Package ‘spicy’

May 5, 2025

Title Descriptive Statistics and Data Management Tools

Version 0.1.0

Description Extracts and summarizes metadata from data frames, including variable names, labels, types, and missing values. Computes compact descriptive statistics, frequency tables, and cross-tabulations to assist with efficient data exploration. Facilitates the identification of missing data patterns and structural issues in datasets. Designed to streamline initial data management and exploratory analysis workflows within 'R'.

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 7.3.2

Imports clipr, collapse, dplyr, haven, labelled, rlang, stats, stringi, stringr, tibble, tidyselect, utils

Suggests testthat (>= 3.0.0)

Depends R (>= 4.1.0)

Config/testthat/edition 3

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2025-05-05 08:10:02 UTC

Contents

`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,
  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` (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` (default), includes row names in the clipboard output. If `FALSE`, row names are omitted.
`col.names`	Logical. If `TRUE` (default), includes column names in the clipboard output. If `FALSE`, column names are omitted.

`message`	Logical. If `TRUE` (default), displays a success message after copying. If `FALSE`, no success message is printed.
`quiet`	Logical. If `TRUE`, suppresses all messages, including success, coercion notices, and warnings. If `FALSE` (default), messages are shown.
`...`	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(mtcars)

  # Data frame with row names as column
  copy_clipboard(mtcars, row.names.as.col = "car")

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

  # Table
  tbl <- table(iris$Species)
  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(mtcars, 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. Uses tidyselect helpers like `tidyselect::everything()`, `tidyselect::starts_with()`, etc. If `regex = TRUE`, `select` is treated as a regex string.
`exclude`	Character vector of column names to exclude after selection.
`count`	Value(s) to count. 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"`. `"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 (default `TRUE`). If `FALSE`, uses strict matching via `identical()`.
`ignore_case`	Logical (default `FALSE`). If `TRUE`, performs case-insensitive string comparisons.
`regex`	Logical (default `FALSE`). If `TRUE`, interprets `select` as a regular expression pattern.
`verbose`	Logical (default `FALSE`). 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` *Compute Cramer's V*

Description

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

Usage

```
cramer_v(x)
```

Arguments

`x`	A contingency table (of class `table`) for which to compute the statistic.

Details

Cramer's V is based on the chi-squared statistic and adjusts for the size of the table. It is suitable for nominal (unordered categorical) variables.

Value

A numeric vector of length 1, representing the Cramer's V statistic.

Examples

```
# Example with mtcars dataset
data(mtcars)

# Discretize continuous variables
mtcars$gear <- as.factor(mtcars$gear)
mtcars$cyl <- as.factor(mtcars$cyl)

# Create contingency table
tab <- table(mtcars$gear, mtcars$cyl)

# Compute Cramer's V
cramer_v(tab)
```

cross_tab — *Cross-Tabulation with Percentages, Weights, and Grouping*

Description

`cross_tab()` produces a cross-tabulation of `x` by `y`, with optional stratification using a grouping variable (`by`). It supports weighted frequencies, row or column percentages, and association statistics (Chi-squared test, Cramer's V).

Usage

```
cross_tab(
  d = parent.frame(),
  x,
  y = NULL,
  by = NULL,
  weights = NULL,
  rescale_weights = FALSE,
  digits = 1,
  rowprct = FALSE,
  row_total = TRUE,
  column_total = TRUE,
  n = TRUE,
  drop = TRUE,
  include_stats = TRUE,
  combine = FALSE,
  ...
)
```

Arguments

`d`	A `data.frame`, or a vector (when using vector input). Must contain all variables used in `x`, `y`, `by`, and `weights`.
`x`	Variable for table rows. Can be unquoted (tidy) or quoted (standard). Must match column name if `d` is a data frame.
`y`	Optional variable for table columns. Same rules as `x`. If `NULL`, computes a one-way frequency table.
`by`	Optional **grouping variable** (or interaction of variables). Used to produce stratified crosstabs. Must refer to columns in `d`, or be a vector of the same length as `x`.
`weights`	Optional numeric vector of weights. Must match length of `x`.
`rescale_weights`	Logical. If `TRUE`, rescales weights so that total weighted count matches unweighted count.
`digits`	Integer. Number of decimal places shown in percentages. Default is `1`.

`rowprct`	Logical. If `TRUE`, computes percentages by row; otherwise by column.
`row_total`	Logical. If `TRUE`, adds row totals (default `TRUE`).
`column_total`	Logical. If `TRUE`, adds column totals (default `TRUE`).
`n`	Logical. If `TRUE`, displays effective counts `N` as an extra row or column (default `TRUE`).
`drop`	Logical. If `TRUE`, drops empty rows or columns (default `TRUE`).
`include_stats`	Logical. If `TRUE`, includes Chi-squared test and Cramer's V when possible (default `TRUE`).
`combine`	Logical. If `TRUE`, combines all stratified tables into one tibble with a `by` column.
`...`	Additional arguments passed to `print.spicy()`, such as `show_all = TRUE`

Details

The function is flexible:

- Accepts both **standard** (quoted) and **tidy** (unquoted) variable input
- Performs stratified tabulations using a **grouping variable** (`by`)
- Optionally combines group-level tables into a single tibble with `combine = TRUE`
- Pipe-friendly with both base R (`|>`) and magrittr (`%>%`)

All variables (`x`, `y`, `by`, `weights`) must be present in the data frame `d` (unless vector input is used).

Value

A tibble of class `spicy`, or a list of such tibbles if `combine = FALSE` and `by` is used.

Warnings and Errors

- If `weights` is non-numeric, an error is thrown.
- If `weights` does not match the number of observations, an error is thrown.
- If `rescale_weights = TRUE` but no weights are provided, a warning is issued.
- If all values in `by` are `NA`, an error is thrown.
- If `by` has only one unique level (or all `NA`), a warning is issued.

Examples

```
data(mtcars)
mtcars$gear <- factor(mtcars$gear)
mtcars$cyl <- factor(mtcars$cyl)
mtcars$vs <- factor(mtcars$vs, labels = c("V", "S"))
mtcars$am <- factor(mtcars$am, labels = c("auto", "manual"))

# Basic usage
cross_tab(mtcars, cyl, gear)

# Using extracted variables
cross_tab(mtcars$cyl, mtcars$gear)
```

```
# Pipe-friendly syntax
mtcars |> cross_tab(cyl, gear, by = am)

# With row percentages
cross_tab(mtcars, cyl, gear, by = am, rowprct = TRUE)

# Using weights
cross_tab(mtcars, cyl, gear, weights = mpg)

# With rescaled weights
cross_tab(mtcars, cyl, gear, weights = mpg, rescale_weights = TRUE)

# Grouped by a single variable
cross_tab(mtcars, cyl, gear, by = am)

# Grouped by interaction of two variables
cross_tab(mtcars, cyl, gear, by = interaction(am, vs), combine = TRUE)

# Combined output for grouped data
cross_tab(mtcars, cyl, gear, by = am, combine = TRUE)

# Without totals or sample size
cross_tab(mtcars, cyl, gear, row_total = FALSE, column_total = FALSE, n = FALSE)
```

`freq` *Frequency Table*

Description

`freq()` creates a frequency table for a variable or vector, with options for weighting, sorting, handling missing values, and calculating percentages.

Usage

```
freq(
  data,
  x = NULL,
  weights = NULL,
  digits = 1,
  cum = FALSE,
  total = TRUE,
  exclude = NULL,
  sort = "",
  valid = TRUE,
  na_val = NULL,
  rescale_weights = FALSE,
  info = TRUE,
  labelled_levels = c("prefixed", "labels", "values"),
  styled = TRUE,
  show_empty_levels = FALSE,
  ...
)
```

Arguments

`data`	A `data.frame`, vector or factor. If a `data.frame` is provided, the target variable `x` must be specified. Matrices are not supported; please extract a column or convert to a vector or tibble before use.
`x`	A dataframe variable.
`weights`	A numeric vector of weights. Must be the same length as `x`.
`digits`	Numeric. Number of digits to be displayed for percentages. Default is `1`. For N, 2 digits are displayed if there is a weight variable with non-integer weights or if rescale_weight = T, otherwise 0.
`cum`	Logical. If `FALSE` (the default), do not display cumulative percentages. If `TRUE`, display cumulative percentages.
`total`	Logical. If `TRUE` (the default), add a final row of totals. If `FALSE`, remove a final row of totals.
`exclude`	Values to exclude (e.g., `NA`, "Other"). Default is `NULL`.
`sort`	Sorting method for values: • `""` (default): No specific sorting. • `"+"`: Sort by increasing frequency. • `"-"`: Sort by decreasing frequency. • `"name+"`: Sort alphabetically (A-Z). • `"name-"`: Sort alphabetically (Z-A).
`valid`	Logical. If `TRUE` (the default), display valid percentages (excluding missing values). If `FALSE`, do not display valid percentages.
`na_val`	Character or numeric. For factors, character or numeric vectors, values to be treated as `NA`.
`rescale_weights`	Logical. If `FALSE` (the default), do not rescale weights. If `TRUE`, the total count will be the same as the unweighted `x`.
`info`	Logical. If `TRUE` (the default), print a title and a note (label and class of `x`, variable weight, dataframe name) information about the model (model formula, number of observations, residual standard deviation and more).
`labelled_levels`	For `labelled` variables, controls how values are displayed using `labelled::to_factor(levels = "prefixed")`: • `"prefixed"` or `"p"` (default): Show labels as `[value] label` • `"labels"` or `"l"`: Show only the label • `"values"` or `"v"`: Show only the underlying value

`styled`	Logical. If `TRUE` (default), formats the output using `print.spicy()`, which aligns columns dynamically in a structured three-line table. If `FALSE`, returns a standard `data.frame` without formatting.
`show_empty_levels`	Logical. If `FALSE` (default), factor levels with `N = 0` are removed from the output. Set to `TRUE` to retain all levels, even those with no observations.
`...`	Additional arguments passed to `print.spicy()`, such as `show_all = TRUE`

Value

A formatted `data.frame` containing unique values of `x`, their frequencies (`N`), percentages (`%`), percentages of valid values (`Valid%`), with a "Total" row.

- If `cum = TRUE`, cumulative frequencies (`%cum` and `Valid%cum`) are included.

Examples

```
data(iris)
data(mtcars)
freq(iris, Species)
iris |> freq(Species, cum = TRUE)
freq(mtcars, cyl, sort = "-", cum = TRUE)
freq(mtcars, gear, weights = mpg, rescale_weights = TRUE)

# With labelled variable
library(labelled)
df <- data.frame(
var1 = set_variable_labels(1:5, label = "Numeric Variable with Label"),
var2 = labelled(1:5, c("Low" = 1, "Medium" = 2, "High" = 3)),
var3 = set_variable_labels(
labelled(1:5, c("Bad" = 1, "Average" = 2, "Good" = 3)),
label = "Labelled Variable with Label"))
df |> freq(var2)
df |> freq(var2,labelled_levels = "l")
df |> freq(var2,labelled_levels = "v")
df |> freq(var3)
df |> freq(var3,labelled_levels = "v")
df |> freq(var3,labelled_levels = "l")
```

`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.
`digits`	Optional number of decimal places to round the result.
`regex`	If `TRUE`, the `select` argument is treated as a regular expression. If `FALSE`, uses tidyselect helpers.
`verbose`	If `TRUE`, prints a message about processing.

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)
```

`print.spicy` *Print a Formatted Data Frame with Aligned Columns*

Description

`print.spicy()` prints a data frame with properly aligned columns, following a structured three-line table format. The first column is left-aligned, while all other columns are right-aligned. Column widths are dynamically adjusted based on the longest value in each column, including column names.

Usage

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

Arguments

`x`	A data frame, matrix, array (2D), or table.
`...`	Additional arguments (not used).

Value

Invisibly returns `x` after displaying its formatted content.

Examples

```
x <- mtcars
class(x) <- c("spicy", class(x))
print(x)
```

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.
`digits`	Optional number of decimal places to round the result.
`regex`	If `TRUE`, the `select` argument is treated as a regular expression. If `FALSE`, uses tidyselect helpers.
`verbose`	If `TRUE`, prints a message about processing.

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)
```

`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 `VARLIST iris`. If the data frame has been transformed or subsetted, the title will display an asterisk (`*`), e.g. `VARLIST iris*`.

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"`)
- `Ndist_val`: 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. `VARLIST df*`).

Examples

```
varlist(iris)
iris |> varlist()
iris |> varlist(starts_with("Sepal"), tbl = TRUE)
```

```
varlist(mtcars, where(is.numeric), values = TRUE, tbl = TRUE)
varlist(head(mtcars), tbl = TRUE)
varlist(mtcars, tbl = TRUE)
varlist(iris[, 1:3], tbl = TRUE)
varlist(mtcars[1:10, ], tbl = TRUE)

vl(iris)
iris |> vl()
vl(mtcars, starts_with("d"))
vl(head(iris), include_na = TRUE)
vl(iris[, 1:3], values = TRUE, tbl = TRUE)
```

Index