Package 'spicy'

March 23, 2026

Title Descriptive Statistics and Data Management Tools

Version 0.6.0

Description Provides tools for descriptive data analysis, variable inspection, and quick tabulation workflows in 'R'. Summarizes variable metadata, labels, classes, missing values, and representative values, with support for readable frequency tables, cross-tabulations, association measures for contingency tables (Cramer's V, Phi, Goodman-Kruskal Gamma, Kendall's Tau-b, Somers' D, and others), and APA-style reporting tables. Includes helpers for interactive codebooks, variable label extraction, clipboard export, and row-wise descriptive summaries. Designed to make descriptive analysis faster, clearer, and easier to work with in practice.

License MIT + file LICENSE

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

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

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Imports crayon, dplyr, haven, labelled, rlang, stats, stringr, tibble, tidyselect, utils

Suggests clipr, DT, flextable, gt, knitr, officer, openxlsx, rmarkdown, testthat (>= 3.0.0), tinytable

VignetteBuilder knitr

Depends R (>= 4.1.0)

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Amal Tawfik [aut, cre, cph] (ORCID: <https://orcid.org/0009-0006-2422-1555>, ROR: <https://ror.org/04j47fz63>)

Maintainer Amal Tawfik <amal.tawfik@hesav.ch>

Repository CRAN

Date/Publication 2026-03-23 08:50:02 UTC

Contents

assoc_measures . . . 2
code_book . . . 4
contingency_coef . . . 5
copy_clipboard . . . 6
count_n . . . 8
cramer_v . . . 12
cross_tab . . . 14
freq . . . 16
gamma_gk . . . 19
goodman_kruskal_tau . . . 20
kendall_tau_b . . . 22
kendall_tau_c . . . 23
label_from_names . . . 24
lambda_gk . . . 25
mean_n . . . 26
phi . . . 29
print.spicy_assoc_detail . . . 30
print.spicy_assoc_table . . . 30
print.spicy_cross_table . . . 31
print.spicy_freq_table . . . 32
sochealth . . . 33
somers_d . . . 35
spicy_print_table . . . 36
spicy_tables . . . 38
sum_n . . . 39
table_apa . . . 41
uncertainty_coef . . . 45
varlist . . . 46
yule_q . . . 48

Index **50**

`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 number between 0 and 1 giving the confidence level (default 0.95). Set to NULL to omit the confidence interval.
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().

Standard error formulas follow the DescTools implementations (Signorell et al., 2024).

Value

A data frame with columns measure, estimate, se, ci_lower, ci_upper, and p_value. For nominal measures (Cramer's V, Phi, Contingency Coef.), the p-value comes from the Pearson chi-squared test of independence. For all other measures, it is a Wald z-test of H0: measure = 0.

References

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

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()

Examples

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

code_book *Generate an interactive variable codebook*

Description

code_book() creates an interactive and exportable codebook summarizing all variables of a data frame. It builds upon varlist() to provide an overview of variable names, labels, classes, and representative values in a sortable, searchable table.

The output is displayed as an interactive DT::datatable() in the Viewer pane, allowing filtering, column reordering, and export (copy, print, CSV, Excel, PDF) directly.

Usage

```
code_book(x, values = FALSE, include_na = FALSE, title = "Codebook", ...)
```

Arguments

x	A data frame or tibble.
values	Logical. If FALSE (the default), displays a compact summary of the variable's values. For numeric, character, date/time, labelled, and factor variables, up to four unique non-missing values are shown: the first three values, followed by an ellipsis (...), and the last value. Values are sorted when appropriate (e.g., numeric, character, date) For factors, the levels are used directly and are not sorted. For labelled variables, prefixed labels are displayed via labelled::to_factor(levels = "prefixed"). If TRUE, all unique non-missing values are displayed.
include_na	Logical. If TRUE, unique missing values (NA, NaN) are explicitly appended at the end of the Values summary when present in the variable. This applies to all variable types. If FALSE (the default), missing values are omitted from Values but still counted in the NAs column.
title	Optional character string displayed as the table title in the Viewer. Defaults to "Codebook". Set to NULL to remove the title completely.
...	Additional arguments (currently unused).

Details

- The interactive datatable supports column sorting, searching, and client-side export to various formats.
- All exports occur client-side through the Viewer or Tab.

Value

A DT::datatable object.

Dependencies

Requires the following package:

- **DT**

See Also

`varlist()` for generating the underlying variable summaries.

Examples

```
## Not run:
# Launch the interactive codebook (opens in Viewer)
code_book(sochealth)

## End(Not run)
```

---

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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

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.

See Also

`cramer_v()`, `assoc_measures()`

Examples

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

---

copy_clipboard *Copy data to the clipboard*

Description

`copy_clipboard()` copies a data frame, matrix, array (2D or higher), table or vector to the clipboard. You can paste the result into a text editor (e.g. Notepad++, Sublime Text), a spreadsheet (e.g. Excel, LibreOffice Calc), or a word processor (e.g. Word).

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 is supplied, it is used as the column name for row names.
row.names	Logical. If TRUE (the default), includes row names in the clipboard output. If FALSE, row names are omitted.
col.names	Logical. If TRUE (the default), includes column names in the clipboard output. If FALSE, column names are omitted.
show_message	Logical. If TRUE (the default), displays a success message after copying. If FALSE, no success message is printed.
quiet	Logical. If FALSE (the default), messages are shown. If TRUE, suppresses all messages, including success, coercion notices, and warnings.
...	Additional arguments passed to clipr::write_clip().

Details

Note: Objects that are not data frames or 2D matrices (e.g. atomic vectors, arrays, tables) are automatically converted to character when copied to the clipboard, as required by clipr::write_clip(). The original object in R remains unchanged.

For multidimensional arrays (e.g. 3D arrays), the entire array is flattened into a 1D character vector, with each element on a new line. To preserve a tabular structure, you should extract a 2D slice before copying. For example: copy_clipboard(my_array[, , 1]).

Value

Invisibly returns the object x. The main purpose is the side effect of copying data to the clipboard.

Examples

```
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)
}
```

---

count_n *Row-wise Count of Specific or Special Values*

Description

count_n() counts, for each row of a data frame or matrix, how many times one or more values appear across selected columns. It supports type-safe comparison, case-insensitive string matching, and detection of special values such as NA, NaN, Inf, and -Inf.

Usage

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

Arguments

data	A data frame or matrix. Optional inside mutate().
select	Columns to include. Defaults to tidyselect::everything(). Uses tidyselect helpers like tidyselect::starts_with(), etc. If regex = TRUE, select is treated as a regex string.
exclude	Character vector of column names to exclude after selection. 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.
special	Character vector of special values to count: "NA", "NaN", "Inf", "-Inf", or "all". Defaults to NULL. "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.

Details

This function is particularly useful for summarizing data quality or patterns in row-wise structures, and is designed to work fluently inside dplyr::mutate() pipelines.

Internally, count_n() wraps the stable and dependency-free base function base_count_n(), allowing high flexibility and testability.

Value

A numeric vector of row-wise counts (unnamed).

Note

This function is inspired by datawizard::row_count(), but provides additional flexibility:

- **Element-wise type-safe matching** using identical() when allow_coercion = FALSE. This ensures that both the value and its type match exactly, enabling precise comparisons in mixed-type columns.
- **Support for multiple values in** count, allowing queries like count = c(2, 3) or count = c("yes", "no") to count any of several values per row.
- **Detection of special values** such as NA, NaN, Inf, and -Inf through the special argument — a feature not available in row_count().
- **Tidyverse-native behavior**: can be used inside mutate() without explicitly passing a data argument.

Value coercion behavior:

R automatically coerces mixed-type vectors passed to count into a common type. For example, count = c(2, "2") becomes c("2", "2"), because R converts numeric and character values to a unified type. This means that mixed-type checks are not possible at runtime once count is passed

to the function. To ensure accurate type-sensitive matching, users should avoid mixing types in `count` explicitly.

Strict matching mode (`allow_coercion = FALSE`)**:**

When strict matching is enabled, each value in `count` must match the type of the target column exactly.

For factor columns, this means that `count` must also be a factor. Supplying `count = "b"` (a character string) will not match a factor value, even if the label appears identical.

A common and intuitive approach is to use `count = factor("b")`, which works in many cases. However, `identical()` — used internally for strict comparisons — also checks the internal structure of the factor, including the order and content of its levels. As a result, comparisons may still fail if the levels differ, even when the label is the same.

To ensure a perfect match (label **and** levels), you can reuse a value taken directly from the data (e.g., `df$x[2]`). This guarantees that both the class and the factor levels align. However, this approach only works reliably if all selected columns have the same factor structure.

Case-insensitive matching (`ignore_case = TRUE`)**:**

When `ignore_case = TRUE`, all values involved in the comparison are converted to lowercase using `tolower()` before matching. This behavior applies to both character and factor columns. Factors are first converted to character internally.

Importantly, this case-insensitive mode takes precedence over strict type comparison: values are no longer compared using `identical()`, but rather using lowercase string equality. This enables more flexible matching — for example, `"b"` and `"B"` will match even when `allow_coercion = FALSE`.

Example: strict vs. case-insensitive matching with factors:

```
df <- tibble::tibble(
  x = factor(c("a", "b", "c")),
  y = factor(c("b", "B", "a"))
)

# Strict match fails with character input
count_n(df, count = "b", allow_coercion = FALSE)
#> [1] 0 0 0

# Match works only where factor levels match exactly
count_n(df, count = factor("b", levels = levels(df$x)), allow_coercion = FALSE)
#> [1] 0 1 0

# Case-insensitive match succeeds for both "b" and "B"
count_n(df, count = "b", ignore_case = TRUE)
#> [1] 1 2 0
```

Like `datawizard::row_count()`, this function also supports regex-based column selection, case-insensitive string comparison, and column exclusion.

Examples

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

```
library(haven)

# 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")
)
df
count_n(df, count = 2)
count_n(df, count = 2, allow_coercion = FALSE)
count_n(df, count = "2", ignore_case = TRUE)
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)
)
df
count_n(df, count = 2)
count_n(df, count = 2, allow_coercion = FALSE)
count_n(df, count = "b", ignore_case = FALSE)
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_n(df, count = TRUE, select = logic)
count_n(df, count = 2, select = lab)
df <- df |> mutate(lab_chr = as_factor(lab))
count_n(df, count = "Yes", select = lab_chr, allow_coercion = TRUE)
count_n(df, count = "Yes", select = lab_chr, allow_coercion = FALSE)

# Count special values
count_n(df, special = "NA")
count_n(df, special = "NaN")
count_n(df, special = "-Inf")
count_n(df, special = c("NA", "NaN"))
count_n(df, special = "all")

# 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")
)
df
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 = everything(), exclude = "name", count = 2)
count_n(df, select = "^score_", regex = TRUE, count = 2)
count_n(df, select = "lang", regex = TRUE, count = "2")
df |> mutate(nb_two = count_n(count = 2))
df |>
  select(score_math, score_science) |>
  mutate(nb_two = count_n(count = 2))
df$nb_two <- count_n(df, select = starts_with("score_"), count = 2)
df[1:3, ] |> count_n(select = starts_with("score_"), count = 2)

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

# 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)

# Using a value from the data: guarantees type and levels match for column x
count_n(df, count = df$x[2], allow_coercion = FALSE)

# Case-insensitive match (factors are converted to character internally)
count_n(df, count = "b", ignore_case = TRUE)
count_n(df, count = "B", ignore_case = TRUE)
```

`cramer_v` *Cramer's V*

Description

`cramer_v()` computes Cramer'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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval.
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.
.include_se	Internal parameter; do not use.

Details

Cramer'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 confidence interval uses the Fisher z-transformation. Standard error formulas follow the DescTools implementations (Signorell et al., 2024).

Value

When detail = FALSE: a single numeric value (the estimate). When detail = TRUE and conf_level is non-NULL: c(estimate, ci_lower, ci_upper, p_value). When detail = TRUE and conf_level = NULL: c(estimate, 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.

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()

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), percentage displays, and inferential statistics.

`cross_tab()` produces weighted or unweighted contingency tables with row or column percentages, optional grouping via `by`, and associated Chi-squared tests with an association measure and diagnostic information.

Both `x` and `y` variables are required. For one-way frequency tables, use `freq()` instead.

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,
  styled = TRUE,
  show_n = TRUE
)

## S3 method for class 'spicy_cross_table_list'
print(x, ...)
```

Arguments

`data`	A data frame. Alternatively, a vector when using the vector-based interface.
`x`	Row variable (unquoted).
`y`	Column variable (unquoted). Mandatory; for one-way tables, use `freq()`.
`by`	Optional grouping variable or expression. Can be a single variable or a combination of multiple variables (e.g. `interaction(vs, am)`).
`weights`	Optional numeric weights.

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), "row", "column". Unique abbreviations are accepted (e.g. "n", "r", "c").
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 Cramer'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. Defaults to 1 for percentages, 0 for counts.
styled	Logical. If TRUE (the default), returns a spicy_cross_table object (for formatted printing). If FALSE, returns a plain data.frame.
show_n	Logical. If TRUE (the default), adds marginal N totals when percent != "none".
...	Additional arguments passed to individual print methods.

Value

A data.frame, list of data.frames, or spicy_cross_table object. When by is used, returns a spicy_cross_table_list.

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.

These options are convenient for users who wish to enforce consistent behavior across multiple calls to cross_tab() and other spicy table functions. 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)
```

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)
```

---

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.

When `styled = TRUE`, the function prints a spicy-formatted ASCII table using `print.spicy_freq_table()` and `spicy_print_table()`; otherwise, it returns a `data.frame` containing frequencies and proportions.

Usage

```
freq(
  data,
  x = NULL,
  weights = NULL,
  digits = 1,
  valid = TRUE,
  cum = FALSE,
  sort = "",
  na_val = NULL,
  labelled_levels = c("prefixed", "labels", "values", "p", "l", "v"),
  rescale = TRUE,
```

```
  styled = TRUE,
  ...
)
```

Arguments

`data`	A `data.frame`, vector, or factor. If a data frame is provided, specify the target variable `x`.
`x`	A variable from `data` (unquoted).
`weights`	Optional numeric vector of weights (same length as `x`). The variable may be referenced as a bare name when it belongs to `data`.
`digits`	Number of decimal digits to display for percentages (default: `1`).
`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: • `""` - no sorting (default) • `"+"` - increasing frequency • `"-"` - decreasing frequency • `"name+"` - alphabetical A-Z • `"name-"` - alphabetical Z-A

`na_val`	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
`rescale`	Logical. If `TRUE` (default), rescale weights so that their total equals the unweighted sample size.
`styled`	Logical. If `TRUE` (default), print the formatted spicy table. If `FALSE`, return a plain `data.frame` with frequency values.
`...`	Additional arguments passed to `print.spicy_freq_table()`.

Details

This function is designed to mimic common frequency procedures from statistical software such as SPSS or Stata, while integrating the flexibility of R's data structures.

It automatically detects the type of input (`vector`, `factor`, or `labelled`) and applies appropriate transformations, including:

- Handling of labelled variables via **labelled** or **haven**
- Optional recoding of specific values as missing (`na_val`)
- Optional weighting with a rescaling mechanism
- Support for cumulative percentages (`cum = TRUE`)
- Multiple display modes for labels via `labelled_levels`

When weighting is applied (`weights`), the frequencies and percentages are computed proportionally to the weights. The argument `rescale = TRUE` normalizes weights so their sum equals the unweighted sample size.

Value

A `data.frame` with 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 percentages (if `cum = TRUE`)

If `styled = TRUE`, prints the formatted table to the console and returns it invisibly.

See Also

`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)

# 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 = "-")

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

# Weighted frequencies (normalized)
freq(df, sexe, weights = poids, rescale = TRUE)

# Weighted frequencies (without rescaling)
freq(df, sexe, weights = poids, rescale = FALSE)

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

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

# Non-styled return (for programmatic use)
f <- freq(df, sexe, styled = FALSE)
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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval.
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.
.include_se	Internal parameter; do not use.

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. 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).

See Also

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

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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval.
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.
.include_se	Internal parameter; do not use.

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. 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).

See Also

lambda_gk(), uncertainty_coef(), assoc_measures()

Examples

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

`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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default 3). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

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. 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-b = 0 (Wald z-test).

See Also

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

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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

Details

Stuart's Tau-c is computed as $\tau_c = 2m(C-D)/(n^2(m-1))$, where $m = \min(r, c)$. It is appropriate for rectangular tables and is not restricted to the range $[-1, 1]$ only for square tables. 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).

See Also

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

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*), and assigns the part after sep as a labelled::var_label(). This works even if the label itself contains 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. "name. label"). (by default from LimeSurvey).
sep	Character string used as separator between name and label. Default is ". " (LimeSurvey's default), but any literal string can be used.

Details

This function is especially useful for **LimeSurvey CSV exports** when using *Export results* → *Export format: CSV* → *Headings: Question code & question text*, where column names look like "code. question text". In this case the default separator is ". ".

Value

A base tibble with column names equal to the *names* (before sep) and var_label attributes equal to the *labels* (after sep).

Examples

```
# Example with LimeSurvey-style column names
df <- data.frame(
  "age. Age of respondent" = c(25, 30),
  "score. Total score. Manually computed." = c(12, 14),
  check.names = FALSE
)
```

```
# sep = ". " by default (LimeSurvey)
out <- label_from_names(df)
labelled::var_label(out)

# Example with a 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 = "|")
labelled::var_label(out2)
```

---

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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

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). 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).

See Also

`goodman_kruskal_tau()`, `uncertainty_coef()`, `assoc_measures()`

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 Optional Minimum Valid Values*

Description

`mean_n()` computes row means from a `data.frame` or `matrix`, handling missing values (`NA`s) automatically. Row-wise means are calculated across selected numeric columns, with an optional condition on the minimum number (or proportion) of valid (non-missing) values required for a row to be included. Non-numeric columns are excluded automatically and reported.

Usage

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

Arguments

data	A data.frame or matrix.
select	Columns to include. If regex = FALSE, use tidyselect syntax (default: dplyr::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. If a proportion, it's applied to the number of selected columns. Defaults to NULL (all values must be valid).
digits	Optional number of decimal places to round the result. 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.

Value

A numeric vector of row-wise means.

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, .include_se = FALSE)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

Details

The phi coefficient is $\phi = \sqrt{\chi^2/n}$. It is equivalent to Cramer's V for 2x2 tables and equals the Pearson correlation between the two binary variables. The confidence interval uses the Fisher z-transformation. 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 the null hypothesis of no association (Pearson chi-squared test).

See Also

`cramer_v()`, `yule_q()`, `assoc_measures()`

Examples

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

print.spicy_assoc_detail

Print a detailed association measure result

Description

Formats a `spicy_assoc_detail` vector (returned by association functions with `detail = TRUE`) with fixed decimal places and `< 0.001` notation for small p-values.

Usage

```
## S3 method for class 'spicy_assoc_detail'
print(x, digits = attr(x, "digits") %||% 3L, ...)
```

Arguments

`x`	A `spicy_assoc_detail` object.
`digits`	Number of decimal places for the estimate, SE, and confidence interval. Defaults to 3. The p-value is always formatted separately (`< 0.001` or three decimal places).
`...`	Ignored.

Value

`x`, invisibly.

See Also

`cramer_v()`, `assoc_measures()`

print.spicy_assoc_table

Print an association measures summary table

Description

Formats a `spicy_assoc_table` data frame (returned by `assoc_measures()`) with fixed decimal places, aligned columns, and `< 0.001` notation for small p-values.

Usage

```
## S3 method for class 'spicy_assoc_table'
print(x, digits = attr(x, "digits") %||% 3L, ...)
```

Arguments

x	A spicy_assoc_table object.
digits	Number of decimal places for estimates, SE, and confidence intervals. Defaults to 3. The p-value is always formatted separately (< 0.001 or three decimal places).
...	Ignored.

Value

x, invisibly.

See Also

assoc_measures()

---

print.spicy_cross_table

Print method for spicy_cross_table objects

Description

Prints a formatted SPSS-like crosstable created by cross_tab().

Usage

```
## S3 method for class 'spicy_cross_table'
print(x, digits = NULL, ...)
```

Arguments

x	A spicy_cross_table object.
digits	Optional integer; number of decimal places to display. Defaults to the value stored in the object.
...	Additional arguments passed to internal formatting functions.

`print.spicy_freq_table` *Styled print method for* `freq()` *tables*

Description

Internal print method used by `freq()` to display a styled, spicy-formatted frequency table in the console. It formats valid, missing, and total rows; handles cumulative and valid percentages; and appends a labeled footer including metadata such as variable label, class, dataset name, and weighting information.

Usage

```
## S3 method for class 'spicy_freq_table'
print(x, ...)
```

Arguments

`x`	A `data.frame` returned by `freq()` with attached attributes: • `"digits"`: number of decimal digits to display • `"data_name"`: name of the source dataset • `"var_name"`: name of the variable • `"var_label"`: variable label, if defined • `"class_name"`: original class of the variable • `"weighted"`, `"rescaled"`, `"weight_var"`: weighting metadata
`...`	Additional arguments (ignored, required for S3 method compatibility)

Details

This function is part of the *spicy table rendering engine*. It is automatically called when printing the result of `freq()` with `styled = TRUE`. The output uses `spicy_print_table()` internally to render a colorized ASCII table with consistent alignment and separators.

The printed table includes:

- Valid and missing value sections (if applicable)
- Optional cumulative and valid percentages
- A final 'Total' row shown in the **Category** column
- A footer summarizing metadata (variable label, data source, weights)

Value

Invisibly returns `x` after printing the formatted table.

Output structure

The printed table includes the following columns:

- **Category**: Sections such as "Valid", "Missing", and "Total"
- **Values**: Observed categories or levels
- **Freq.**: Frequency count (weighted if applicable)
- **Percent**: Percentage of total
- **Valid Percent**: Percentage among valid values (optional)
- **Cum. Percent**: Cumulative percentage (optional)
- **Cum. Valid Percent**: Cumulative valid percentage (optional)

See Also

`freq()` for the main frequency table generator. `spicy_print_table()` for the generic ASCII table renderer.

Examples

```
# Example using labelled data
library(labelled)
x <- labelled(
  c(1, 2, 3, 1, 2, 3, 1, 2, NA),
  labels = c("Low" = 1, "Medium" = 2, "High" = 3)
)
var_label(x) <- "Satisfaction level"
# Capture result without printing, then print explicitly
df <- spicy::freq(x, styled = FALSE)
print(df) # dispatches to print.spicy_freq_table()
```

---

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, and APA-style reporting.

Usage

```
sochealth
```

Format

A tibble with 1200 rows and 24 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.

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).

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. 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.

Details

All variables carry labels (accessible via `labelled::var_label()` and displayed by `varlist()`). Several ordered factors are included so that `cross_tab()` can demonstrate automatic ordinal measure selection.

Source

Simulated data for illustration purposes.

Examples

```
data(sochealth)
varlist(sochealth)
freq(sochealth, education)
cross_tab(sochealth, education, self_rated_health)
```

`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,
  .include_se = FALSE
)
```

Arguments

`x`	A contingency table (of class `table`).
`direction`	Direction of prediction: `"row"` (default, column predicts row), `"column"` (row predicts column), or `"symmetric"` (average 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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

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 is the harmonic mean of the two asymmetric values. 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: D = 0 (Wald z-test).

See Also

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

Examples

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

---

spicy_print_table *Print a spicy-formatted ASCII table*

Description

User-facing helper that prints a visually aligned, spicy-styled ASCII table created by functions such as `freq()` or `cross_tab()`. It automatically adjusts column alignment, spacing, and separators for improved readability in console outputs.

This function wraps the internal renderer `build_ascii_table()`, adding optional titles, notes, and automatic alignment rules depending on the type of table.

Usage

```
spicy_print_table(
  x,
  title = attr(x, "title"),
  note = attr(x, "note"),
  padding = c("compact", "normal", "wide"),
  first_column_line = TRUE,
  row_total_line = TRUE,
  column_total_line = TRUE,
  bottom_line = FALSE,
  lines_color = "darkgrey",
  align_left_cols = NULL,
  ...
)
```

Arguments

x	A spicy_table or data.frame to be printed.
title	Optional title displayed above the table. Defaults to the "title" attribute of x if present.
note	Optional note displayed below the table. Defaults to the "note" attribute of x if present.
padding	Character string controlling horizontal spacing between columns: • "compact" - minimal spacing • "normal" - moderate spacing (default) • "wide" - extra spacing (for wide displays)
first_column_line	Logical. If TRUE (the default), adds a vertical separator after the first column.
row_total_line, column_total_line, bottom_line	Logical flags controlling the presence of horizontal lines before total rows/columns or at the bottom of the table. Both row_total_line and column_total_line default to TRUE; bottom_line defaults to FALSE.
lines_color	Character. Color for table separators. Defaults to "darkgrey". Only applied if the output supports ANSI colors (see crayon::has_color()).
align_left_cols	Integer vector of column indices to left-align. If NULL (the default), alignment is auto-detected based on x: • For freq tables -> c(1, 2) • For cross tables -> 1
...	Additional arguments passed to build_ascii_table().

Details

spicy_print_table() detects whether the table represents frequencies (freq-style) or cross-tabulations (cross-style) and adjusts formatting accordingly:

- For **frequency tables**, the first two columns (*Category* and *Values*) are left-aligned.
- For **cross tables**, only the first column (row variable) is left-aligned.

The function supports Unicode line-drawing characters and colored separators using the **crayon** package, with graceful fallback to monochrome output when color is not supported.

Value

Invisibly returns x, after printing the formatted ASCII table to the console.

See Also

build_ascii_table() for the underlying text rendering engine. print.spicy_freq_table() for the specialized printing method used by freq().

Examples

```
# Simple demonstration
df <- data.frame(
  Category = c("Valid", "", "Missing", "Total"),
  Values = c("Yes", "No", "NA", ""),
  Freq. = c(12, 8, 1, 21),
  Percent = c(57.1, 38.1, 4.8, 100.0)
)

spicy_print_table(df,
  title = "Frequency table: Example",
  note = "Class: data.frame\nData: demo"
)
```

`spicy_tables` *Spicy Table Engine: Frequency and Cross-tabulation Rendering*

Description

The *spicy table engine* provides a cohesive set of tools for creating and printing formatted ASCII tables in R, designed for descriptive statistics.

Functions in this family include:

- `freq()` — frequency tables with support for weights, labelled data, and cumulative percentages
- `spicy_print_table()` — general-purpose ASCII table printer
- `build_ascii_table()` — internal rendering engine for column alignment and formatting

Details

All functions in this family share a common philosophy:

- Console-friendly display with Unicode box-drawing characters
- Consistent alignment and spacing across outputs
- Automatic detection of variable type (`factor`, `labelled`, `numeric`)
- Optional integration of variable labels and weighting information

Core functions

- `freq()` — Main entry point for generating frequency tables.
- `spicy_print_table()` — Applies formatting and optional titles or notes.
- `build_ascii_table()` — Internal engine handling padding, alignment, and box rules.

Output styling

The spicy table engine supports multiple padding options via `padding`: `"compact"` (default), `"normal"`, and `"wide"`. Horizontal and vertical rules can be customized, and colors are supported when the terminal allows ANSI color output (via the **crayon** package).

See Also

`print.spicy_freq_table()` for the specialized frequency display method. `labelled::to_factor()` and `dplyr::pull()` for data transformations.

---

sum_n *Row Sums with Optional Minimum Valid Values*

Description

`sum_n()` computes row sums from a `data.frame` or `matrix`, handling missing values (`NA`s) automatically. Row-wise sums are calculated across selected numeric columns, with an optional condition on the minimum number (or proportion) of valid (non-missing) values required for a row to be included. Non-numeric columns are excluded automatically and reported.

Usage

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

Arguments

`data`	A `data.frame` or `matrix`.
`select`	Columns to include. If `regex = FALSE`, use tidyselect syntax (default: `dplyr::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. If a proportion, it's applied to the number of selected columns. Defaults to `NULL` (all values must be valid).
`digits`	Optional number of decimal places to round the result. 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.

Value

A numeric vector of row-wise sums

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_apa` *Build APA-Style Cross-Tabulation Tables*

Description

`table_apa()` builds a publication-ready table by crossing one grouping variable (`group_var`) with one or many row variables (`row_vars`), using `spicy::cross_tab()` internally.

Usage

```
table_apa(
  data,
```

```
  row_vars,
  group_var,
  labels = NULL,
  levels_keep = NULL,
  include_total = TRUE,
  drop_na = TRUE,
  weights = NULL,
  rescale = FALSE,
  correct = FALSE,
  simulate_p = FALSE,
  simulate_B = 2000,
  percent_digits = 1,
  p_digits = 3,
  v_digits = 2,
  assoc_measure = "auto",
  assoc_ci = FALSE,
  decimal_mark = ".",
 output = c("wide", "long", "tinytable", "gt", "flextable", "excel", "clipboard",
    "word"),
  style = c("auto", "raw", "report"),
  indent_text = "  ",
  indent_text_excel_clipboard = "      ",
  add_multilevel_header = TRUE,
  blank_na_wide = FALSE,
  excel_path = NULL,
  excel_sheet = "APA",
  clipboard_delim = "\t",
  word_path = NULL
)
```

Arguments

`data`	A data frame.
`row_vars`	Character vector of variable names to place in rows.
`group_var`	Single character variable name used for columns/groups.
`labels`	Optional character labels for `row_vars` (same length).
`levels_keep`	Optional character vector of levels to keep/order for row modalities. If `NULL`, all observed levels are kept.
`include_total`	Logical. If `TRUE` (the default), includes a `Total` group when available.
`drop_na`	Logical. If `TRUE` (the default), removes rows with `NA` in the row/group variable before each cross-tabulation. If `FALSE`, missing values are displayed as a dedicated `"(Missing)"` level.
`weights`	Optional weights. Either `NULL` (the default), a numeric vector of length `nrow(data)`, or a single column name in `data`.
`rescale`	Logical. If `FALSE` (the default), weights are used as-is. If `TRUE`, rescales weights so total weighted N matches raw N. Passed to `spicy::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().
simulate_p	Logical. If FALSE (the default), uses asymptotic p-values. If TRUE, uses Monte Carlo simulation. Passed to spicy::cross_tab().
simulate_B	Integer. Number of Monte Carlo replicates when simulate_p = TRUE. Defaults to 2000.
percent_digits	Number of digits for percentages in report outputs. Defaults to 1.
p_digits	Number of digits for p-values (except < .001). Defaults to 3.
v_digits	Number of digits for the association measure. Defaults to 2.
assoc_measure	Passed to cross_tab(). Which association measure to report ("auto", "cramer_v", "phi", "gamma", "tau_b", "tau_c", "somers_d", "lambda", "none"). Defaults to "auto".
assoc_ci	Passed to cross_tab(). If TRUE, includes the confidence interval. In data/export formats (wide, long, excel, clipboard), two extra columns CI lower and CI upper are added. In rendered formats (gt, tinytable, flextable, word), the CI is shown inline as .14 [.08, .19] in the association measure column. Defaults to FALSE.
decimal_mark	Decimal separator (". " or ", "). Defaults to ". ".
output	Output format: "wide" (the default), "long", "tinytable", "gt", "flextable", "excel", "clipboard", "word".
style	"auto" (the default) to select by output type, "raw" for machine-friendly outputs, "report" for formatted outputs.
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.
blank_na_wide	Logical. If FALSE (the default), NA values are kept as-is in wide raw output. If TRUE, replaces them with empty strings.
excel_path	Path for output = "excel". Defaults to NULL.
excel_sheet	Sheet name for Excel export. Defaults to "APA".
clipboard_delim	Delimiter for clipboard text export. Defaults to "\t".
word_path	Path for output = "word" or optional save path when output = "flextable". Defaults to NULL.

Details

It supports raw data outputs (wide, long) and report-oriented outputs (tinytable, flextable, excel, clipboard, word) with multi-level headers, p-values, and an association measure.

Optional output engines require suggested packages:

- tinytable for output = "tinytable"
- gt for output = "gt"
- flextable + officer for output = "flextable"/"word"
- openxlsx for output = "excel"
- clipr for output = "clipboard"

Value

Depends on output and style:

- "long" + "raw": long numeric data frame.
- "wide" + "raw": wide numeric data frame.
- "long" + "report": long formatted character data frame.
- "wide" + "report": wide formatted character data frame.
- "tinytable": a tinytable object.
- "gt": a gt_tbl object.
- "flextable": a flextable object.
- "excel" / "clipboard" / "word": invisibly returns written object/path.

Examples

```
# Raw long output (machine-friendly)
table_apa(
  data = sochealth,
  row_vars = c("smoking", "physical_activity"),
  group_var = "education",
  labels = c("Current smoker", "Physical activity"),
  output = "long",
  style = "raw"
)

# Raw wide output
table_apa(
  data = sochealth,
  row_vars = c("smoking", "physical_activity"),
  group_var = "education",
  labels = c("Current smoker", "Physical activity"),
  output = "wide",
  style = "raw"
)

# Weighted example
table_apa(
  data = sochealth,
  row_vars = c("smoking", "physical_activity"),
  group_var = "education",
  labels = c("Current smoker", "Physical activity"),
  weights = "weight",
```

```
  rescale = TRUE,
  simulate_p = FALSE,
  output = "long",
  style = "raw"
)


# Optional output: tinytable
if (requireNamespace("tinytable", quietly = TRUE)) {
  table_apa(
    data = sochealth,
    row_vars = c("smoking", "physical_activity"),
    group_var = "education",
    labels = c("Current smoker", "Physical activity"),
    output = "tinytable"
  )
}

# Optional output: Excel
if (requireNamespace("openxlsx", quietly = TRUE)) {
  table_apa(
    data = sochealth,
    row_vars = c("smoking", "physical_activity"),
    group_var = "education",
    labels = c("Current smoker", "Physical activity"),
    output = "excel",
    excel_path = tempfile(fileext = ".xlsx")
  )
}
```

`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,
  .include_se = FALSE
)
```

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 number between 0 and 1 giving the confidence level (default 0.95). Only used when detail = TRUE. Set to NULL to omit the confidence interval.
digits	Number of decimal places used when printing the result (default 3). Only affects the detail = TRUE output.
.include_se	Internal parameter; do not use.

Details

The uncertainty coefficient measures association using Shannon entropy. For direction = "row": $U = (H_X + H_Y - H_{XY})/H_X$, where H_X, H_Y are the marginal entropies and H_{XY} is the joint entropy. The symmetric version is $U = 2(H_X + H_Y - H_{XY})/(H_X + H_Y)$. 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: U = 0 (Wald z-test).

See Also

lambda_gk(), goodman_kruskal_tau(), assoc_measures()

Examples

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

---

varlist *Generate a comprehensive summary of the variables*

Description

varlist() lists the variables of a data frame and extracts essential metadata, including variable names, labels, summary values, classes, number of distinct values, number of valid (non-missing) observations, and number of missing values.

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,
  .raw_expr = substitute(x)
)

vl(x, ..., values = FALSE, tbl = FALSE, include_na = FALSE)
```

Arguments

x	A data frame or a transformation of one. Must be named and identifiable.
...	Optional tidyselect-style column selectors (e.g. `starts_with("var")`, `where(is.numeric)`, etc.).
values	Logical. If `FALSE` (the default), only min/max or representative values are displayed. If `TRUE`, all unique values are listed.
tbl	Logical. If `FALSE` (the default), the summary is opened in the Viewer (if interactive). If `TRUE`, a tibble is returned instead.
include_na	Logical. If `TRUE`, missing values (`NA`) are included in the `Values` column. Default is `FALSE`.
.raw_expr	Internal. Do not use. Captures the original expression from `vl()` to generate an informative title. Used only for internal purposes.

Details

The function can also apply tidyselect-style variable selectors to filter columns dynamically.

If used interactively (e.g. in RStudio), the summary is displayed in the Viewer pane with a contextual title like `vl: sochealth`. If the data frame has been transformed or subsetted, the title will display an asterisk (`*`), e.g. `vl: sochealth*`.

For full documentation, see `varlist()`.

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 (max 4 values: 3 + ... + last) is shown. 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 used as-is. Missing values (`NA`, `NaN`) are optionally appended at the end (controlled via `include_na`).

- `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 If `tbl = FALSE` and used interactively, the summary is displayed in the Viewer pane. If the data frame is a transformation (e.g. `head(df)` or `df[ , 1:3]`), an asterisk (`*`) is appended to the name in the title (e.g. `vl: df*`).

Examples

```
varlist(sochealth)
sochealth |> varlist()
varlist(sochealth, where(is.numeric), values = TRUE, tbl = TRUE)
varlist(sochealth, tbl = TRUE)
varlist(sochealth, starts_with("bmi"), tbl = TRUE)

vl(sochealth)
sochealth |> vl()
vl(sochealth, starts_with("bmi"))
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, .include_se = FALSE)
```

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 number between 0 and 1 giving the confidence level (default `0.95`). Only used when `detail = TRUE`. Set to `NULL` to omit the confidence interval.
`digits`	Number of decimal places used when printing the result (default `3`). Only affects the `detail = TRUE` output.
`.include_se`	Internal parameter; do not use.

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 $SE = 0.5(1 - Q^2)\sqrt{1/a + 1/b + 1/c + 1/d}$. 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).

See Also

`phi()`, `gamma_gk()`, `assoc_measures()`

Examples

```
tab <- table(sochealth$smoking, sochealth$sex)
yule_q(tab)
```

Index

∗ **datasets**
    sochealth, 33
∗ **descriptive**
    spicy_tables, 38
∗ **frequency**
    spicy_tables, 38
∗ **spicy tables**
    spicy_tables, 38
∗ **spicy**
    spicy_tables, 38
∗ **tables**
    spicy_tables, 38

assoc_measures, 2
assoc_measures(), *6*, *13*, *20–22*, *24*, *26*, *29–31*, *36*, *46*, *49*

build_ascii_table(), *36–38*

clipr::write_clip(), *7*
code_book, 4
contingency_coef, 5
contingency_coef(), *3*, *13*
copy_clipboard, 6
count_n, 8
cramer_v, 12
cramer_v(), *3*, *6*, *20–23*, *26*, *29, 30*, *35, 36*, *46*, *48, 49*
crayon::has_color(), *37*
cross_tab, 14
cross_tab(), *31*, *34*, *36*, *43*

datawizard::row_count(), *9, 10*
dplyr::pull(), *39*

freq, 16
freq(), *14*, *32, 33*, *36–38*

gamma_gk, 19
gamma_gk(), *3*, *22*, *24*, *36*, *49*
goodman_kruskal_tau, 20
goodman_kruskal_tau(), *3*, *26*, *46*

kendall_tau_b, 22
kendall_tau_b(), *3*, *20*, *24*, *36*
kendall_tau_c, 23
kendall_tau_c(), *3*, *20*, *22*

label_from_names, 24
labelled::to_factor(), *39*
labelled::var_label(), *24*, *34*
lambda_gk, 25
lambda_gk(), *3*, *21*, *46*

mean_n, 26

phi, 29
phi(), *3*, *13*, *49*
print.spicy_assoc_detail, 30
print.spicy_assoc_table, 30
print.spicy_cross_table, 31
print.spicy_cross_table_list *(*cross_tab*)*, 14
print.spicy_freq_table, 32
print.spicy_freq_table(), *16–18*, *37*, *39*

sochealth, 33
somers_d, 35
somers_d(), *3*, *20*, *22*, *24*
spicy_print_table, 36
spicy_print_table(), *16*, *18*, *32, 33*, *38*
spicy_tables, 38
sum_n, 39

table_apa, 41
tidyselect::starts_with(), *8*

uncertainty_coef, 45
uncertainty_coef(), *3*, *21*, *26*

varlist, 46
varlist(), *4, 5*, *34*, *47*

vl *(*varlist*)*, 46

yule_q, 48
yule_q(), *3*, *29*