", "flextable", "excel",
    "clipboard", "word"),
  excel_path = NULL,
  excel_sheet = NULL,
  clipboard_delim = "\t",
  word_path = NULL,
  word_template = NULL,
  style = NULL
)
```

Arguments

models	A fitted model object, or a list of such fits (named or unnamed; classes may be mixed). Single fits are auto-promoted to a 1-element list. A broad set of model classes is supported – linear / generalized linear (`lm`, `glm`, `MASS::glm.nb`), mixed-effects (`lmer`, `lme`, `glmmTMB`), survival (`coxph`, `survreg`), ordinal (`polr`, `clm`, and the cumulative-link mixed models `ordinal::clmm` and `glmmTMB` with `family = glmmTMB::ordinal()`), `mgcv::gam`/`bam`, `betareg`, `mlogit`, `survey::svyglm`, `rms` (`ols`/`lrm`/`cph`/`Glm`), and Bayesian (`rstanarm`/`brms`), among others. An unsupported class raises `spicy_unsupported`. Raw data + formula is not accepted – fit-only API.
vcov	Variance-covariance estimator: `"classical"`, `"HC0"`-`"HC5"`, `"CR0"`-`"CR3"`, `"bootstrap"`, or `"jackknife"`. A scalar is recycled to all models; a list (one string per model) allows mixed estimators. Default `"classical"`. The resampling estimators (`lm` / `glm`, including `MASS::glm.nb`) refit each replicate on resampled rows of the fixed evaluated design; for `glm.nb` the dispersion parameter theta is held at its full-sample estimate (Stata's `nbreg, vce(bootstrap)` re-estimates it per replicate – the two conventions differ slightly). Resampling that fails on nearly every replicate raises `spicy_resampling_failed` rather than silently reporting a different estimator. Quantile regression (`quantreg::rq`) uses its own estimator family instead: `"classical"` resolves to the heteroskedasticity-robust `"nid"` sandwich (Hendricks-Koenker, Hall-Sheather bandwidth – quantreg's own large-sample default and the Stata `vce(robust)` analog), with `"iid"` (Koenker-Bassett homoskedastic, Stata `vce(iid)` parity), `"ker"` (Powell kernel) and `"rank"` (rank-score inversion: genuine CIs, no SE / t / p – those cells render as dashes) as opt-ins, and `"bootstrap"` running quantreg's native `boot.rq` (`bsmethod = "xy"`;

with `cluster =`, the Hagemann 2017 wild gradient cluster bootstrap – the one cluster-robust route for `rq`; `CR*` and `HC*` are refused for `rq`, as is `"jackknife"`, whose leave-one-out form is inconsistent for quantiles). With `nested = TRUE`, the change test of each pair uses the same estimator (a Wald test; see *Hierarchical (nested) model comparison*). See *Inference and standard errors*.

`cluster` Cluster identifier for cluster-robust variance (used when `vcov` is `"CR0"`-`"CR3"` or a cluster-bootstrap / cluster-jackknife). Three accepted forms (see *How to specify* `cluster` in the details):

- Formula: `~region`, `~region:year` (recommended).
- String column name: `"region"`.
- Atomic vector of length `nobs(fit)`: `df$region`, `interaction(df$region, df$year)`, ... (for keys derived on the fly).

For multi-model use, pass a list of one form per model (mix-and-match allowed). A bare unquoted name (`cluster = region`) is not column selection: it is evaluated as an ordinary variable (the vector form when `region` exists in the calling environment, a migration error pointing at `~region` / `"region"` otherwise). Default `NULL` (no clustering).

`ci_level` Confidence level for all reported CIs (B, β, AME, partial effect sizes). Default `0.95`.

`ci_method` CI construction. `"wald"` (default) uses `estimate +/- z x SE` (`t x SE` for `lm`). `"profile"` (`glm` and ordinal `MASS::polr` / `ordinal::clm`) uses the profile-likelihood CI from `stats::confint()` – `MASS::confint.glm()` for `glm`, `confint.polr()` / `confint.clm()` for ordinal fits – asymmetric, exact for likelihood-based inference (Venables & Ripley *MASS* Section 7.2). Only the CI bounds change; estimate, SE, statistic and p-value remain Wald. For ordinal fits the profile covers the predictor coefficients (the cut-point thresholds stay Wald), and a robust `vcov` takes precedence (its Wald-robust CIs are used instead, with a consolidated warning). `"profile"` with `lm` raises `spicy_invalid_input`. `"boot_percentile"` (requires `vcov = "bootstrap"` for every model) replaces the coefficient CI bounds with equal-tailed percentile intervals of the bootstrap replicates (the `boot::boot.ci()` `type = "perc"` convention; Davison & Hinkley 1997, ch. 5), reusing the same resamples as the bootstrap SEs. Only the CI bounds change; estimate, SE, statistic and p-value remain Wald from the bootstrap covariance – the Stata convention (normal-based table CIs by default, percentile on request via `estat bootstrap`). AME and standardized-beta CIs are not covered. With `exponentiate = TRUE` the percentile bounds are exponentiated (percentile intervals are transformation-respecting). `"hdi"` (Bayesian `stanreg` / `brmsfit` fits only) replaces the default equal-tailed credible interval with the highest-density interval – the shortest interval containing `ci_level` of the posterior draws (Kruschke 2015), relabeling the column header `95% HDI`. Unlike the equal-tailed interval the HDI is not transformation-invariant, so under `exponentiate = TRUE` it is recomputed on the exponentiated draws rather than transformed. Requesting `"hdi"` for a frequentist fit, or `"profile"` / `"boot_percentile"` for a Bayesian fit, raises `spicy_invalid_input`.

`boot_n` Number of bootstrap replicates when `vcov = "bootstrap"`, and of the bootstrap of the RMST and risk-difference columns: a positive whole number, as in `table_continuous_lm()` and `table_multiresponse()`. Default `1000L`, the

default of table_continuous_lm(), which computes the same bootstrap variance. That variance, of a lm or glm fit, needs at least 10 valid replicates: a smaller boot_n is an error. Typed in a table that runs no bootstrap, it is ignored with a warning (class spicy_ignored_arg).

tau — RMST horizon: the "rmst" column family reports the restricted-mean-survival-time difference over [0, tau] (coxph and survreg fits). A positive time on the outcome's scale, or "minmax" for the smallest per-group maximum follow-up (the resolved value is disclosed in the table note). No default: the horizon defines the estimand. Stratified fits (strata()) are supported: standardization keeps each subject's own stratum baseline, disclosed in the table note.

at_time — Landmark time for the "risk_diff" column family: the difference in cumulative incidence at at_time (coxph and survreg fits). A positive time on the outcome's scale; no default.

standardized — Standardization method for the "beta" column. One of "none" (default), "refit", "posthoc", "basic", "smart", "pseudo". "pseudo" is *glm only* (Menard 2011 fully-standardized); using it with lm() raises spicy_invalid_input. Supported classes: lm, glm (incl. MASS::glm.nb), the mixed engines (lmer / glmer / glmmTMB / nlme::lme), and fixed-effects stan_glm-style stanreg fits for the algebraic flavors "posthoc" / "basic" / "smart" – exact affine rescales of the posterior draws (median, MAD SD and credible bounds all scale by the same positive SD ratio; the beta stays on the link scale under exponentiate = TRUE). Standard-formula fixed-effects brmsfit models are supported too (their design matrix is recovered through **insight**); the scale factors are engine-invariant, so the same model fit with rstanarm or brms gets identical betas up to sampling noise. Bayesian "refit" (would re-run the sampler) and "pseudo" (per-draw latent variance; planned) are refused. Also refused: multilevel fits (stan_glmer / stan_lmer and brm() with group-level terms; a Bayesian beta would need an explicit sd(Y) decomposition that the frequentist mixed engines resolve by refitting on z-scored data), non-GLM stanreg subclasses (stan_polr, stan_betareg), and brms formulas with distributional, multivariate or special terms (mo(), s(), ...). In all refused Bayesian cases, standardize predictors before fitting instead. Any other class raises spicy_unsupported_standardized rather than rendering an empty beta column. See the *Standardized coefficients* section.

exponentiate — Logical. When TRUE, B, the CI bounds, and the SE (delta method: SE_OR = OR x SE_log-odds) are exp()-transformed for links where the result is a ratio: logit (OR), log (IRR / RR / MR per family, generic exp(B) for other log-link families – a genuine ratio of means), and binomial / ordinal cloglog (HR; grouped-time proportional hazards, Prentice & Gloeckler 1978). For the *ordinal* (cumulative) cloglog model the displayed HR is exp(-B), not exp(B): the cumulative parametrization cloglog P(Y <= j) = zeta_j - xB places the hazard of the grouped event on -B, so exp(B) would be the reciprocal of the hazard ratio (the CI endpoints are negated and swapped accordingly, and the table note discloses the convention; a cloglog clm with nominal = terms refuses exponentiate because threshold-side coefficients have no covariate-HR reading). The statistic and p-value stay on the link scale (invariant under monotone transformation). Identity-link fits are left untouched (a spicy_ignored_arg warning fires when no model in the table exponentiates), so mixed lm + logit tables keep working. Any other link (probit, cauchit, inverse – the Gamma() default –, 1/mu^2, sqrt, ordinal loglog, ...)

raises spicy_invalid_input: the exponential of such a coefficient has no ratio interpretation, and reporting it would mislabel the estimate. Report response-scale effects for those models via the AME column instead. Note that the displayed CI is the link-scale CI with exponentiated endpoints (Wald or profile per ci_method), so it is asymmetric around the ratio and cannot be reconstructed as estimate ± z × SE; the delta-method SE follows the Stata convention ([R] logistic, Methods and formulas: se(OR) = OR × se(b)) and the log-scale CI endpoints carry the uncertainty more faithfully than this SE. The footer states the SE scale and the CI asymmetry whenever an SE column is displayed. Default FALSE.

p_adjust — Multiple-comparison adjustment method applied to the family of estimated coefficient p-values within each model (intercept and reference rows excluded). One of "none" (default), "holm", "hochberg", "hommel", "bonferroni", "BH" / "fdr", or "BY". Delegated to stats::p.adjust(); applied per-model and per estimate_type (B and AME p-values are adjusted independently within their own families). Active adjustments are documented in the footer (method + family size). See the *Multiple-comparison adjustment* section for when this is and is not appropriate.

show_columns — Character vector of tokens selecting the per-coefficient columns and their display order. Accepts **atomic tokens** ("b", "se", "ci", "t", "p", "beta", "n", "n_events", "pd" (probability of direction, Bayesian fits only), "rhat" / "ess_bulk" / "ess_tail" / "mcse" (per-coefficient sampler diagnostics and the Monte Carlo standard error of the displayed posterior median, all-Bayesian tables only), "ame", "ame_se", "ame_ci", "ame_p", "rmst" + "rmst_se" / "rmst_ci" / "rmst_p", "risk_diff" + its _se / _ci / _p companions, "partial_f2" + "partial_f2_ci", "partial_eta2" + "partial_eta2_ci", "partial_omega2" + "partial_omega2_ci", "partial_chi2", "r2" / "adj_r2" (per-fit variance explained, lm screens)) and **group tokens** ("all_b", "all_b_compact", "all_b_full", "all_beta", "all_ame", "all_ame_compact", "all_f2", "all_eta2", "all_omega2"). See *Vocabulary tokens* in the details for the full enumeration. Default NULL selects a context-aware layout: "all_b" (single model) or "all_b_compact" (multi-model, and the single-multinomial outcome-as-columns layout, which has the same width pressure – restore CIs with atomic tokens, e.g. c("b", "se", "ci", "p")). The "p" token is always the B / beta p-value; for the AME-specific p-value use "ame_p".

keep — Character vector of regexes. Only coefficient rows whose term name (as in stats::coef() – e.g. "wt", "cyl6", "factor(cyl)8") matches at least one pattern are kept. The intercept and the subordinate blocks (thresholds, random effects) are exempt from keep and drop: show_intercept, show_thresholds and show_re govern them. Mutually exclusive with drop. Filtering is a display choice; p_adjust runs against the full coefficient family before filtering. Default NULL (no filter).

drop — Character vector of regexes. Coefficient rows matching any pattern are removed, with the exemptions of keep. Mutually exclusive with keep. Default NULL.

show_intercept — Whether to display the intercept row. Default TRUE (APA convention). Hide via FALSE.

show_thresholds — For ordinal cumulative-link models (MASS::polr, ordinal::clm, ordinal::clmm,

glmmTMB with the ordinal family), whether to display the estimated category thresholds (cut-points) as a subordinate "Thresholds" block of rows below the predictors, carrying B / SE / CI / p like the predictor rows. Default TRUE. FALSE collapses them to a compact one-line footer note instead. Thresholds are reported on the log-odds (B) scale and are **never exponentiated** (under exponentiate = TRUE their rows stay on the log-odds scale). Has no effect on non-ordinal models, and the rows are shown only when a coefficient column ("b"/"beta") is in show_columns.

show_components

For models with secondary components, whether to display them as labeled subordinate blocks of rows below the primary (count / conditional / location) coefficients. Default TRUE:

- pscl::zeroinfl and glmmTMB(ziformula =): a Zero-inflation block – the model for the probability of a **structural (excess) zero**.
- pscl::hurdle: a Zero hurdle block – the model for the probability of a **nonzero count** (note the opposite direction vs zero-inflation; the footer names each block's meaning).
- glmmTMB(dispformula =): a Dispersion block (only when dispersion was actually modeled; log scale, never exponentiated).
- ordinal::clm(scale = ~): a Scale effects block – the covariate effects on the **log standard deviation of the latent response**. Never exponentiated: their exponential is a ratio of latent standard deviations, not an odds ratio.

Component rows carry full Wald inference (B / SE / z / p / CI), join the p_adjust family, and take significance stars. Under exponentiate = TRUE a component is exponentiated **only when its link makes the result an odds ratio** (the logit zero components); probit / cauchit / cloglog zero links and count-type hurdle zero parts stay on the link scale, disclosed in the footer. FALSE omits the blocks (the title still names the model type).

intercept_position

Where to place the intercept when shown. "first" (default, APA) or "last" (Stata-style, intercept just above the fit-stats footer). Ignored when show_intercept = FALSE (with spicy_ignored_arg warning).

factor_layout Layout of factor predictors. Applies to **any categorical predictor** – factor, ordered, character, or logical (R coerces the latter two to factors at fit time). Two options:

- "grouped" (default): the variable name on its own header row ending with : (e.g., education:); each level follows as an indented sub-row with the bare level name. APA convention.
- "flat": each non-reference dummy is one row with the <variable><level> form (e.g., educationUpper); no header, no indent. Econometrics convention.

reference_style

Rendering of factor reference levels. Four modes, distinguishing WHERE the reference information is exposed (in a row, inline, in the footer, or nowhere):

- "row" (default): explicit row Female (ref.) with en-dashes in all stat columns (NEJM / BMJ clinical convention). reference_label controls the suffix.
- "annotation": the row is dropped and the reference is shown inline. Under factor_layout = "grouped" the factor header reads education: [ref: Lower]; under factor_layout = "flat" the marker [vs Lower] is attached to the **first non-reference dummy** of each factor (subsequent dummies inherit the same reference).
- "footer": the row is dropped and a sentence Reference categories: education = Lower; sex = is added to the footer note. SAS PROC LOGISTIC / SPSS "Categorical Variables Codings" convention. Best for publication-grade dense multi-factor tables.
- "none": the row is dropped and no reference information is displayed anywhere. The user is responsible for stating the reference convention elsewhere (article text, table caption). Under factor_layout = "flat", an informational message is emitted to flag the silent omission.

Ordered factors with AME: under R's default contr.poly, ordered factors have B coefficients named .L / .Q / .C (orthogonal polynomial trends) which have no per-level reference semantics. When "ame" is in show_columns, however, the AME block is per-level contrasts against levels()[1]. A synthetic reference row anchored on levels()[1] is therefore emitted so the reader sees the AME baseline explicitly, with the same reference_style handling as plain treatment-coded factors. The [vs <ref>] annotation in "annotation" mode is attached to the first AME row, not to the polynomial-trend rows.

reference_label

Suffix shown after the reference level in reference_style = "row" mode. NULL (default) prints "(ref.)", translated under options(spicy.language =); a string you supply is printed as typed. Ignored by the other three modes, whose fixed wording follows the language: [ref: ...], [vs ...], and the footer line Reference categories: .... The suffix follows the level after a space, unless it opens on ,, ; or :, which attach to the level directly: a level such as T0 (2018) reads T0 (2018) (ref.); reference_label = ", ref." gives T0 (2018), ref.; reference_style = "annotation" moves the reference out of the row.

show_fit_stats Character vector of tokens for the model-level rows below the coefficients; row order follows token order. NULL (default) resolves class-aware:

- lm, estimatr::lm_robust(): c("nobs", "r2", "adj_r2"). estimatr::iv_robust() gets "nobs" alone – the 2SLS R-squared is not the classical one.
- glm, ordinal polr / clm: c("nobs", "pseudo_r2_mcfadden", "pseudo_r2_nagelkerke", "aic") (McFadden = Stata ologit default, Nagelkerke = SPSS PLUM).
- mixed lm + glm: the union of the two (the renderer en-dashes per cell the stat not defined for a given model class).

Under nested = TRUE the default is extended with the class-appropriate change-stat tokens (e.g. "r2_change", "f_change" for lm). See *Vocabulary tokens* (show_fit_stats subsection) and *Hierarchical (nested) model comparison* in the details for the full vocabulary.

`fit_stats_layout`

Layout of the fit-stat values (`n`, `R^2`, `AIC`, ...) within each model's column group. Two options:

- `"first_col"` (default): the value is placed in the FIRST numeric sub-column of each model (typically `B`); the model's remaining sub-columns (`SE`, `LL`, `UL`, `p`, ...) are left empty for that row. The APA Manual 7 Table 7.13 layout.
- `"merged"`: the model's numeric sub-columns are merged into a single wide cell containing the fit-stat value, centered under the model spanner. Stata `esttab` layout / *Econometrica* and *AER* journal convention. Resolves the mixed-precision look of `"first_col"` (an integer `n` row sharing the B column with two-decimal coefficients).

Cell merging is supported by `excel`, `flextable`, and `word` (via flextable). `gt`, `tinytable`, `clipboard`, and `default` (console) always render in `"first_col"` mode regardless of this setting:

- `gt` lacks a native row-spanning cell-merge API (`tab_spanner` covers columns, not row-cell ranges).
- `tinytable`'s `style_tt(colspan = N)` emits HTML `colspan` only on header rows, not on body cells.
- `clipboard` ships TSV plaintext.
- `default` ships fixed-width ASCII.

Decimal alignment of every numeric column is preserved in both modes: the `B` column decimal-aligns its coefficient values plus any fit-stat value(s) in `"first_col"` mode (native primitives handle the mixed-precision case), and trivially decimal-aligns in `"merged"` mode (the fit-stat values move out of the B column into the merged cell).

`show_re`

Logical. `TRUE` (default) renders the random-effects variance components of a mixed-effects fit (`lmer`, `glmer`, `glmmTMB`, `lme`, `clmm`) as a subordinate **"Random effects" block of table rows** below the fixed effects: one row per standard deviation / correlation per grouping factor, plus the residual, each with its estimate, SE, and CI in the shared coefficient columns. The group sizes (`N (groups)`) and the ICC render as fit-statistic rows; the footer reports the estimation method (`REML` / `ML`) and the likelihood-ratio test of the whole random part against the no-random-effects model, with the boundary-corrected chi-bar-squared p-value (Self & Liang 1987; Stram & Lee 1994). Variance-component rows deliberately carry **no per-row p-value**: a Wald test of a variance is invalid at the boundary of the parameter space, and no reporting guideline requests one (see the *Mixed-effects models* section of the Publication-ready regression tables article). `FALSE` suppresses the block. No effect on fits without random effects (`lm`, `glm`, `coxph`, ...).

`re_scale`

One of `"sd"` (default) or `"variance"`. Controls the display scale of the random-effects rows:

- `"sd"`: report the random-effect standard deviation σ (Gelman 2005, *Technometrics*: "*directly interpretable as the size of the variation across groups*"). Standard error and CI converted via the Delta method: $SE(\sigma) = SE(\sigma^2)/(2\sigma)$; $CI(\sigma) = \sqrt{CI(\sigma^2)}$.

- `"variance"`: report σ^2 (the canonical internal scale; SE and CI come straight from the Hessian / `nlme::intervals()` / `glmmTMB::confint()` without rescaling).

Correlation rows (ρ) are unitless and pass through either way.

`re_columns` Character vector. Subset of `c("est", "se", "ci")` controlling which cells of the random-effects rows are **displayed** (`"est"` is mandatory); deselected SE / CI cells render as an en-dash on those rows only. Useful for slimming output (`re_columns = "est"`) or for journals that want only standard errors (`re_columns = c("est", "se")`). Display-only: the underlying data (`broom::tidy()`, `as_structured()`) always carries the full SE + CI.

Note. Under the default `re_ci = "wald"` the SE and CI are Wald (`est ± z * SE`, clamped at 0 for variances). Wald can be optimistic near the variance boundary (Self & Liang 1987 chi-bar-squared); request boundary-respecting profile-likelihood intervals with `re_ci = "profile"` when robustness is critical. See the *Mixed-effects models* section of the Publication-ready regression tables article.

For `lmer` / `glmer` fits these SEs come from `merDeriv`, whose cost grows super-linearly with the number of observations (about a minute at n of roughly 2,700). Above `options("spicy.re_se_max_n")` (default `1000`) they are skipped: the rows keep their estimates, the SE / CI cells render as en-dashes, a table note states the omission, and a `spicy_caveat` warning points here. Raise the cap (e.g. `options(spicy.re_se_max_n = Inf)`) to force the computation, or test the random terms with `re_test = "lrt"`.

`re_test` One of `"none"` (default), `"lrt"`, or `"rlrt"`. Opt-in **per-term significance test** for the random-effect variance components, filling the otherwise-empty p column of the Random effects rows. Never a Wald test (invalid at the boundary sigma = 0):

- `"lrt"`: likelihood-ratio test of each random term vs the model refitted without it (the term's variance plus its covariances with the other terms of its bar), referred to the boundary-corrected chi-bar-squared mixture `0.5 chi2(q-1) + 0.5 chi2(q)` (Self & Liang 1987; Stram & Lee 1994). The reduction scheme matches `lmerTest::ranova()`; the mixture reference makes the p-value exact-asymptotic rather than conservative. A bar's intercept is tested only when it is the bar's single term. Supported: `lmer`, `glmer`, `glmmTMB`, and `lme` with a simple `random = ~ terms | group` structure.
- `"rlrt"`: exact restricted likelihood-ratio test with a simulated finite-sample null (`RLRsim::exactRLRT()`; Crainiceanu & Ruppert 2004). Only defined for a Gaussian `lmer` / `lme` fit with a single variance component.

The test statistic and df stay out of the displayed t/z column (they are chi-square-scale, not t/z) but are carried in `broom::tidy()` (`test_type "chibar2"` / `"rlrt"`). The whole-block LR test in the footer is unaffected. Correlation and residual rows are never tested (a correlation is tested jointly with its slope; the residual has no zero-variance null). Refits happen once per random term: expect a noticeable cost on large models. For structures outside these two routes (e.g. multiple variance components needing a finite-sample null), `pbkrtest::PBmodcomp()` offers a parametric-bootstrap LR test on the same nested pair of fits (Halekoh & Hojsgaard 2014) – run it directly and report its p-value alongside the table.

re_ci — One of "wald" (default) or "profile". Uncertainty route for the random-effect variance-component rows of lmer / glmer fits:

- "wald": SE and symmetric CI from the observed information (merDeriv), subject to the options("spicy.re_se_max_n") size cap (see re_columns).
- "profile": **profile-likelihood CIs** via confint(fit, method = "profile") – the route lme4 itself documents and defaults to. The intervals are asymmetric and respect the boundary at 0; no SE is shown (lme4's position: a symmetric SE misdescribes the skewed sampling distribution of a variance). Sidesteps the size cap entirely (roughly two seconds per variance parameter even at n ~ 7,000; glmer profiles cost more). The footer discloses the method, and likelihood intervals transform exactly, so re_scale = "variance" shows the profile CI of the variance itself.

glmmTMB and nlme::lme fits keep their engine-native CIs (TMB's sdreport; nlme's apVar) and refuse "profile".

model_labels — Per-model labels used as the **column-group spanner** above each model's subcolumns (console + gt / flextable / tinytable / Excel / Word renderers). NULL (default) resolves automatically; see *Multi-model semantics* for the full rule. A character vector of length length(models) overrides. Refused (error) for a single multinomial model: there the column groups are outcome categories, relabeled via outcome_labels.

outcome_labels — Optional **Outcome body row** override. NULL (default) hides the row entirely – under the multi-model spanner the DV is already visible above the data. A character vector of length length(models) forces an explicit Outcome row with those values (the spanner stays as "Model 1, ..." unless model_labels is also supplied). FALSE also suppresses the row. **Single multinomial model** (outcome-as-columns layout): repurposed as the category spanner override – a character vector with one label per non-reference outcome category (unique, in model order), e.g. c("Student vs Employed", ...); the reference category's AME-only group (when displayed) keeps its own name, and FALSE is a no-op (there is no Outcome row to suppress).

stars — Significance asterisks. FALSE (default, APA 7 Section 6.46) – no stars. TRUE – APA cutoffs c("*" = 0.05, "**" = 0.01, "***" = 0.001). A named numeric vector specifies custom thresholds, e.g. c("+" = 0.10, "*" = 0.05, "**" = 0.01, "***" = 0.001). With output = "excel" a marked estimate is written as text ("64.63***" is not a number); the cells with no marker stay numeric.

nested — Whether to inject pairwise change-statistic rows for adjacent models (M2 vs M1, M3 vs M2, ...). FALSE (default) – pure side-by-side display. TRUE – requires identical nobs and identical response variable across all models. Under a non-classical vcov, each pair is tested by a Wald test of the coefficients the current model adds, on that model's matrix, shown as "Wald F-change" or "Wald χ^2 (change)" by the regime of its coefficient tests. See *Hierarchical (nested) model comparison* for the rule, the classes it covers and its small-sample caution.

digits — Decimal places for general numeric tokens (b, beta, se, ci, t, f_change, lrt_change, deviance, deviance_change, ame, ame_se, weighted_nobs). Default 2L.

p_digits — Decimal places for p-values (p, ame_p, p_change). APA-strict: leading zero stripped, <.001 (or <.0001 etc. depending on p_digits) for small values. De-

fault `3L`; at least `1L`, as in the descriptive tables. Every `*_digits` argument is a whole number: a fraction is an error.

`effect_size_digits`
: Decimals for per-coefficient effect sizes (`partial_f2`, `partial_eta2`, `partial_omega2`). Default `2L`.

`fit_digits`
: Decimals for variance-explained / model-level effect-size fit stats (`r2`, `adj_r2`, `r2_change`, `adj_r2_change`, `omega2`, `f2`, `f2_change`, `sigma`, `rmse`). Default `2L`.

`ic_digits`
: Decimals for information criteria (`AIC`, `AICc`, `BIC`, and their `_change` form). Default `1L`.

`decimal_mark`
: The decimal mark, a single character: `"."` (the default) or `","` in most styles; the Lancet style of `spicy_style()` uses the middle dot. When `","` is used, the CI bracket separator switches to `"; "` automatically to avoid `"0,18 [0,07, 0,30]"` ambiguity. With `output = "excel"` a numeric cell is displayed by Excel with the *viewer's* locale separator, which a file cannot set, so a non-default mark makes the body go out as pre-formatted text (the default `"."` keeps every cell a real number).

`align`
: Numeric column alignment. `"decimal"` (default) – pre-pad cells so decimal marks line up vertically (publication-style). For CI cells (`[LL, UL]`) the left bracket, the LL decimal point, the comma separator, the UL decimal point, and the right bracket are independently aligned across rows. `"center"` and `"right"` apply the same alignment to every numeric column.

`padding`
: Non-negative integer giving the extra characters added to each data column's auto-computed width when the default `print` method renders the table. Default `0L` (compact – fits more models in the same console / page width). Use `2L` (Stata-like) or `4L` for a more spacious look. Headers stay centered above the data region regardless of padding.

`labels`
: How the variables are named in the table: the outcome in the title (and above its columns when the models have different outcomes), a numeric term in its row, and a factor in its header. Four forms:

  - `labels = FALSE` shows the names the model gives them (`bmi`, `age`, `sex`).
  - `labels = TRUE` shows each variable's `label` attribute (as set by `haven`, `labelled`, or `label_from_names()`) when it is present, not `NA`, and not empty, and the name otherwise: "Linear regression: Body mass index", "Age (years)", "Sex:". The attribute is read off the model frame, or off the data the model was fit on where the model frame lost it. Factor levels, interactions, and transformed terms (`log(income)`, `poly(age, 2)`) are not variables and keep their names, as do the coefficient rows of `factor_layout = "flat"`.
  - `labels = c(age = "Age (years)", sexMale = "Men")`, a named character vector, gives the variables and rows it names these labels. Every other variable follows `options(spicy.var_labels)`: its name when the option is unset or `FALSE`, its `label` attribute when it is `TRUE`.
  - `labels = var_labels(age = "Age in years")` gives the variables and rows it names these labels, and every other variable its `label` attribute, whatever the option says. `var_labels()` alone is `labels = TRUE`.

The names are model identifiers, not display text: the outcome, the term labels of `stats::terms()` (`"age"`, `"sex"`, `"age:sex"`, a factor's term naming its header), and the coefficient names (`"(Intercept)"`, `"sexMale"`), each naming one row. They keep that form whatever the language of the table. Classes with no `terms` component (`nls()`, `sampleSelection::selection()`) are keyed on their coefficient (or parameter) names alone. With several models, a name is looked up in the terms of all of them and lands on the rows where it exists. A name that matches nothing in the table warns (class `spicy_unknown_name`) and is left out, so one vector can still serve every table of a project. The headers of the subordinate blocks (`Random effects`, `Thresholds`, `Zero-inflation`, ...) are set by the package and cannot be relabeled here.

The default reads `options(spicy.var_labels)`, and a `TRUE` or `FALSE` typed in the call wins over it. Any other value, `NULL` included, is an error. A variable label documents the data file; a published table carries the wording you set here. `note_labels` gives the outcome a shorter label in the title.

`note_labels` The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one `labels` gives it: a named character vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and `NULL` (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`title`, `note` Override or suppress the auto-built caption / methodological footer. Three modes per argument, those `note` has in every table function:

- `NULL` (default): the package builds the standard caption ("Linear regression on <DV>" / "Hierarchical linear regression on <DV>" / ..., the outcome named by its label in `note_labels`, else in `labels`) and a methodological note, one paragraph in the order of every table of the package: what the model is (its family, design, scale of the coefficients, reference categories, ...), the cases a univariable screen fits on, the inference (VCV type, p-adjust method, ...), the glosses of the blocks, then the abbreviations the column headers and the rows of the fit statistics and of the random effects show ("AIC = Akaike information criterion"), defined as the style in force asks (`spicy_style(abbreviations = )`; under `"jama"` they open the note on a line of their own), and the sentences of `note_add`. Under `stars`, the legend of the stars follows on a line of its own.
- `FALSE`: the corresponding banner row is omitted from every output engine, for `note` the legend of the stars included. Use when the surrounding manuscript provides its own caption / note.
- character string (length 1): replaces the auto-built text. A `title` is printed verbatim. A `note` replaces the general note and is printed as typed (multi-line notes accepted via embedded `"\n"`) after the lead-in of the style, which a string opening with a lead-in of its own does not repeat, the sentences of `note_add` after it; the legend of the stars stays on its line below, and `note = ""` keeps that legend alone.

Validation messages, the spanner row, and the in-body change- stat rows are *not* affected – they belong to the table structure, not to the banner.

`note_add` Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in `attr(x, "note")`, and the sentences the table writes stay. With a string in `note`, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An `NA` or an empty string is an error, and so is `note_add` with `note = FALSE`, which leaves no note to add to.

`output` Output type. `"default"` (a printable `spicy_regression_table`); `"data.frame"` (the displayed table with its numbers); `"long"` (one row per coefficient, broom's names); `"tinytable"` / `"gt"` / `"flextable"` (rich-format tables); `"excel"` (writes to `excel_path`); `"clipboard"` (copies to system clipboard); `"word"` (writes flextable to `word_path`).

`excel_path` File path for `output = "excel"`, ending in `".xlsx"`, in a directory that exists. Default `NULL` (required when `output = "excel"`).

`excel_sheet` Sheet name when writing to Excel. `NULL` (the default) uses `"Regression"`.

`clipboard_delim`

Field delimiter for `output = "clipboard"`. Default `"\t"` (tab-separated, pastes cleanly into Excel / Google Sheets / Word). The clipboard payload mirrors the Excel layout (title row, spanner row, header, body, footer note) but is plain text – horizontal rules, cell merging, decimal alignment and the monospace font cannot be encoded in delimited text and are therefore absent from the paste. A cell holding the delimiter itself (a label with a comma under `clipboard_delim = ","`, or any number under `decimal_mark = ","`), a double quote or a line break is quoted RFC 4180-style, so the grid survives whatever delimiter you choose.

Paste behavior by target:

- **Excel / Google Sheets:** numerics are auto-detected and right-aligned; text cells stay left-aligned. (P-values such as `.005` get re-parsed as `0.005` by Excel's auto-format – to preserve the APA leading-zero-dropped display, prefer `output = "excel"`.)
- **Word:** the paste is converted to a Word table; all cells start left-aligned. Apply a Table Style (Insert > Table > Design) for APA-style borders, and set right-alignment on numeric columns (Layout > Align Right). For a self-contained Word file with borders and alignment pre-applied, use `output = "word"` instead.

`word_path` File path for `output = "word"`, ending in `".docx"`, in a directory that exists. Default `NULL` (required when `output = "word"`). The Word table inherits the flextable styling (Calibri font, APA borders, decimal-aligned numerics) and adds Word-specific features: an auto-numbered caption ("Table 1: ...", "Table 2: ...") via Word's `SEQ` field so multiple `table_regression()` calls in one document number consecutively; a re-printed header row on each page break; row split prevention so a single coefficient row never wraps across two pages; and an APA-styled note line (`*Note.*` italic prefix per APA Manual 7 §7.14).

R Markdown / Quarto: for embedded use, prefer `output = "flextable"` (returns the flextable object that knits to docx/HTML/PDF natively). `output =`

"word" writes a standalone .docx file, suited to scripted exports rather than chunk-level rendering.

word_template — Optional path to a custom .docx file used as the template for output = "word". The template's header, footer, page size, margins, and named styles ("Table Caption" in particular) are honored; the table is appended to the template body. Useful for institutional templates with pre-set headers ("APA Style", "Manuscript Submission Template", etc.). Default NULL (uses flextable's stock template).

Customizing the caption appearance: the table caption is tagged with the Word named style "Table Caption". The visual rendering (italic / bold / color / font) follows whatever that style is set to in the docx template. The stock Word template renders "Table Caption" in italic — the APA Manual 7 §7.10 condensed convention. For a different appearance (Nature-style bold non-italic, APA-strict 2-line bold-number / italic-title, etc.), edit the "Table Caption" style in a docx template and pass it via word_template = "your_template.docx". Style-based delegation keeps the rendered caption consistent with the surrounding document and lets editorial conventions (Nature, APA-strict, journal-specific) be applied without modifying the call site.

style — A journal style: a theme name ("jama", "nejm", "lancet", "annals", "apa", "aer"), a spicy_style() object, or NULL (the default). A style only changes DEFAULTS – any argument you pass explicitly wins over it. Set options(spicy.style =) for document-wide scope. A theme covers numeric formatting, the word that opens the note and the abbreviations the note defines, not full editorial conformity; ?spicy_style lists the exact rules each one encodes and the official document they come from. An unknown name is an error.

Value

A spicy_regression_table object (a data.frame subclass with classes c("spicy_regression_table", "spicy_table", "data.frame")) when output = "default". The result carries rendering attributes (title, note, align, padding) and provenance attributes (outcome, model_ids) consumed by the print method and the broom methods. For other output values, returns the format-specific object: gt_tbl, flextable, tinytable; for "data.frame" the displayed table as a data.frame (see as.data.frame.spicy_regression_table()); for "long" a tbl_df (model_id, outcome, term, estimate_type, estimate, std.error, conf.low, conf.high, statistic, df, p.value, test_type, nobs, r.squared, adj.r.squared, ...); the file path, invisibly, for "excel" and "word"; the table, invisibly, for "clipboard". The two data frames carry the note as printed in attr(x, "note").

Vocabulary tokens

Two vector arguments – show_columns and show_fit_stats – accept named tokens that select **what** to display and in **what order**. All tokens are lowercase (snake_case for compound tokens). Group tokens ("all_b", "all_ame", ...) expand to a fixed vector of atomic tokens; see show_columns below.

show_columns **– per-coefficient columns:**

Each token = one displayed column.

- Coefficient family: "b", "beta" (standardized), "se", "ci", "t", "p".

- Marginal effects: "ame", "ame_se", "ame_ci", "ame_p". "p" always refers to the B-coefficient p-value; for the AME-specific p-value use "ame_p".
- Partial effect sizes – lm only: "partial_f2", "partial_eta2", "partial_omega2", each with a paired _ci companion ("partial_f2_ci", ...).
- Partial effect size – glm and mixed-effects models: "partial_chi2", the term-level Type-II partial chi-square. Each term is tested under the marginality-respecting hypothesis that excludes any higher-order interaction containing it (the car::Anova(type = 2) convention), so the value does not depend on the factor coding. For glm the statistic is the likelihood-ratio chi-square (Long & Freese 2014 Sections 3.2.2, 3.2.4); for lmer / glmer / glmmTMB / nlme::lme it is the Wald chi-square built from the same Type-II hypothesis – a deliberate departure from the Type-III default of SAS PROC MIXED and lmerTest (the two conventions coincide for models without interactions). Rendered as value (df) to disambiguate factor terms (k-1 df) from numeric terms (1 df).
- Variance explained per fit – lm only: "r2", "adj_r2". Like "n", these are *per-fit* columns: they are populated by table_regression_uv() screens, where each predictor block is its own model and the R^2 answers "how much of the outcome does this predictor explain on its own". A model whose R^2 is a single model-level number drops the column and reports it in the fit-statistics rows instead (the "r2" / "adj_r2" tokens of show_fit_stats, on by default for lm). Refused for classes with no least-squares variance partition (glm, coxph, ...), where only competing pseudo-R^2 measures exist: name the one you want in show_fit_stats.
- Sample columns: "n" – per-row N, populated by table_regression_uv() screens (each predictor block is its own fit); models without per-row N data drop the column. "n_events" – outcome event counts as events/N, per factor level (reference row included) and model totals on continuous rows, computed on each model's own estimation sample (STROBE item 16's "data behind the association"; NEJM-style no. of events/total no.). Binary (binomial) outcomes and right-censored coxph fits – other models raise spicy_invalid_input.
- Survival estimands – coxph and survreg: "rmst", "rmst_se", "rmst_ci", "rmst_p" (restricted-mean- survival-time difference over [0, tau]) and "risk_diff", "risk_diff_se", "risk_diff_ci", "risk_diff_p" (cumulative-incidence difference at at_time), by g-computation from the fitted model with bootstrap inference. The horizon arguments are required; see tau / at_time.

Group tokens (presets) expand to a fixed atomic vector before validation:

- "all_b" -> c("b", "se", "ci", "p")
- "all_b_compact" -> c("b", "se", "p")
- "all_b_full" -> c("b", "se", "ci", "t", "p")
- "all_beta" -> c("b", "beta", "se", "ci", "p")
- "all_ame" -> c("ame", "ame_se", "ame_ci", "ame_p")
- "all_ame_compact" -> c("ame", "ame_p")
- "all_f2" / "all_eta2" / "all_omega2" -> partial_* + its _ci companion.

Mix groups and atomic tokens: show_columns = c("all_b", "ame", "ame_p"). Duplicates after expansion are deduplicated; the order of tokens controls the order of the displayed columns. If standardized != "none" and "beta" is not already requested, it is auto-injected after "b". Asking for "beta" while standardized = "none" raises spicy_invalid_input.

Default (show_columns = NULL) is context-aware: "all_b" for a single model (APA-7 Section 6.46 publication layout), "all_b_compact" for two or more models (CI dropped to fit the side-by-side layout; restore it explicitly when needed).

`show_fit_stats` **– model-level rows below the coefficients:**

- Counts: `"nobs"`, `"weighted_nobs"`, `"n_events"` (Cox models: number of events, reported alongside `n` per the field convention; blank for other classes).
- Variance explained (`lm` only): `"r2"`, `"adj_r2"`, `"omega2"`.
- Pseudo-R^2 (`glm` and ordinal `polr` / `clm`): `"pseudo_r2_mcfadden"` (McFadden 1974), `"pseudo_r2_nagelkerke"` (Nagelkerke 1991), `"pseudo_r2_tjur"` (Tjur 2009; binomial only).
- Residual scale: `"sigma"` (lm $\hat{\sigma}$ / glm dispersion), `"rmse"`.
- Survey designs (`survey::svyglm` only): `"eff_p"`, the effective number of parameters of the design (the first element of survey's `extractAIC`, the sum of the Rao-Scott eigenvalues). Opt-in and distinct from `"aic"`, which reports the design-based AIC of Lumley & Scott (2015). Blank for other classes.
- Bayesian fits only: `"r2_bayes"` (posterior-median Bayesian R^2, Gelman et al. 2019 – in the all-Bayesian default), `"elpd_loo"` and `"looic"` (PSIS-LOO expected log predictive density and its deviance-scale twin, Vehtari et al. 2017; opt-in, a few seconds per model; the footer discloses the elpd standard error), and `"waic"` (Watanabe-Akaike; PSIS-LOO is generally preferred).
- Negative-binomial dispersion (`MASS::glm.nb` only): `"theta"` ($V = \mu + \mu^2/\theta$) and `"alpha"` ($= 1/\theta$, the Stata `nbreg` convention). Refused for other families.
- Beta-regression precision (`betareg::betareg` only): `"phi"` ($Var(y) = \mu(1-\mu)/(1+\phi)$, Ferrari & Cribari-Neto 2004; higher ϕ = less dispersion). A constant precision is back-transformed from its link, so `y ~ x | 1` reports the same ϕ as `y ~ x`. Refused for other families and when the precision has covariates (`y ~ x | z`), so it is not a single number.
- GEE (`geepack::geeglm` only): `"qic"` and `"qicu"` – the quasi-likelihood information criteria (Pan 2001; QIC for working-correlation choice, QICu for covariate choice) via `geepack::QIC()`, computed only when requested (QIC silently refits the independence working model); `"scale"` – the estimated scale (dispersion) parameter, shown only when the fit estimated it (blank under `scale.fix = TRUE`, matching geepack's own summary); `"max_cluster_size"` – the largest cluster in the estimation sample (on by default with `"n_groups"`, which renders the `N (<id>)` cluster count). Cluster statistics count the clusters geepack itself used (`geese$clusz`, consecutive runs of `id =` values), so they always describe the displayed sandwich inference. Refused when no model in the table is a geeglm fit; the likelihood-based tokens (`"aic"`, pseudo-R^2, ...) are refused in return for all-GEE tables – quasi-likelihood has no likelihood.
- Absorbed fixed effects (`fixest` fits, and `estimatr` fits built with `fixed_effects =`): `"fixed_effects"` renders a `Fixed effects:` block at the top of the fit statistics – one Yes / No row per absorbed factor (the `etable` / `esttab` convention), blank cells for models without the concept in mixed tables – and is on by default for both. Varying-slope-only factors (`Origin[[x]]`) absorb no intercept: they read No when another model absorbs that factor, and contribute no row otherwise. `"within_r2"` is the FE-partialed within R-squared (a default for `feols`, opt-in for `estimatr`; GLM-family fixest fits report fixest's McFadden `pr2` instead). Both tokens are refused when no model in the table absorbs fixed effects.
- Effect size: `"f2"`.
- Information criteria: `"aic"`, `"aicc"`, `"bic"`, `"deviance"` (lowercase like every other token; the rendered row labels stay `AIC` / `AICc` / `BIC`).
- Change-stats for hierarchical comparison (active under `nested = TRUE`; see *Hierarchical comparison* below): `"r2_change"`, `"adj_r2_change"`, `"f_change"`, `"f2_change"`, `"lrt_change"`, `"aic_change"`, `"aicc_change"`, `"bic_change"`, `"deviance_change"`, `"p_change"`.

Default (resolved when NULL) is class-aware: lm and lm_robust fits get c("nobs", "r2", "adj_r2"); glm and ordinal polr / clm fits get c("nobs", "pseudo_r2_mcfadden", "pseudo_r2_nagelkerke", "aic"); mixed lm + glm sets union both groups (the renderer per-row en-dashes the inappropriate cell); Cox fits get c("nobs", "n_events", "aic"); GEE fits get c("nobs", "n_groups", "max_cluster_size") (the cluster structure, Stata xtgee-header style). When nested = TRUE, the class-aware default is extended with change tokens (c("r2_change", "f_change", "p_change") for lm, c("lrt_change", "p_change") for glm). The order of tokens in show_fit_stats controls the order of the rows.

Multi-model semantics

Pass a single fit or a list() of fits. Multi-model layout draws a centered **spanner label** above each model's sub-columns:

- list("Naive" = m1, "Adjusted" = m2) -> spanner labels "Naive" / "Adjusted". Partial naming (list("Naive" = m1, m2)) auto-fills missing slots as "Model <position>".
- list(m1, m2) (unnamed) -> if all response variables differ, each outcome becomes the spanner label of its model and the redundant Outcome body row is suppressed. The outcome is named as in the title: its name (from formula(fit)[[2]]), or its label attribute under labels = TRUE. If DVs match, the labels default to "Model 1, 2, ...".
- model_labels = c("A", "B") overrides everything.

A default label is worded in the language of the table (options(spicy.language)), over the columns and in the notes that cite the model; the column names of as.data.frame() and as_structured() keep the English one ("Model 1: B").

Duplicate explicit names in the list are rejected (spicy_invalid_input) – they would silently collide in the internal model_id key. So is a name that collides with the auto-filled label of another slot (list("Model 2" = m1, m2)): two models under one spanner cannot be told apart in the table, nor addressed by inline(). Rename the model, or pass model_labels.

Inference and standard errors

vcov selects the variance-covariance estimator:

- "classical" – OLS (lm) / Fisher information (glm).
- "HC0" to "HC5" – heteroskedasticity-consistent (via sandwich::vcovHC()).
- "CR0" to "CR3" – cluster-robust with Satterthwaite-corrected df (via clubSandwich::vcovCR()). Requires cluster.
- "bootstrap" – nonparametric or cluster bootstrap (boot_n replicates).
- "jackknife" – leave-one-out / leave-one-cluster-out.

For multi-model use, both vcov and cluster accept a single value (recycled to all models) or a list (one per model). The same fit can appear several times with different estimators to compare standard errors side-by-side.

Inferential regimes for lm / glm (B and AME share the same regime):

- classical, HC* -> t with df.residual (lm) / z-asymptotic (glm).

- bootstrap, jackknife -> z asymptotic.
- CR0-CR3 -> t with **Satterthwaite-corrected df** (B via clubSandwich::coef_test(); AME via clubSandwich::linear_contrast(); Pustejovsky & Tipton 2018). Under non-linear terms (poly(), I(), log(), splines::ns()), AME falls back to z-asymptotic with a spicy_fallback warning.

For other classes the t-vs-z axis follows the estimator's native reference distribution (e.g. glm, glmmTMB, survival, ordinal, betareg, mlogit use z; lm, lme, lmer, rms::ols use t).

Multinomial models: outcome categories as columns:

A single nnet::multinom model renders the publication layout: predictors as rows, one column group per **non-reference outcome category** (spanner = category name), so a predictor's effect can be compared across equations along its row. The mandatory footer note Reference outcome: <level>. names the base category (Stata's "base outcome" line); outcome_labels relabels the spanners (e.g. "Student vs Employed"). Fit statistics print once, under the first group. When per-category AMEs are requested the reference category appears as a last, AME-only group (its coefficient cells are empty – the reference has no equation; its AMEs complete the zero-sum across categories). The default show_columns compacts to B / SE / p exactly like multi-model tables; request atomic tokens to restore CIs. Multi-model and nested = TRUE multinomial tables keep the one-row-per-(category, predictor) layout – categories-within-models would need two spanner levels. tidy() and output = "long" always return the long form ("<category>: <term>" rows), whatever the display; as_structured() and output = "data.frame" mirror the displayed table (one column set per category), as they do for every other layout.

Robust SE availability by model class:

Not every estimator is defined for every class. A robust vcov the class cannot honor fails fast with spicy_unsupported_vcov – never a silent model-based result under a robust label:

lm, glm, MASS::glm.nb all of classical, HC*, CR*, bootstrap, jackknife. lm additionally takes "CR1S" – the full Stata regress, vce(cluster) convention: the CR1S small-sample scaling (equal to sandwich::vcovCL() with type = "HC1") with t(G - 1) inference, G the number of clusters. Use it to reproduce Stata tables; "CR2" (Bell-McCaffrey, Satterthwaite df) remains the recommended modern choice. "CR1S" is refused for glm with an explanation: Stata's ML commands use a different convention (G/(G-1) scaling, z), so the label would not match Stata output there.

mlogit classical + "CR1" only (cluster at the choice-situation level) – sandwich::vcovHC() mis-scales the sandwich for mlogit's per-choice-situation scores, so HC* is refused.

lmer, lme classical + CR* only – HC* and the resamplers (which refit lm/glm) are not defined for these.

coxph, rms (ols/lrm/cph/Glm) classical + "CR0" only: the Lin-Wei sandwich of coxph(cluster =) and rms::robcov() are one cluster estimator each, with no small-sample factor. "CR1" to "CR3" are refused, as are HC* and the resamplers.

survreg, mgcv::gam/bam, polr, clm, betareg, nnet::multinom, pscl::zeroinfl/hurdle classical + "CR1" only: sandwich::vcovCL() computes one cluster estimator for these, the CR0 sandwich scaled by $G/(G-1)$, G the number of clusters (**clubSandwich**'s CR1). "CR0", "CR2" and "CR3" are refused, as are HC* and the resamplers. clm with a scale / nominal (partial-PO) component is classical only. multinom needs **sandwich** >= 3.1-2 (which added its estfun() method); its cluster is one entry per observation. For the two-part pscl fits the cluster sandwich covers both components (count and zero) at once.

`quantreg::rq` its own estimator family, not the sandwich vocabulary: `classical` (= `"nid"`, quantreg's large-sample default), `"iid"`, `"ker"`, `"rank"` (intervals only) and a native `"bootstrap"`, clustered via the wild gradient bootstrap. `HC*`, `CR*` and `jackknife` are refused, each with its own reason – see the `vcov` argument.

`geepack::geeglm` no spicy-side estimator at all: GEE inference is robust by construction, so the fit's own sandwich (or jackknife) standard errors – chosen by geeglm's `std.err =` option, clustered on its `id =` – are the displayed inference. `HC*` / `CR*` and `cluster` are refused with a pointer to those fit options.

`survey::svyglm` `classical` only, on principle: the design-based Taylor / replicate variance is already the robust variance for the declared design, and clustering belongs in the design itself (`survey::svydesign(ids = )`), not in the table call.

`estimatr` **(**`lm_robust`/`iv_robust`**) and** `fixest` these fits carry the variance they were computed with – estimatr's `se_type =`, and whatever fixest's own `vcov` interface produced – and spicy never overwrites it. A `vcov` request is refused; set the estimator through the fitting package instead (`se_type =` for estimatr; `vcov =` at estimation or in `summary()` for fixest).

`glmmTMB` `classical` + `"CR0"` only, from glmmTMB's own sandwich (**glmmTMB** >= 1.1.15), which covers the coefficients, the zero-inflation and dispersion blocks and the AME. Without `cluster`, the cluster is the top-level grouping factor of the random effects. The estimator needs independent clusters that contain the random effects – every group within one cluster – so crossed random effects need a `cluster` that contains them all. It keeps the standard errors valid when the random-effects distribution or the conditional variance is misspecified, provided the estimates themselves stay consistent (Liang & Zeger 1986; Verbeke & Lesaffre 1997); with few clusters it understates the variance, and this class has no small-sample correction (Cameron & Miller 2015). `"CR1"`-`"CR3"` and `HC*` are refused (the sandwich has one score per cluster), as are REML fits and the ordinal family.

Other classes (`glmer`**,** `rstanarm`/`brms`**, ...)** `classical` (model-based) only (**clubSandwich** has no working backend for `glmer`).

For the mixed-effects fits (`lmer`, `lme`, `glmmTMB`), the robust estimator covers the fixed effects: their coefficients, their joint tests (`"partial_chi2"`), the standardized coefficients and the AME. The random-effect variance components keep their model-based inference.

Cluster-robust backends differ by class but are each cross-validated to the field-standard oracle: `lm`/`glm`/`lmer`/`lme` use **clubSandwich** (CR2 = Bell-McCaffrey, with Satterthwaite df for `lm`/`lme`/`lmer`); `coxph`/`cph` use the Lin-Wei grouped-dfbeta sandwich (identical to `coxph(..., cluster=)`); `survreg`/`gam`/`polr`/`clm`/`betar` and the `pscl` two-part fits use `sandwich::vcovCL()`. For `gam`/`bam`, spicy forms the sandwich itself, from the GLM score and the penalized covariance of the fit, and lets `vcovCL()` do only the clustering arithmetic: the sandwich methods for these fits are wrong for a free dispersion, a non-canonical link and a non-Gaussian `bam()`. Extended families other than `nb()` (`betar`, `ocat`, `scat`, `tw`, ...) have another score and are refused. For all of these `vcovCL()` classes, the one estimator is CR1, requested as `"CR1"`; the other `CR*` tokens are refused rather than mapped. `rms` fits use `rms::robcov()` (which needs the fit's `x = TRUE, y = TRUE`). The Lin-Wei and `robcov()` sandwiches have no small-sample factor: their one estimator is requested as `"CR0"`, and the other `CR*` tokens are refused rather than mapped. `glmmTMB` fits use `glmmTMB::sandwich()`, with one score per cluster (the gradient of the cluster's marginal log-likelihood) and no finite-sample factor; its one estimator is requested as `"CR0"`, and the other `CR*` tokens are refused rather than mapped. `cluster` length is one entry per observation, except `mlogit` (one per choice situation) and censored `coxph` (one per subject).

How to specify `cluster`**:**

Three accepted forms, in order of preference:

1. **Formula** – `cluster = ~region` (or `cluster = ~region:year` for the interaction of two variables). The variables are looked up in `model.frame(fit)` first, then in the original `data` argument captured by the fit. **Recommended**: independent of the dataset's name, composable for multi-way clustering, consistent with `sandwich::vcovCL()` / `clubSandwich::vcovCR()`.
2. **String** – `cluster = "region"`. A single column name resolved the same way as the formula. Convenient but cannot express interactions.
3. **Vector** – `cluster = df$region`. An atomic vector of length `nobs(fit)`. Use this when the cluster key is **derived on the fly** (`cluster = interaction(df$region, df$year)`, `cluster = as.integer(format(df$date, "%Y"))`), comes from a **different dataset** with matching row order, or is otherwise not a column of the model's `data`.

A bare unquoted name (`cluster = region`) is **not** column selection: it is evaluated as an ordinary R variable. If an object `region` exists in the calling environment, its value is used as the **vector** form above; if it does not (the typical "unquoted column name" intent), the call fails with a migration error pointing at `~region` / `"region"`. Bare-name column selection is deliberately unsupported – it would require non-standard evaluation magic that breaks under programmatic use (function wrapping, dynamic column choice, loops).

For multi-model use, mix forms freely: `cluster = list(~region, "region", df$region)`.

Bayesian fits (rstanarm / brms)

`stanreg` and `brmsfit` models are summarized from their posterior draws: `B` is the posterior median, `SE` the posterior MAD SD (the scaled median absolute deviation – the pairing of *Regression and Other Stories* and rstanarm's own print; equal to the posterior SD for a normal posterior), and the interval an equal-tailed credible interval (header `95% CrI`; `ci_method = "hdi"` opts into the highest-density interval, header `95% HDI`). There is no p-value or t-statistic; group presets (`"all_b"`, ...) expand without them, and the opt-in `"pd"` column reports the probability of direction (Makowski et al. 2019), with a footer definition. Under `exponentiate = TRUE` all quantities are computed from the exponentiated draws directly (the SE is the posterior MAD SD on the ratio scale, not a delta-method approximation). The AME columns are equally draws-native: `avg_slopes()` computes the marginal effects per posterior draw, and the table reports their posterior median, MAD SD and credible interval (equal-tailed or HDI per `ci_method`); `"ame_p"` has nothing to fill and follows the p-column policy (dropped from presets, refused as an atomic token, dashed in mixed tables). Standardized betas follow the same draws logic: `"posthoc"` / `"basic"` / `"smart"` are exact affine rescales of the link-scale summaries (the Gaussian family divides by SD(y); every other family standardizes predictors only, the frequentist glm convention); `"refit"` and `"pseudo"` are refused.

Fit statistics: `"r2_bayes"` (in the Bayesian default) plus the opt-in `"elpd_loo"` / `"looic"` / `"waic"`, whose standard errors are disclosed in the footer; unreliable estimates (PSIS-LOO Pareto k above the sample-size-specific threshold $\min(1 - 1/\log_{10} S, 0.7)$ of Vehtari et al. 2024 – the same bound `loo::loo()` prints – and WAIC p_waic > 0.4) add a footer caveat instead of being silenced. Every table is backed by an automatic sampler-diagnostics guard (R-hat >= 1.01, ESS below 100 per chain – floored at 400 so fewer chains never weaken the bar –, divergent transitions, E-BFMI < 0.2, per Vehtari et al. 2021): problems add a footer line and raise a warning classed `spicy_bayes_diagnostics` (nested under `spicy_caveat`, so `withCallingHandlers(spicy_bayes_diagnostics = ...)` mutes the guard selectively); clean fits print nothing. Per-coefficient `"rhat"` / `"ess_bulk"` / `"ess_tail"` columns are available in all-Bayesian tables, as is `"mcse"` – the Monte Carlo standard error of the displayed posterior median, the criterion for how many digits a table can honestly show

(a displayed digit is Monte-Carlo stable when twice the MCSE stays below it; Gelman, Vehtari, McElreath et al. 2026, sec. 11.6). Under `exponentiate = TRUE` the MCSE is recomputed on the exponentiated draws.

Refused on principle (classed error, never a silent fallback): likelihood-based fit statistics (`"aic"`, pseudo-R^2, ...), `p_adjust`, robust / cluster `vcov` (model the clustering with group-level terms instead), `ci_method = "profile"` / `"boot_percentile"`, and non-MCMC fits (`algorithm = "meanfield"` / `"optimizing"` – refit with `algorithm = "sampling"`). See the Bayesian regression tables article.

Hierarchical (nested) model comparison

`nested = TRUE` adds **per-pair change statistics as in-table rows** (APA Table 7.13 / Stata `esttab` / SPSS Model Summary convention). Each adjacent pair (M2 vs M1, M3 vs M2, ...) contributes one column of change stats; the FIRST model column gets en-dashes (no previous model to compare to). Validation requires identical `nobs` and identical response variable across all models.

Structural refusals. Four hierarchies carry no valid change statistic whatever their `nobs` and response, and are refused with `spicy_invalid_input` rather than rendered with empty or meaningless cells:

- *A REML pair whose fixed effects differ* (`nlme::lme(method = "REML")`, `glmmTMB(REML = TRUE)`). The restricted likelihood depends on the fixed-effects design matrix, so the two criteria are not on a common scale and the likelihood-ratio test is not valid (Pinheiro & Bates 2000 Section 2.4.2). Refit both by maximum likelihood (`method = "ML"`, `REML = FALSE`). A REML pair that differs only in its random structure stays valid and is accepted.
- *Two* `fixest` *fits that absorbed different fixed effects*. fixest counts every absorbed level as an estimated parameter, so a test across two `| fe` structures measures the change in the absorbed effects together with the change in the coefficients, and the two cannot be separated. Hold the `| fe` part fixed across the hierarchy.
- *Mixed-effects fits from different engines* (`lme4`, `glmmTMB`, `nlme::lme`). The change statistics come from the engine's own two-model `anova()`, and no engine's method accepts a fit produced by another. Refit one of the two with the other's engine.
- *A* `MASS::rlm` *hierarchy*. M-estimation minimizes a bounded loss on the scaled residuals: it partitions no sums of squares and it is not a likelihood, so there is neither a likelihood-ratio test nor a partial F to report. Every change token is refused – including on a single fit or a `nested = FALSE` pair, where the token is the explicit request. Compare the least-squares fits (`lm()`) if the hierarchy itself is the question.

In the first three, `nested = FALSE` renders the models side by side instead.

Default change tokens auto-injected when `show_fit_stats` is `NULL`:

- All-lm (and `nls`): `c("r2_change", "f_change", "p_change")` – APA hierarchical regression standard.
- All-glm: `c("lrt_change", "p_change")` – Hosmer & Lemeshow Section 3.5; Long & Freese 2014 Section 3.2.4.
- Any other hierarchy that carries a likelihood (`survreg`, `polr`, `clm`, `coxph`, `gls`, `betareg`, `multinom`, `fixest`, ...): the same `c("lrt_change", "p_change")`.
- Mixed-effects (`lmer` / `glmer` / `glmmTMB` / `nlme::lme`): `c("aic_change", "bic_change", "lrt_change", "p_change")` – Pinheiro & Bates 2000 Section 2.4.1.

- Quantile (rq): c("f_change", "p_change") – the Wald-type test quantreg::anova.rq() reports.

To customize, pass the change tokens directly to show_fit_stats. The variance-explained change tokens ("r2_change", "adj_r2_change", "f_change", "f2_change") raise spicy_invalid_input on any hierarchy whose nested comparison is a likelihood-ratio test – glm, and equally every other likelihood class and the mixed-effects families: that comparison reports a chi-square, not a variance-explained change, and the message points at "lrt_change" + "p_change". A quantile hierarchy refuses them too, and "lrt_change" with them, pointing at "f_change" + "p_change" instead. lm and nls keep the least-squares tokens.

The change test follows vcov. Under the classical vcov the change test is the partial F, the likelihood-ratio test or quantreg::anova.rq()'s Wald test, as above. Under any other vcov, and under the classical token when the fit carries a robust variance of its own, it is the Wald test of the coefficients the current model adds, on the matrix its coefficient rows were computed from: $W = b'V^{-1}b$, with b the q added coefficients (aliased ones left out) and V their block of the matrix. The test takes the regime of the current model's coefficient tests:

- *F*, in a row labeled "Wald F-change": W/q on (q, df) degrees of freedom, the convention of lmtest::waldtest(test = "F") and car::linearHypothesis(). It applies to lm under "HC0"-"HC5" (and "CR1S", on $G - 1$ degrees of freedom), rms::ols under "CR0" or after rms::robcov(), quantreg::rq under "iid" and "ker", and a fixest::feols fit estimated with a non-iid vcov (on fixest's own degrees of freedom, as fixest::wald()). Under "CR0"-"CR3", lm, glm (MASS::glm.nb included), lme4::lmer and nlme::lme use the HTZ test of clubSandwich::Wald_test(), the small-sample correction of their coefficient tests.
- *Chi-square*, in a row labeled "Wald χ^2 (change)" that replaces the likelihood-ratio row: W on q degrees of freedom, the convention of lmtest::waldtest(test = "Chisq"). It applies to glm and glm.nb under "HC0"-"HC5", lm and glm under "bootstrap" and "jackknife", rq under "bootstrap", under "CR0" to coxph and rms::lrm / cph / Glm, and under "CR1" to survreg, mgcv::gam and bam, polr, clm, betareg, multinom, mlogit, zeroinfl and hurdle; also to survreg(robust = TRUE), a fixest::fepois or feglm fit with a non-iid vcov, and an rms fit of these classes passed through robcov().

ΔR^2, the information criteria and the deviance change do not depend on the vcov and are unchanged, and a vcov list tests each pair with the current model's estimator. The previous model's coefficients must all appear, under the same names, in the current one, and a smooth term added to a gam under a robust vcov is refused (spicy_invalid_input in both cases). rq keeps anova.rq()'s test under "nid", its default, and under "rank", which yields no matrix. A Wald test of many constraints in a small sample is liberal: on 219 observations, a block of 19 predictors gave F(19, 178) = 6.91 classically and 9.06 as a Wald HC3 test. The HTZ correction under "CR0"-"CR3" mitigates it.

Standardized coefficients

standardized controls the method when "beta" is in show_columns:

- "refit" – refit on z-scored data. For lm both X and Y are z-scored (Cohen et al. 2003 gold standard); for glm only numeric X (Long & Freese 2014 Section 4.7.2 "x-standardization").
- "posthoc" – post-hoc scaling. lm: $\beta = B \times SD(X)/SD(Y)$; glm: X-only $\beta = B \times SD(X)$ (Y is undefined on the link scale).
- "basic" – like "posthoc" but factor dummies are scaled by their column SD.

- `"smart"` – Gelman (2008): continuous numeric inputs are scaled by `2 * SD(X)` (a +/-1 SD swing then spans the same range as a binary's 0 to 1 step); binary inputs – numeric 0/1 and factor dummies – are left unscaled. (Before 0.13.0 the rule was applied inverted; see NEWS.)
- `"pseudo"` – *glm only*. Menard (2004, 2011) fully-standardized $\beta = B \times SD(X)/SD(Y^*)$, with Y^* the latent variable on the link scale and $SD(Y^*) = \sqrt{Var(\hat{\eta}) + Var_{link}}$ ($\pi^2/3$ logit, 1 probit, $\pi^2/6$ cloglog). Binomial families only; non-binomial returns NA with a `spicy_caveat`.
- `"none"` (default) – no β computed.

Interactions and transforms. Under `"refit"`, an interaction's β is the coefficient of the product of the z-scored components – the recommended treatment for such models (Cohen et al. 2003 Section 7.7; Aiken & West 1991; Friedrich 1982). Under `"posthoc"`, `"basic"`, and `"smart"`, the product / transformed design column is treated as a single numeric column and scaled by its own SD (under `"smart"`, by `2 * SD` when the product column is continuous; a binary product column – e.g. a binary-by-binary interaction – stays unscaled like any binary input) – the convention of SPSS beta, Stata `regress, beta`, SAS PROC REG `STB`, and `lm.beta::lm.beta()`, identical to `effectsize::standardize_parameters(method = "basic")` on those columns. The two conventions differ whenever the components are correlated; a `spicy_caveat` warns and the footer names the convention actually used. `"refit"` declines formulas with inline transforms (`log(x)`, `poly()`, `factor()` written in the formula): the model frame's evaluated columns cannot be re-evaluated on z-scored data, so spicy falls back to `"posthoc"` with a warning (`effectsize`'s refit instead standardizes *before* the transform – a different estimand). Pre-build transformed columns in `data` for an exact refit.

Multiple-comparison adjustment

Adjusting the p-values of all coefficients of a single regression model is **not** the standard convention. Each coefficient tests a distinct hypothesis on a distinct predictor – not the situation multiple-testing procedures were designed for (Rothman 1990; Greenland 2017; APA Manual 7 Section 6.46; Harrell *Regression Modeling Strategies* Section 5.4; Gelman, Hill & Yajima 2012). Hence the default `p_adjust = "none"`.

Adjustment is appropriate for: mass screening with no prior hypothesis (typically `"BH"` / FDR), pre-registered multi-endpoint confirmatory designs (typically `"holm"`), or when a journal / SAP explicitly requests it.

The adjustment runs **before** any `keep` / `drop` filtering, so the family is the model's full coefficient set (intercept and reference rows excluded), not the displayed subset – filtering is a display choice and must not change the inferential family.

Tied event times and the survival estimands

A Cox model has to decide what to do when several events share the same time, and `survival::coxph()` decides with `ties = "efron"` by default. The absolute estimands do not make a second decision:

The baseline hazard behind the RMST-difference and risk-difference columns follows the tie-handling convention of the fit, exactly as `survival::survfit()` and `survival::basehaz()` do. A `ties = "breslow"` fit gives a Breslow baseline; the default Efron fit gives an Efron baseline. One rule, so the hazard-ratio column and the dRMST column always come from the same likelihood. To change the convention, change the fit.

When the choice matters is not simply "when there are many ties". What governs it is the fraction of tied events *within the risk set*, together with the effect size and the spread of the covariates: a large risk set dilutes a tied block, a small one – fine strata, matched or nested case-control designs – does not. A file with hundreds of tied days can be insensitive to the convention while a small stratified one with a handful of ties is not.

And Efron is the better default, not the correct answer: the approximations are all biased at finite sample size and agree asymptotically, and Efron's virtue is a much smaller small-sample bias (Hertz-Picciotto & Rockhill 1997). Efron himself, quoting Peto, put it as "it probably doesn't make much difference" (Efron 1977).

Output formats and broom integration

`output` selects the return type:

- `"default"` – a `spicy_regression_table` (`data.frame` subclass) printed via `spicy_print_table()`.
- `"data.frame"` – the displayed table with its numbers, as.data.frame() of the default object: one row per printed row, the key columns `variable`, `label`, `level` and `.row_role`, then the printed columns, numeric, each interval split into its two bounds.
- `"long"` – the long-format tibble, one row per `(model_id, term, estimate_type)`, with broom's column names.
- `"tinytable"` / `"gt"` / `"flextable"` – rich-format HTML / Word / PDF tables (require the corresponding Suggests package). In a PDF, a `"flextable"` table wider than the page can be fitted with `flextable::fit_to_width()`.
- `"excel"` – writes to `excel_path` via `openxlsx2::write_xlsx()` and returns the path invisibly.
- `"word"` – writes to `word_path` via `flextable::save_as_docx()` and returns the path invisibly.
- `"clipboard"` – copies to the system clipboard via `clipr::write_clip()`.

`broom::tidy()` returns a long tibble with one row per `(model_id, term, estimate_type)` and broom-canonical column names (`estimate`, `std.error`, `conf.low`, `conf.high`, `statistic`, `p.value`). `broom::glance()` returns one row per model with the model-level statistics; `df.residual` is kept numeric so cluster-robust Satterthwaite df is preserved.

Global options

- `options(spicy.style = )` – the journal style applied by default to all nine table functions (`table_categorical()`, `table_categorical_svy()`, `table_continuous()`, `table_continuous_svy()`, `table_continuous_lm()`, `table_outcome()`, `table_multiresponse()`, `table_regression()` and `table_regression_uv()`). Takes a theme name (`"jama"`, `"nejm"`, `"lancet"`, `"annals"`, `"apa"`, `"aer"`) or a `spicy_style()` object. The style of a report is a property of the document, so it is set once in the setup chunk; the `style` argument overrides it per call, and any formatting argument you type overrides both. `options(spicy.style = NULL)` restores spicy's own defaults. See `spicy_style()` for the exact rules each theme encodes and the official document they come from.
- `options(spicy.note_style = )` – how the table note is rendered by the `"tinytable"` engine (all nine table functions). A note is subordinate to the table it documents, so it is set one size

down, in black: `0.9em` (`\small` in LaTeX), matching the `"gt"` engine and the 10pt-on-11pt of the Word engine. Two opt-ins:

- `"none"` – no intervention; the note is rendered exactly as the receiving document template styles it.
- any other string – extra arguments for the Typst `text()` call around the note, appended to the size (e.g. `options(spicy.note_style = "fill: luma(89)")` for a gray note). Typst only; the HTML note keeps the plain `0.9em`.

`options(spicy.note_style = NULL)` restores the default.

Weights

No `weights` argument: weights are a property of the fit (extracted via `stats::weights()`). Pass them when fitting: `lm(y ~ x, data = df, weights = w)`. All downstream computations (vcov, AME, standardization, `weighted_nobs`) extract them automatically.

Internationalization

Output follows `options(spicy.language)` (see `spicy_labels()`); the column names of `as.data.frame()` and `as_structured()` stay English in every language. Override user-facing strings via `reference_label`, `model_labels`, `outcome_labels`, and `labels`. The title and footer are post-processable via `attr(result, "title")` and `attr(result, "note")`.

Classed conditions

Every error and warning emitted by `table_regression()` carries a classed condition for programmatic dispatch via `tryCatch()` or `withCallingHandlers()`. Errors inherit from `spicy_error` (root); warnings from `spicy_warning`. Specific leaves used by this function include `spicy_invalid_input`, `spicy_invalid_data`, `spicy_unsupported`, `spicy_unsupported_vcov`, `spicy_unsupported_standardized`, `spicy_missing_pkg`, `spicy_missing_column`, `spicy_ignored_arg`, `spicy_caveat`, `spicy_fallback`. See `spicy` for the full taxonomy.

References

APA Manual 7 (American Psychological Association, 2020), Tables 7.13-7.15.

Aiken, L.S. & West, S.G. (1991). *Multiple regression: Testing and interpreting interactions*.

Cameron, A.C. & Miller, D.L. (2015). A practitioner's guide to cluster-robust inference. *Journal of Human Resources*, 50(2), 317-372.

Cohen, J., Cohen, P., West, S.G., & Aiken, L.S. (2003). *Applied multiple regression / correlation analysis for the behavioral sciences* (3rd ed.). Lawrence Erlbaum.

Davison, A.C. & Hinkley, D.V. (1997). *Bootstrap methods and their application*. Cambridge University Press.

Efron, B. (1977). The efficiency of Cox's likelihood function for censored data. *Journal of the American Statistical Association*, 72(359), 557-565.

Friedrich, R.J. (1982). In defense of multiplicative terms in multiple regression equations. *American Journal of Political Science*, 26(4), 797-833.

Hertz-Picciotto, I. & Rockhill, B. (1997). Validity and efficiency of approximation methods for tied survival times in Cox regression. *Biometrics*, 53(3), 1151-1156.

Liang, K.-Y. & Zeger, S.L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika*, 73(1), 13-22.

Pustejovsky, J.E. & Tipton, E. (2018). Small-sample methods for cluster-robust variance estimation and hypothesis testing in fixed effects models. *Journal of Business & Economic Statistics*, 36(4), 672-683.

Verbeke, G. & Lesaffre, E. (1997). The effect of misspecifying the random-effects distribution in linear mixed models for longitudinal data. *Computational Statistics & Data Analysis*, 23(4), 541-556.

Wasserstein, R.L., Schirm, A.L., & Lazar, N.A. (2019). Moving to a world beyond "p < 0.05". *The American Statistician*, 73(sup1), 1-19.

See Also

`table_regression_models()` for the registry of supported model classes and the per-family behavior reference (also reachable as `?table_regression_mixed`, `?table_regression_ordinal`, ...). If a class is not listed there, try `table_regression(fit)` anyway – unsupported classes error with a clear message. Other regression-table functions: `table_continuous_lm()` for one-predictor-by-many-outcomes descriptive tables. Other spicy table functions: `freq()`, `cross_tab()`, `table_categorical()`, `table_continuous()`. Underlying machinery: `spicy_print_table()` for ASCII rendering; `build_ascii_table()` for the low-level renderer. Inferential infrastructure (internal): `compute_model_vcov()`, `compute_coef_inference()`, `compute_wald_test()`. broom integration: `broom::tidy()`, `broom::glance()`.

Examples

```
# ---- Single-model usage ------------------------------------------
fit <- lm(wellbeing_score ~ age + sex + smoking, data = sochealth)

# Default APA layout: B / SE / 95% CI / p plus the n / R^2 /
# Adj. R^2 fit-stats footer. Factor reference level is annotated
# with `(ref.)` and shows an en dash in the statistic columns.
table_regression(fit)


# Standardized coefficients (beta) injected next to B. "refit"
# is the Cohen et al. (2003) refit-on-z-scores convention;
# "basic" reproduces the SPSS / Stata regress, beta definition.
table_regression(fit, standardized = "refit")



# Custom column set: B + AME + AME-specific p-value. Note that
# the `p` token always belongs to B, never to AME -- use the
# explicit `ame_p` token for AME inference.
if (requireNamespace("marginaleffects", quietly = TRUE)) {
  table_regression(
    fit,
    show_columns = c("b", "p", "ame", "ame_ci", "ame_p")
  )
}
```

```
# Group-token shortcut: "all_b" + "all_ame" expands to the full
# B / AME column families side by side.
if (requireNamespace("marginaleffects", quietly = TRUE)) {
  table_regression(fit, show_columns = c("all_b", "all_ame"))
}


# ---- Cluster-robust variance ------------------------------------
# CR2 (Bell-McCaffrey) with Satterthwaite-corrected df is the
# recommended default under few clusters. Three forms are accepted
# for `cluster`; the formula is preferred for composability with
# multi-way clustering and for programmatic robustness.

if (requireNamespace("clubSandwich", quietly = TRUE)) {
  table_regression(fit, vcov = "CR2", cluster = ~region)
}
if (requireNamespace("clubSandwich", quietly = TRUE)) {
  table_regression(fit, vcov = "CR2", cluster = "region")
}
if (requireNamespace("clubSandwich", quietly = TRUE)) {
  table_regression(fit, vcov = "CR2", cluster = ~region:age_group)
}



# ---- Hierarchical (nested) regression ---------------------------
# Adds in-table change-statistic rows (Delta R^2 / F-change /
# p-change for lm; LRT / p-change for glm) below the fit-stats.
# Note: hierarchical comparison requires identical observations
# across all models -- prepare a complete-case subset first so
# R's listwise deletion does not produce different `nobs` per
# model (which the function rejects).
sochealth_cc <- na.omit(
  sochealth[, c("wellbeing_score", "age", "sex", "smoking")]
)
m1 <- lm(wellbeing_score ~ age,                  data = sochealth_cc)
m2 <- lm(wellbeing_score ~ age + sex,            data = sochealth_cc)
m3 <- lm(wellbeing_score ~ age + sex + smoking,  data = sochealth_cc)
table_regression(
  list("Step 1" = m1, "Step 2" = m2, "Step 3" = m3),
  nested = TRUE
)



# ---- Side-by-side variance comparison ---------------------------
# Same fit, three vcovs in one wide table. Useful for showing the
# sensitivity of inference to the variance assumption.
if (requireNamespace("clubSandwich", quietly = TRUE)) {
  table_regression(
```

```
    list("Classical" = fit, "HC3" = fit, "CR2" = fit),
    vcov    = list("classical", "HC3", "CR2"),
    cluster = list(NULL, NULL, ~region)
  )
}


# ---- Tidy long format for downstream pipelines -------------------
if (requireNamespace("broom", quietly = TRUE)) {
  broom::tidy(table_regression(fit))
}


# ---- Mixed-effects models ----------------------------------------
# Linear mixed-effects (lme4). The footer adds a random-effects
# panel with sigma + Wald SE / CI from `merDeriv`, the Nakagawa
# marginal / conditional R^2 fit-stats, and a per-class p-value
# annotation line.
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- lme4::lmer(Reaction ~ Days + (Days | Subject),
                    data = lme4::sleepstudy)
  table_regression(fit)

  # Switch to the variance scale (sigma^2 instead of sigma).
  table_regression(fit, re_scale = "variance")

  # Minimal random-effects display: estimates only, no SE / CI.
  table_regression(fit, re_columns = "est")

  # Suppress the random-effects panel entirely.
  table_regression(fit, show_re = FALSE)
}

# Hierarchical mixed-effects comparison (nested LRT).
if (requireNamespace("lme4", quietly = TRUE)) {
  m1 <- lme4::lmer(Reaction ~ 1     + (1 | Subject),
                   data = lme4::sleepstudy, REML = FALSE)
  m2 <- lme4::lmer(Reaction ~ Days  + (1 | Subject),
                   data = lme4::sleepstudy, REML = FALSE)
  m3 <- lme4::lmer(Reaction ~ Days  + (Days | Subject),
                   data = lme4::sleepstudy, REML = FALSE)
  table_regression(list(m1, m2, m3), nested = TRUE)
}


## Not run:
# ---- Rich-format outputs (require optional Suggests packages) ----
table_regression(fit, output = "gt")
table_regression(fit, output = "flextable")
table_regression(fit, output = "tinytable")

# ---- File outputs ------------------------------------------------
table_regression(fit, output = "excel",
```

```
                  excel_path = tempfile(fileext = ".xlsx"))
table_regression(fit, output = "word",
                  word_path  = tempfile(fileext = ".docx"))

# ---- System clipboard (interactive use) --------------------------
table_regression(fit, output = "clipboard")

## End(Not run)
```

`table_regression_models` *Supported models and per-family behavior of table_regression()*

Description

`table_regression_models()` returns the registry of model classes supported by `table_regression()`, one row per engine, with each class's family, average-marginal-effects estimand, `exponentiate` semantics, and labeled table blocks. The same registry drives this page's table, so the published list cannot drift from the code.

This page is also the reference for **per-family behavior** (the sections below). It is reachable as `?table_regression_models`, `?table_regression_mixed`, `?table_regression_ordinal`, `?table_regression_counts`, `?table_regression_categorical`, `?table_regression_survival`, `?table_regression_robust`, or `?table_regression_bayesian`.

If a class is not listed: fit the model and call `table_regression(fit)` anyway – unsupported classes error with a clear message naming the supported set. Feature requests are welcome on the issue tracker.

Usage

```
table_regression_models()
```

Value

A data frame with one row per supported engine and columns `family`, `class`, `engine`, `ame`, `exponentiate`, `blocks`.

Supported classes

Family	Class	Engine
Linear and generalized linear	`lm`	`stats::lm()`
Linear and generalized linear	`glm`	`stats::glm()`
Linear and generalized linear	`negbin`	`MASS::glm.nb()`
Linear and generalized linear	`rlm`	`MASS::rlm()`
Linear and generalized linear	`nls`	`stats::nls()`
Robust, IV, quantile, panel	`lm_robust`	`estimatr::lm_robust()`

Robust, IV, quantile, panel	iv_robust	estimatr::iv_robust()
Robust, IV, quantile, panel	ivreg	AER::ivreg()
Robust, IV, quantile, panel	tobit	AER::tobit()
Robust, IV, quantile, panel	rq	quantreg::rq()
Robust, IV, quantile, panel	fixest	fixest::feols(), fixest::feglm(), fixest::fepois(), fixest::fen
Mixed effects	lmerMod	lme4::lmer()
Mixed effects	glmerMod	lme4::glmer()
Mixed effects	glmmTMB	glmmTMB::glmmTMB()
Mixed effects	lme	nlme::lme()
Mixed effects	gls	nlme::gls()
Population-averaged (GEE)	geeglm	geepack::geeglm()
Ordinal	polr	MASS::polr()
Ordinal	clm	ordinal::clm()
Ordinal	clmm	ordinal::clmm()
Categorical	multinom	nnet::multinom()
Categorical	mlogit	mlogit::mlogit()
Counts, two-part	zeroinfl	pscl::zeroinfl()
Counts, two-part	hurdle	pscl::hurdle()
Survival	coxph	survival::coxph()
Survival	survreg	survival::survreg()
Survival	cph	rms::cph()
Survival	flexsurvreg	flexsurv::flexsurvreg()
Survey-weighted	svyglm	survey::svyglm()
Survey-weighted	svyolr	survey::svyolr()
Survey-weighted	svycoxph	survey::svycoxph()
Additive, proportions, selection	gam	mgcv::gam(), mgcv::bam()
Additive, proportions, selection	betareg	betareg::betareg()
Additive, proportions, selection	selection	sampleSelection::selection()
rms	ols	rms::ols()
rms	lrm	rms::lrm()
rms	Glm	rms::Glm()
Bayesian	stanreg	rstanarm::stan_glm(), rstanarm::stan_glmer()
Bayesian	brmsfit	brms::brm()

Shared semantics (all classes)

- A robust vcov request is honored through the class's field-standard backend, or **refused with a clear error** naming the supported set; the footer always names the estimator actually applied.
- exponentiate = TRUE is link-gated: it produces a labeled ratio (OR / IRR / HR / RR / MR / TR) only where the link warrants one. Identity-link fits warn and are left untouched; non-ratio links (probit, cauchit, inverse, ...) are **refused with a clear error**.
- Class-specific structure renders as labeled subordinate blocks of rows in the same table, each explained by a footer line.
- Fit statistics default to the family's field standard (show_fit_stats overrides; class-inappropriate tokens are rejected with a pointer to the right ones).

- Everything is available programmatically: `broom::tidy()`, `glance()`, `as_structured()`, `as.data.frame()`.

Mixed effects

Fixed effects: Satterthwaite t (`lmer` + **lmerTest**), Wald z (`glmer`, `glmmTMB`), containment-df t (`lme`). Random effects render as a `Random effects` block of rows (SD / correlation / residual with SE and CI; `re_scale`, `re_columns`), deliberately with **no per-row p-value** (boundary-invalid Wald; Self & Liang 1987) – the footer carries the chi-bar-squared LR test of the whole random part, and `re_test = "lrt" / "rlrt"` adds an opt-in boundary-correct per-term test. `N (groups)` and `ICC` are fit-stat rows; Nakagawa marginal / conditional R-squared are the default R-squared family. `CR*` cluster-robust standard errors are available for `lmer` and `lme` fits, via **clubSandwich** with Satterthwaite degrees of freedom. `glmmTMB` fits take `"CR0"` from glmmTMB's own sandwich (Wald z), clustered by default on the top-level grouping factor; see the robust-SE section of `table_regression()` for when it holds. `glmer` fits keep their model-based standard errors: a `CR*` request on them is refused with a clear error.

Population-averaged (GEE) models

`geepack::geeglm()` fits are read on their own terms: the sandwich standard errors the fit computed (its `std.err =` option, clustered on its `id =`) are the displayed inference – GEE is robust by construction, so spicy's `vcov` / `cluster` arguments are refused with a pointer to the fit options. Coefficients are population-averaged (marginal) effects; the footer discloses the working correlation structure with its estimated alpha. Wald z inference; `exponentiate` follows the usual link gates (OR / IRR / RR). Default fit statistics report the cluster structure (n, `N (<id>)`, largest cluster); the quasi-likelihood information criteria `"qic"` / `"qicu"` (Pan 2001) and the `"scale"` (dispersion) parameter are opt-in – there is no likelihood, so AIC, pseudo-R-squared, `nested = TRUE`, and `standardized` are refused. See the population-averaged section of the Mixed-effects regression tables article for the contrast with subject-specific mixed models.

Ordinal models

Cut-points render as a `Thresholds` block (log-odds scale, never exponentiated; `show_thresholds`). Partial-proportional-odds `clm` terms render as a `Non-proportional effects` block, one coefficient per cut-point. `exponentiate` yields proportional odds ratios under logit; `ci_method = "profile"` profiles the predictor coefficients. AME is per-category (the marginal effect on each P(Y = k)). Defaults include McFadden and Nagelkerke pseudo-R-squared. See the Ordinal regression tables article.

Cumulative-link mixed models (`ordinal::clmm()`, and `glmmTMB::glmmTMB()` with `family = glmmTMB::ordinal()`) add a `Random effects` block and the mixed-model fit statistics (n, groups, AIC, BIC; no R-squared). Their inference is Wald only: AME, a robust `vcov`, `ci_method = "profile"` and `standardized` are refused, as is `nested = TRUE` for `clmm`. `exponentiate` follows the `clm` rule (odds ratios under logit, hazard ratios under cloglog), except for a `clmm` fit with the cloglog link, where it is refused: on the current `ordinal`, the cloglog and loglog links of `clmm()` do not match those of `clm()` and of `glmmTMB()`.

Counts and two-part models

Two-part models show their full model: the zero component renders as a `Zero-inflation` block (`zeroinfl`, glmmTMB `ziformula`: probability of a structural zero) or a `Zero hurdle` block (`hurdle`: probability of a nonzero count – the opposite direction, hence the distinct label), and a `Dispersion` block when `dispformula` has covariates. Component coefficients join the `p_adjust` family and take stars; a zero component is exponentiated only under a logit link (odds ratio). AME is the combined-response effect on E(Y). `"CR1"` for `pscl` fits covers both components via `sandwich::vcovCL()`, and `"CR0"` for `glmmTMB` fits covers the zero-inflation and dispersion blocks. Opt out with `show_components = FALSE`.

Categorical outcomes

`multinom` renders per non-reference outcome; `exponentiate` yields odds ratios of each outcome against the reference outcome – the baseline-category logits are log-odds (Agresti; SAS prints "Odds Ratio Estimates" under its generalized-logit link; Stata's `mlogit, rrr` labels the same quantity a relative-risk ratio). AME is per-outcome. `nested = TRUE` compares nested `multinom` fits by likelihood-ratio test (the `anova.multinom()` convention). Cluster-robust `"CR1"` is available (one cluster value per observation; sandwich >= 3.1-2) and the AME columns honor it; `HC*` is refused – a multi-equation model has no working residuals. `mlogit` renders per-alternative rows; AME is refused (no `slopes()` method exists for its data format). `"CR1"` is available with one cluster value per choice situation, and `n` counts choice situations; `HC*` is refused (`sandwich::vcovHC()` mis-scales the meat for mlogit's per-chooser score structure).

Survival models

Cox models exponentiate to hazard ratios; `survreg` log-scale distributions to time ratios (identity-scale distributions are left untouched). AME is refused for Cox fits (no marginal-probability effect on the hazard scale); their absolute-effect columns are the `"rmst"` and `"risk_diff"` families instead – covariate-adjusted RMST and cumulative-incidence differences by g-computation, with the mandatory `tau` / `at_time` horizons. For `coxph`: right-censored single-record fits, `strata()` supported (within-stratum baselines), `tt()` refused. For `survreg`: the closed-form AFT curves are standardized directly (stratified `survreg` refused). `"CR0"` uses the Lin-Wei grouped-dfbeta sandwich (`coxph`) or `rms::robcov()` (`cph`, needs `x = TRUE, y = TRUE`); `survreg` takes `"CR1"`, from `sandwich::vcovCL()`. `nested = TRUE` compares nested Cox fits by likelihood-ratio test.

Survey-design models

Fits from a `survey::svydesign()` or `survey::as.svrepdesign()` design – `svyglm` (and its replicate sibling `svrepglm`), `svyolr`, `svycoxph` (and `svrepcoxph`) – are read as design-based throughout: the coefficients, the variance and the reference distribution all come from survey.

Inference is Wald **t** at the degrees of freedom survey writes on the FIT – `df.residual` for `svyglm` / `svyolr`, `degf.resid` or `degf.residual` for the two Cox engines – which is what `survey::regTermTest()` takes as its denominator. It is not `survey::degf(design)`, and it is not re-derived here: the six engines of survey do not share one expression and are not harmonized (a Cox fit carries `degf(design) - p + 1` although it has no intercept for the `+ 1` to cancel, so it ends one above the two other classes). The value is read off the object. The footer names the design and prints the number, and the average marginal effects answer to the same distribution as the coefficient rows.

The average marginal effect is the Horvitz-Thompson estimator: the mean unit-level effect weighted by the sampling weights of the analytic sample, with its variance from the delta method on the design vcov.

Counts are both reported: the observed `n` and the `Weighted n` the estimates describe. `svycoxph` adds the number of events, and its concordance goes to the footer.

What is refused, and why: every model-derived variance (`HC*`, `CR*`, bootstrap, jackknife) – the design is the variance authority, and the way to change the estimator is to change the design; every likelihood statistic (AIC, BIC, logLik, deviance, pseudo-R-squared) for `svyolr` and `svycoxph` – there is no likelihood, and survey's own `deviance()` returns a sign-flipped likelihood-ratio statistic on one Cox engine and a bare zero on the other; the AME for `svycoxph`, on the same ground as for a plain Cox fit; and the `"rmst"` / `"risk_diff"` columns for `svycoxph`, whose uncertainty comes from resampling subjects and so ignores the strata and clusters the design declares (use `survey::svykm()` for a marginal curve).

`nested = TRUE` is refused for a design-based table: there is no likelihood to compare, so every change statistic would be empty, and a block of empty rows reads like an answer. Put the models side by side with `nested = FALSE`, and test a term under the design with `survey::regTermTest()`.

`svyglm` keeps an `AIC` row: survey's `extractAIC.svyglm` computes the design-based AIC of Lumley & Scott (2015), the one information criterion published for this class, and `show_fit_stats = "eff_p"` reports the effective number of design parameters beside it. `BIC.svyglm` requires a maximal model and has no default, so it stays blank. See the Summary tables from a survey design article.

Robust, IV, quantile and panel models

`estimatr` fits keep their own robust SEs (never overwritten); `quantreg::rq()` defaults to the heteroskedasticity-robust `"nid"` sandwich (quantreg's own large-sample default), with `"iid"`, `"ker"`, `"rank"` (CIs only) and a native `"bootstrap"` – clustered via the wild gradient bootstrap – as `vcov` options (the footer names the estimator); `fixest` fits – and `estimatr` fits built with `fixed_effects =` – disclose their absorbed fixed effects as a default-on `Fixed effects:` block (one Yes / No row per factor; varying-slope-only factors are not absorbed intercepts and read No), with per-factor `N (<factor>)` counts via the opt-in `n_groups` token and the within R-squared in the default fit statistics for `fixest` (opt-in via `within_r2` for `estimatr`).

Bayesian models

Posterior median, posterior MAD SD, and equal-tailed credible intervals (`ci_method = "hdi"` opts into the highest-density interval); deliberately no p-value column and no stars – the probability of direction (`"pd"`) is the opt-in posterior summary. A sampler-diagnostics guard checks every fit (R-hat, ESS, divergences, E-BFMI) and per-coefficient `"rhat"` / `"ess_bulk"` / `"ess_tail"` / `"mcse"` columns are available. The AME columns are draws-native (posterior median, MAD SD and credible interval of the per-draw `avg_slopes()`; no `"ame_p"`), and so are the standardized betas (`"posthoc"` / `"basic"` / `"smart"`, exact affine rescales of the draws) on fixed-effects fits: `stan_glm`-style models and standard-formula `brm()` models, whose design matrix is recovered through **insight**. Multilevel fits, `stan_polr` / `stan_betareg`, brms formulas with distributional or special terms, and `"refit"` / `"pseudo"` are refused with a pre-standardization hint. Multilevel fits (`stan_glmer`, `brm` with grouping terms) report their random effects as a block – posterior median SD and credible interval per component, from the draws – with no likelihood-ratio line. `p_adjust` and likelihood-based fit-statistic tokens are refused (no p-values, no likelihood-based information

criteria in a posterior); `"r2_bayes"` is in the default fit statistics and `"elpd_loo"` / `"looic"` / `"waic"` are opt-in, with standard errors and reliability caveats in the footer; compare models with `loo::loo_compare()` outside the table.

See Also

`table_regression()`; the Publication-ready regression tables and Ordinal regression tables articles.

Examples

```
table_regression_models()

# All engines of one family:
subset(table_regression_models(), family == "Mixed effects")
```

---

table_regression_uv *Univariable screening table (with optional multivariable merge)*

Description

Fits one model per candidate predictor (the *univariable screen*) and renders them as a single table with one row block per predictor. With `multivariable = TRUE` (default), the full model containing all predictors is merged side by side under `"Univariable"` / `"Multivariable"` column groups – the standard presentation of applied epidemiology (the `gtsummary::tbl_uvregression()` + `tbl_merge()` workflow).

Usage

```
table_regression_uv(
  data,
  outcome,
  predictors,
  method = c("lm", "glm", "coxph"),
  family = stats::binomial(),
  multivariable = TRUE,
  complete_cases = FALSE,
  show_columns = c("n", "b", "ci", "p"),
  show_intercept = FALSE,
  labels = getOption("spicy.var_labels", FALSE),
  title = NULL,
  ...,
  note_labels = NULL,
  note = NULL,
  note_add = NULL
)
```

Arguments

data	A data frame. A survey design is refused with a pointer to the design-based route (class spicy_wrong_regime).
outcome	The outcome column (unquoted name, tidyselect). For method = "coxph", a Surv(time, status) expression evaluated in data (the tbl_uvregression convention).
predictors	Candidate predictor columns (tidyselect, e.g. c(age, sex, education) or where(is.numeric)). The outcome column(s) are not part of the selection: everything() and the other helpers skip them, and naming one in predictors, or giving its position, is an error. A name that is not a column of data is an error (class spicy_missing_column).
method	"lm" (default), "glm", or "coxph" (requires the survival package; estimates render as HRs with exponentiate = TRUE).
family	A stats::family for method = "glm", in any of the three forms stats::glm() accepts: a family object (binomial()), its name ("binomial"), or the bare constructor (binomial). Default binomial(), so method = "glm" alone is the logistic screen; supplying family without method selects the glm screen directly (a family can only mean that). Refused for method = "coxph", and gaussian() with the identity link is refused too: use method = "lm" for the linear screen. With method = "lm", any non-gaussian family is refused the same way (use method = "glm"), and a supplied gaussian() is ignored with a classed warning – the linear screen already fits it.
multivariable	Logical, default TRUE: merge the full model (all predictors together) as a second column group.
complete_cases	Logical, default FALSE. TRUE restricts ALL models to the rows complete on outcome + every predictor (common-sample comparison); the reduction is disclosed in the table note.
show_columns	Passed to table_regression(). Default c("n", "b", "ci", "p") – the tbl_uvregression column set; "n" is the per-predictor sample size. The multivariable group carries no N column (its single n is a fit-statistics row, as in the reference layouts). For binary outcomes, add "n_events" for outcome event counts as events/N per factor level (each column group counts on its own estimation sample). For method = "lm", add "r2" (and/or "adj_r2") for the share of outcome variance each predictor explains **on its own** – see *Variance explained* below. For method = "coxph", the RMST / risk-difference families ("rmst", "risk_diff", ...) work with an explicit numeric tau / at_time shared by every column: each univariable fit runs its own boot_n-replicate bootstrap, and the multivariable group reports the covariate-adjusted estimand from the full fit. tau = "minmax" is refused (per-fit horizons would make the column incomparable across predictors).
show_intercept	Display the (Intercept) rows? Default FALSE – the opposite of table_regression()'s default, because each univariable fit carries its own nuisance intercept. See *Intercepts*.
labels	How the variables are named: the outcome in the title, a numeric predictor in its row, and a factor in its header. The four forms of table_regression():

- labels = FALSE shows the column names.

- `labels = TRUE` shows each column's `label` attribute (as set by `haven`, `labelled`, or `label_from_names()`) when it is present, not `NA`, and not empty, and the column name otherwise. Factor levels keep their names.
- `labels = c(age = "Age (years)", sexMale = "Men")`, a named character vector, gives the variables and rows it names these labels. Every other variable follows `options(spicy.var_labels)`: its name when the option is unset or `FALSE`, its `label` attribute when it is `TRUE`.
- `labels = var_labels(age = "Age in years")` gives the variables and rows it names these labels, and every other variable its `label` attribute, whatever the option says. `var_labels()` alone is `labels = TRUE`.

The names are the outcome and predictor columns and the coefficient names (`"sexMale"` names the row of one level), as in `table_regression()`. A name that matches nothing warns and is left out. The default reads `options(spicy.var_labels)`, and a `TRUE` or `FALSE` typed in the call wins over it. Any other value, `NULL` included, is an error. A variable label documents the data file; a published table carries the wording you set here. `note_labels` gives the outcome a shorter label in the title.

`title` Table title; `NULL` (default) builds `"Univariable and multivariable <type> regression: <outcome>"`.

`...` Passed to `table_regression()` (`exponentiate`, `vcov`, `cluster`, `p_adjust`, `digits`, `output`, ...), each by its full name: an unknown or abbreviated name is an error. `nested` is not meaningful for a screen and is refused. `cluster` must be a single vector with one value per row of `data`; it is aligned to each fit's own estimation sample automatically.

`note_labels` The label a variable takes where the table names it in a sentence, in the title and in the notes, in place of the one `labels` gives it: a named character vector, `note_labels = c(sex = "sex at birth")`, or a `var_labels()` object, which reads as one. `labels` can then name a row with a whole question while the note says "Missing values removed: sex at birth (2).". The rows, the column headers and the spanners keep the labels of `labels`, and so do the title and the notes for a variable `note_labels` does not name. A name that matches no column warns and is left out, as in `labels`, and `NULL` (the default) renames nothing. A note label that another variable of the table shows is an error (class `spicy_label_collision`): choose another one, or relabel the other variable.

`note` The note under the table. `NULL` (the default) builds it: the general note, one paragraph opened by the lead-in of the style ("Note." by default, none under the `jama` and `lancet` themes; see `note_label` in `spicy_style()`), then, in a `table_regression()` with `stars`, the legend of the stars on a line of its own. A single string replaces the general note, the sentences the table writes about its data included. It is printed as typed, line breaks (`"\n"`) included, after the lead-in of the style, which a string opening with a lead-in of its own does not repeat (`note = "Note. Data weighted."`), and the legend of the stars stays below it. `""` keeps that legend alone, and no note at all without stars. `FALSE` removes the note from every output, the legend included.

`note_add` Sentences to add to the note: a character vector of one or more strings, each printed as typed (no period is added) and joined to the note by a space. They

close its paragraph, after the definitions of the abbreviations and before the legend of the stars, in every output and in `attr(x, "note")`, and the sentences the table writes stay. With a string in `note`, they follow it, so a wrapper can add a house sentence to every table while one table replaces its note. An `NA` or an empty string is an error, and so is `note_add` with `note = FALSE`, which leaves no note to add to.

Value

See `table_regression()` (same output contract).

Sample sizes

By default each univariable model is fit on its **own complete cases**, so N varies across predictors – that is what the `N` column discloses (shown on the first row of each block), and a table note states it whenever the Ns differ. The multivariable model is fit on the complete cases of **all** its variables (its `n` appears in the fit-statistics rows). Pass `complete_cases = TRUE` to restrict every model – univariable and multivariable – to the common complete-case sample.

Variance explained

`show_columns = c("n", "b", "ci", "p", "r2")` adds an R^2 column to the screen (`method = "lm"`): each predictor block reports its **own** model's R^2, on the first row of the block like `N`. It answers what a coefficient and its interval cannot – how much of the outcome the predictor accounts for – and often shows that a firmly established association still explains a small share of the variance. Add `"adj_r2"` for the adjusted form. On the multivariable side the R^2 is one number for the whole model, so it stays in the fit-statistics rows (where it is shown by default) instead of being repeated down a column. Not available for `method = "glm"` or `"coxph"`: outside least squares only competing pseudo-R^2 measures exist, and spicy asks you to name the one you want (`show_fit_stats = "pseudo_r2_mcfadden"`).

Multiplicity

`p_adjust` (passed through to `table_regression()`) treats the whole univariable screen as ONE family (all screened coefficients together); the multivariable model is its own family, as in any multi-model table.

Why the default screen is linear

The default `method = "lm"` fits the linear screen: R's canonical model, and – when the outcome is continuous – the estimand with the most direct reading. If the outcome looks binary under this default, the screen proceeds as a linear probability model and says so in a classed warning: LPM coefficients are probability differences, comparable across models and samples in a way that odds ratios are not (Mood, 2010), but the model's built-in heteroskedasticity calls for `vcov = "HC3"`. A two-level factor (or logical) outcome is coded 0/1 on its second level – the glm convention – and the warning names the modeled probability; an outcome with more observed levels is refused (a multinomial outcome has no linear screen). The classical epidemiological screen is one argument away: `method = "glm"` (with the default `family = binomial()`) gives the logistic screen, and supplying any `family` selects the glm screen directly.

Intercepts

Hidden by default on both sides (each univariable fit has its own nuisance intercept), matching `gtsummary::tbl_regression()`'s `intercept = FALSE` default. Pass `show_intercept = TRUE` to display them: each univariable block then opens with its own fit's `(Intercept)` row, and the multivariable model shows its intercept as in any `table_regression()` table.

References

Batra, N. et al. (Eds.) (2021). *The Epidemiologist R Handbook*, Univariate and multivariable regression. https://epirhandbook.com/en/new_pages/regression.html

Mood, C. (2010). Logistic regression: Why we cannot do what we think we can do, and what we can do about it. *European Sociological Review*, 26(1), 67-82. doi:10.1093/esr/jcp006

Sjoberg, D.D., Whiting, K., Curry, M., Lavery, J.A., & Larmarange, J. (2021). Reproducible summary tables with the gtsummary package. *The R Journal*, 13(1), 570-580.

Examples

```
table_regression_uv(
  sochealth,
  outcome    = smoking,
  predictors = c(age, sex, education),
  family     = binomial(),
  exponentiate = TRUE
)

# Linear screen with the share of variance each predictor
# explains on its own (the multivariable model reports its own
# R-squared in the fit-statistics rows).
table_regression_uv(
  sochealth,
  outcome      = wellbeing_score,
  predictors   = c(age, sex, bmi),
  show_columns = c("n", "b", "ci", "p", "r2")
)
```

---

uncertainty_coef *Uncertainty coefficient*

Description

`uncertainty_coef()` computes the Uncertainty Coefficient (Theil's U) for a two-way contingency table, based on information entropy.

Usage

```
uncertainty_coef(
  x,
  direction = c("symmetric", "row", "column"),
  detail = FALSE,
  conf_level = 0.95,
  digits = 3L
)
```

Arguments

x	A contingency table (of class table).
direction	Direction of prediction: "symmetric" (default), "row" (column predicts row), or "column" (row predicts column).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

The uncertainty coefficient measures association using Shannon entropy. Let H_X and H_Y be the marginal entropies of the **row** and **column** variables respectively, and H_{XY} the joint entropy.

- direction = "row" (column predicts row): $U = (H_X + H_Y - H_{XY})/H_X$.
- direction = "column" (row predicts column): $U = (H_X + H_Y - H_{XY})/H_Y$.
- direction = "symmetric": $U = 2(H_X + H_Y - H_{XY})/(H_X + H_Y)$.

The default direction = "symmetric" follows the SPSS and DescTools convention: the symmetric coefficient is a standard, well-defined variant with its own asymptotic standard error. somers_d() deliberately differs (its default is "row") because its symmetric form is a derived quantity without an analytic SE; see its documentation.

When the marginal entropy in the denominator is zero (the predicted variable is constant, e.g. an unused factor level), the coefficient is the undefined form $0/0$: the function returns NA with a spicy_undefined_stat warning, like the other measures in the family. For direction = "symmetric" this happens only when both variables are constant; with a single constant variable the symmetric coefficient is a well-defined 0.

The entropy terms use the standard mathematical convention $0 \log 0 = 0$, matching SPSS / PSPP CROSSTABS and the definition in Cover & Thomas (2006). Note that DescTools::UncertCoef() applies an additional Laplace correction (replacing zero cells with $1/n^2$) before the entropy computation, which produces slightly different point estimates on tables with empty cells; that correction is uncommon in the information-theory literature and is not used here. The asymptotic standard errors follow the DescTools delta method; see cramer_v() for full references.

Value

Same structure as `cramer_v()`: a scalar when `detail = FALSE`, a named vector when `detail = TRUE`. The p-value tests H0: U = 0 (Wald z-test).

References

Theil, H. (1970). On the estimation of relationships involving qualitative variables. *American Journal of Sociology*, 76(1), 103-154. doi:10.1086/224909

See Also

`lambda_gk()`, `goodman_kruskal_tau()`, `assoc_measures()`

Other association measures: `assoc_measures()`, `contingency_coef()`, `cramer_v()`, `gamma_gk()`, `goodman_kruskal_tau()`, `kendall_tau_b()`, `kendall_tau_c()`, `lambda_gk()`, `phi()`, `somers_d()`, `yule_q()`

Examples

```
tab <- table(sochealth$smoking, sochealth$education)
uncertainty_coef(tab)
uncertainty_coef(tab, direction = "row", detail = TRUE)
```

---

var_labels *Variable labels for a table*

Description

Builds a value for the `labels` argument of the descriptive tables (`table_categorical()`, `table_continuous()`, `table_continuous_lm()`, `table_outcome()`, `table_categorical_svy()` and `table_continuous_svy()`) and of the regression tables (`table_regression()` and `table_regression_uv()`). The variables it names take the labels given here, and every other variable shows its `label` attribute (as set by haven, `labelled` or `label_from_names()`), or its name when it has none, whatever `options(spicy.var_labels)` says. `var_labels()` with no argument is the same as `labels = TRUE`.

A plain named vector, `labels = c(bmi = "Body mass index")`, differs in one point: the variables it leaves out follow `options(spicy.var_labels)`, which shows their names unless it is `TRUE`.

In a regression table, the names are the model's: the outcome, the term labels (`age`, or `sex` for the header of a factor), and the coefficient names (`sexMale`, `(Intercept)`). The attribute names the outcome, the numeric terms, and the factor headers. Factor levels, interactions, and transformed terms keep their names.

Usage

```
var_labels(...)
```

Arguments

`...`	`name = "label"` pairs, one per variable to rename, each label a single string. A named character vector can be spliced with `!!!`. A name that matches nothing in a table warns there and is left out.

Value

A named character vector of class `spicy_var_labels`, which prints as the plain named vector.

See Also

The `labels` argument of `table_categorical()` and its siblings, and of `table_regression()`, which lists the four forms it takes.

Examples

```
# The label attribute of every column, one rename.
table_continuous(
  sochealth,
  select = c(bmi, age),
  labels = var_labels(bmi = "BMI (kg/m2)")
)

# A vector of renames kept for a whole project, spliced.
renames <- c(bmi = "BMI (kg/m2)", smoking = "Smoker")
tbl <- table_categorical(
  sochealth,
  select = smoking,
  by = sex,
  labels = var_labels(!!!renames)
)

# A regression table: the outcome and the factor header keep their
# label attribute, the numeric term takes the one given here.
table_regression(
  lm(wellbeing_score ~ age + sex, data = sochealth),
  labels = var_labels(age = "Age in years")
)
```

`varlist` *Generate a comprehensive summary of the variables*

Description

`varlist()` lists the variables of a data frame and extracts essential metadata: variable names, labels, summary values, classes, number of distinct values, number of valid (non-missing) observations, and number of missing values. Tidyselect-style selectors can be supplied to pick or reorder columns dynamically.

`vl()` is a convenient shorthand for `varlist()` that offers identical functionality with a shorter name.

Usage

```
varlist(
  x,
  ...,
  values = FALSE,
  tbl = FALSE,
  include_na = FALSE,
  factor_levels = c("observed", "all"),
  user_na = TRUE
)

vl(
  x,
  ...,
  values = FALSE,
  tbl = FALSE,
  include_na = FALSE,
  factor_levels = c("observed", "all"),
  user_na = TRUE
)
```

Arguments

x	A data frame, or a transformation of one.
...	Optional tidyselect-style column selectors (e.g. `starts_with("var")`, `where(is.numeric)`, etc.). Columns can be selected or reordered, but renaming selections is not supported.
values	Logical. If `FALSE` (the default), displays a compact summary of the variable's values. For numeric, character, date/time, labelled, and factor variables, all unique non-missing values are shown when there are at most four; otherwise the first three values, an ellipsis (`...`), and the last value are shown. Values are sorted when appropriate (e.g., numeric, character, date). For factors, `factor_levels` controls whether observed or all declared levels are shown; level order is preserved. For labelled variables, prefixed labels are displayed via `labelled::to_factor(levels = "prefixed")`. If `TRUE`, all unique non-missing values are displayed.
tbl	Logical. If `FALSE` (the default), opens the summary in the Viewer if the session is interactive. If `TRUE`, returns a tibble.
include_na	Logical. If `TRUE`, unique missing value markers (<NA>, <NaN>) are explicitly appended at the end of the `Values` summary when present in the variable. This applies to all variable types. Literal strings `"NA"`, `"NaN"`, and `""` are quoted to distinguish them from missing markers. If `FALSE` (the default), missing values are omitted from `Values` but still counted in the `NAs` column.
factor_levels	Character. Controls how factor values are displayed in `Values`. `"observed"` (the default; code_book() uses `"all"`) shows only levels present in the data, preserving factor level order. `"all"` shows all declared levels, including unused

levels. An explicit NA level (e.g. from addNA()) is displayed as <NA> among the declared levels.

user_na — Logical. If TRUE (the default), declared missing values count as missing in N_valid, NAs, and N_distinct (all three columns share one missing definition). If FALSE, they count as valid. Either way, the declared codes remain listed in Values (with their value labels when declared) – a codebook documents the full coding scheme. See the "Declared missing values" section of freq().

Details

In an interactive session (RStudio, Positron, ...), the summary opens in the Viewer pane with a contextual title like vl: sochealth. If the data frame has been transformed or subsetted, the title is suffixed with * (e.g. vl: sochealth*); anonymous or ambiguous calls fall back to vl: <data>. Pass tbl = TRUE to return a tibble instead.

The default factor_levels = "observed" mirrors what is actually in the data; code_book() defaults to "all" to document the declared schema. See @param factor_levels to override either default.

Value

A tibble with one row per selected variable, containing the following columns:

- Variable: variable names
- Label: variable labels (if available via the label attribute)
- Values: a summary of the variable's values, depending on the values and include_na arguments. If values = FALSE, a compact summary is shown: all unique values when there are at most four, otherwise 3 + ... + last. If values = TRUE, all unique non-missing values are displayed. For labelled variables, **prefixed labels** are displayed using labelled::to_factor(levels = "prefixed"). For factors, levels are displayed according to factor_levels. Matrix and array columns are summarized by their dimensions. difftime values are annotated with their units, e.g. 1.5, 2.5 (hours). Missing value markers (<NA>, <NaN>) are optionally appended at the end (controlled via include_na). Literal strings "NA", "NaN", and "" are quoted to distinguish them from missing markers.
- Class: the class of each variable (possibly multiple, e.g. "labelled", "numeric")
- N_distinct: number of distinct non-missing values
- N_valid: number of non-missing observations
- NAs: number of missing observations

For matrix and array columns, observations are counted per **row**: a row is treated as missing if any of its cells is NA. N_valid / NAs therefore count complete vs. incomplete rows, not individual cells.

With tbl = FALSE (the default) the tibble is sent to the Viewer (interactive) or surfaced via a message (non-interactive) and the function returns invisibly NULL. Set tbl = TRUE to return the tibble directly for downstream use.

Declared missing values

Survey files imported with **haven** often carry *declared missing values*: codes such as `8 = Don't know` or `9 = Refused` that the source file marks as missing while keeping them distinct from a plain `NA`. Two kinds of declaration exist: `na_values` / `na_range` metadata on `haven::labelled_spss()` vectors, and tagged missing values created by `haven::tagged_na()` (the Stata `.a`, `.b`, ... convention). `haven::read_sav()` keeps the `na_values` / `na_range` declaration only with `user_na = TRUE`: its default turns the declared codes into `NA` on import.

spicy honors the declaration by default (`user_na = TRUE`): declared missing values are excluded from every statistic exactly like `NA` – valid percentages, means, chi-squared tests, association measures, row-wise summaries, and group definitions – but they are not erased from display. `freq()` lists each observed declared value as its own row of the Missing block, with its value label; `cross_tab()`, `table_categorical()`, and `table_continuous()` disclose the exclusion in the table note (`Declared missing values remo`). `varlist()` counts them as missing in `N_valid` / `NAs` / `N_distinct` while still listing the declared codes in `Values`; `code_book()` counts them in `n_declared_missing` and flags them in its `values` table.

Every function involved offers the same escape hatch: set `user_na = FALSE` to ignore the declaration and treat the declared codes as valid values (the behavior of spicy before 0.13.0). Tagged missing values are genuine `NA`s either way; for them, `user_na = FALSE` only collapses the per-tag breakdown back into the regular `NA` count.

See Also

Other variable inspection: `code_book()`, `label_from_names()`

Examples

```
varlist(sochealth, tbl = TRUE)
sochealth |> varlist(tbl = TRUE)
varlist(sochealth, where(is.numeric), values = TRUE, tbl = TRUE)
varlist(
  sochealth,
  starts_with("bmi"),
  values = TRUE,
  include_na = TRUE,
  tbl = TRUE
)

df <- data.frame(
  group = factor(c("A", "B", NA), levels = c("A", "B", "C"))
)
varlist(
  df,
  values = TRUE,
  include_na = TRUE,
  factor_levels = "all",
  tbl = TRUE
)

vl(sochealth, tbl = TRUE)
sochealth |> vl(tbl = TRUE)
```

```
vl(sochealth, starts_with("bmi"), tbl = TRUE)
vl(sochealth, where(is.numeric), values = TRUE, tbl = TRUE)
```

---

yule_q *Yule's Q*

Description

yule_q() computes Yule's Q coefficient of association for a 2x2 contingency table.

Usage

```
yule_q(x, detail = FALSE, conf_level = 0.95, digits = 3L)
```

Arguments

x	A contingency table (of class table).
detail	Logical. If FALSE (default), return the estimate as a numeric scalar. If TRUE, return a named numeric vector including confidence interval and p-value.
conf_level	A single number strictly between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval. Any other value – including percentages such as 95 – raises a classed error (spicy_invalid_input).
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.

Details

For a 2x2 table with cells a, b, c, d, Yule's Q is $Q = (ad - bc)/(ad + bc)$. It is equivalent to the Goodman-Kruskal Gamma for 2x2 tables. The asymptotic standard error is $SE = 0.5(1 - Q^2)\sqrt{1/a + 1/b + 1/c + 1/d}$.

Edge cases: when ad + bc = 0, Q itself is undefined and the function returns NA with a spicy_undefined_stat warning. When any cell is zero (and ad + bc > 0), Q is well-defined but the SE formula divides by zero – the point estimate is returned, and se, ci_lower, ci_upper, and p_value are all NA.

Standard error formulas follow the DescTools implementations (Signorell et al., 2024); see cramer_v() for full references.

Value

Same structure as cramer_v(): a scalar when detail = FALSE, a named vector when detail = TRUE. The p-value tests H0: Q = 0 (Wald z-test).

References

Yule, G. U. (1900). On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London, Series A*, 194, 257-319. doi:10.1098/rsta.1900.0019

See Also

phi(), gamma_gk(), assoc_measures()

Other association measures: assoc_measures(), contingency_coef(), cramer_v(), gamma_gk(), goodman_kruskal_tau(), kendall_tau_b(), kendall_tau_c(), lambda_gk(), phi(), somers_d(), uncertainty_coef()

Examples

```
tab <- table(sochealth$smoking, sochealth$sex)
yule_q(tab)
```

Index

∗ **association measures**
  assoc_measures, 8
  contingency_coef, 15
  cramer_v, 22
  gamma_gk, 33
  goodman_kruskal_tau, 34
  kendall_tau_b, 38
  kendall_tau_c, 39
  lambda_gk, 42
  phi, 47
  somers_d, 51
  uncertainty_coef, 192
  yule_q, 199
∗ **datasets**
  sochealth, 49
∗ **row-wise summaries**
  count_n, 19
  mean_n, 44
  sum_n, 63
∗ **spicy tables**
  table_categorical, 66
  table_continuous, 90
  table_continuous_lm, 107
∗ **variable inspection**
  code_book, 9
  label_from_names, 41
  varlist, 195

addNA(), *12*, *197*
as.data.frame(), *75*, *88*, *98, 99*, *117*, *133*, *142*, *152*, *178*
as.data.frame.spicy_continuous_table(), *98*
as.data.frame.spicy_regression_table(), *168*
as_flextable.spicy_flextable, 3
as_structured, 4
as_structured(), *36*, *38*, *74*, *77, 78*, *89, 90*, *98, 99*, *103*, *136*, *142*, *144*, *154*, *163*, *171*, *179*, *185*
assoc_measures, 8
assoc_measures(), *16*, *23*, *34, 35*, *39, 40*, *44*, *48*, *52*, *194*, *200*

boot::boot.ci(), *157*
broom::glance(), *74*, *98*, *178*, *180*
broom::tidy(), *74*, *76*, *98*, *178*, *180*
build_ascii_table(), *180*

clipr::write_clip(), *17*, *178*
clubSandwich::coef_test(), *112*, *120*, *123*, *172*
clubSandwich::linear_contrast(), *172*
clubSandwich::vcovCR(), *112*, *120*, *123*, *171*, *174*
clubSandwich::Wald_test(), *112*, *120*, *123*
cobalt::bal.tab(), *119*, *121*, *123*
code_book, 9
code_book(), *14*, *21*, *27*, *29–31*, *42*, *45*, *50*, *57*, *64*, *79*, *103*, *122*, *196–198*
contingency_coef, 15
contingency_coef(), *8, 9*, *23*, *34, 35*, *39, 40*, *44*, *48*, *52*, *194*, *200*
copy_clipboard, 17
count_n, 19
count_n(), *46*, *64*
cramer_v, 22
cramer_v(), *8, 9*, *16*, *33–35*, *39, 40*, *43, 44*, *48*, *52*, *193, 194*, *199, 200*
cross_tab, 24
cross_tab(), *14*, *21*, *27*, *29–31*, *45*, *50*, *54*, *64*, *66*, *68*, *70–72*, *76*, *79, 80*, *103, 104*, *122*, *180*, *198*

datawizard::row_count(), *21*

effectsize::cohens_d(), *123*
effectsize::hedges_g(), *123*
effectsize::omega_squared(), *119*, *123*
emmeans::emmeans(), *110*

flextable::flextable(), *3*
forcats::fct_relevel(), *109*, *113*
freq, 28
freq(), *14*, *20, 21*, *24–27*, *31*, *45*, *54*, *64*, *68*, *74*, *79, 80*, *87*, *97*, *103, 104*, *117*, *122*, *133*, *139*, *151*, *180*, *197, 198*

gamma_gk, 33
gamma_gk(), *8, 9*, *16*, *23*, *35*, *39, 40*, *44*, *48*, *52*, *194*, *200*
geepack::QIC(), *170*
goodman_kruskal_tau, 34
goodman_kruskal_tau(), *8, 9*, *16*, *23*, *34*, *39, 40*, *44*, *48*, *52*, *194*, *200*

haven::labelled_spss(), *14*, *21*, *27*, *31*, *45*, *64*, *79*, *103*, *122*, *198*
haven::read_sav(), *14*, *21*, *27*, *31*, *45*, *64*, *79*, *103*, *122*, *198*
haven::tagged_na(), *14*, *21*, *27*, *31*, *45*, *64*, *79*, *103*, *122*, *198*
Hmisc::wtd.quantile(), *100*

inline, 35
inline(), *79*, *171*

kendall_tau_b, 38
kendall_tau_b(), *8, 9*, *16*, *23*, *34, 35*, *40*, *44*, *48*, *52*, *194*, *200*
kendall_tau_c, 39
kendall_tau_c(), *8, 9*, *16*, *23*, *34, 35*, *39*, *44*, *48*, *52*, *194*, *200*

label_from_names, 41
label_from_names(), *14*, *68*, *85*, *95*, *114*, *131*, *140*, *148*, *165*, *190*, *194*, *198*
labelled::var_label(), *41, 42*, *50*
lambda_gk, 42
lambda_gk(), *8, 9*, *16*, *23*, *34, 35*, *39, 40*, *48*, *52*, *194*, *200*

marginaleffects::avg_predictions(), *110*
MASS::confint.glm(), *157*
mean_n, 44
mean_n(), *19–21*, *64*

openxlsx2::write_xlsx(), *178*

phi, 47
phi(), *8, 9*, *16*, *23*, *34, 35*, *39, 40*, *44*, *52*, *194*, *200*
print.spicy_cross_table(), *26*
print.spicy_cross_table_list(), *26*
print.spicy_freq_table(), *28*, *31*

rms::robcov(), *172, 173*

sandwich::vcovCL(), *112*, *172–174*
sandwich::vcovHC(), *112*, *119, 120*, *123*, *171*
set.seed(), *70*, *76*, *143*
sochealth, 49
somers_d, 51
somers_d(), *8, 9*, *16*, *23*, *34, 35*, *39, 40*, *43, 44*, *48*, *193, 194*, *200*
spicy, *179*
spicy_labels, 53
spicy_labels(), *12*, *59*, *179*
spicy_print_table(), *28*, *31*, *178*, *180*
spicy_style, 55
spicy_style(), *54*, *72–74*, *86, 87*, *96, 97*, *99*, *115–117*, *131–133*, *140, 141*, *150, 151*, *165*, *168*, *178*, *190*
spicy_style_names *(*spicy_style*)*, 55
stats::chisq.test(), *76*
stats::coef(), *159*
stats::confint(), *157*
stats::drop1(), *113*, *119*
stats::family, *189*
stats::fisher.test(), *76, 77*, *142*
stats::glm(), *189*
stats::median(), *99*
stats::p.adjust(), *71*, *77*, *86*, *90*, *94*, *103*, *130*, *135*, *139*, *149*, *154*, *159*
stats::quantile(), *99*
stats::relevel(), *113*
stats::terms(), *166*
stats::weights(), *179*
sum_n, 63
sum_n(), *19–21*, *46*
survey::regTermTest(), *186, 187*
survey::svykm(), *187*
survival::basehaz(), *177*
survival::survfit(), *177*

table_categorical, 66
table_categorical(), *4–7*, *14*, *21*, *27*, *31*, *36, 37*, *45*, *50*, *53, 54*, *56, 57*, *64*, *79*,

83–85, *87*, *89, 90*, *92, 93*, *99*, *103, 104*, *109*, *121–123*, *137–140*, *142–145*, *148, 149*, *153, 154*, *178*, *180*, *194, 195*, *198*
table_categorical_svy, 83
table_categorical_svy(), *4–6*, *36, 37*, *57*, *69*, *77*, *136*, *178*, *194*
table_continuous, 90
table_continuous(), *4–7*, *14*, *21*, *27*, *31*, *36, 37*, *45*, *53, 54*, *57*, *64*, *68, 69*, *73*, *77–80*, *86*, *103*, *109*, *113*, *118*, *122, 123*, *128–130*, *133–136*, *139*, *146*, *148–154*, *178*, *180*, *194*, *198*
table_continuous_lm, 107
table_continuous_lm(), *4–7*, *36, 37*, *53, 54*, *57*, *68*, *72, 73*, *77*, *80*, *91*, *93, 94*, *96*, *100, 101*, *103, 104*, *132*, *139*, *149*, *151*, *153*, *157, 158*, *178*, *180*, *194*
table_continuous_svy, 128
table_continuous_svy(), *4, 5*, *36, 37*, *57*, *90*, *178*, *194*
table_multiresponse, 136
table_multiresponse(), *4*, *6, 7*, *36, 37*, *50*, *57*, *112*, *157*, *178*
table_outcome, 146
table_outcome(), *4–6*, *36*, *53, 54*, *57*, *77*, *90*, *103*, *135*, *154*, *178*, *194*
table_regression, 155
table_regression(), *4*, *7*, *36*, *53, 54*, *57*, *71*, *73*, *76*, *78*, *86*, *90*, *94*, *97*, *99*, *103*, *111, 112*, *116*, *120, 121*, *130*, *132*, *135, 136*, *139, 140*, *149*, *151*, *154*, *183*, *185*, *188–192*, *194, 195*
table_regression_bayesian *(*table_regression_models*)*, 183
table_regression_categorical *(*table_regression_models*)*, 183
table_regression_counts *(*table_regression_models*)*, 183
table_regression_mixed *(*table_regression_models*)*, 183
table_regression_models, 183
table_regression_models(), *180*
table_regression_ordinal *(*table_regression_models*)*, 183
table_regression_robust *(*table_regression_models*)*, 183
table_regression_survival *(*table_regression_models*)*, 183
table_regression_uv, 188
table_regression_uv(), *53*, *57*, *169*, *178*, *194*
tidy(), *77*, *88, 89*, *142*
tidyselect::all_of(), *19*
tidyselect::any_of(), *19*
tidyselect::starts_with(), *19*
tryCatch(), *179*

uncertainty_coef, 192
uncertainty_coef(), *8, 9*, *16*, *23*, *34, 35*, *39, 40*, *44*, *48*, *52*, *200*

var_labels, 194
var_labels(), *53*, *69*, *85*, *95*, *115*, *131*, *140*, *148*, *166*, *190*
varlist, 195
varlist(), *11*, *14*, *21*, *27*, *30, 31*, *42*, *45*, *50*, *64*, *79*, *103*, *122*, *198*
vl *(*varlist*)*, 195

withCallingHandlers(), *179*

yule_q, 199
yule_q(), *8, 9*, *16*, *23*, *34, 35*, *39, 40*, *44*, *48*, *52*, *194*