Package 'mariposa'

October 22, 2026

Type Package

Title 'SPSS'-Compatible Statistical Tools for Survey Data

Version 0.7.4

Description Statistical analysis of survey data with full support for survey weights, grouped operations, and 'tidyverse' integration. Provides 80 functions for data import/export ('SPSS', 'Stata', 'SAS', 'Excel') with label roundtripping and tagged NA preservation, label management (variable labels, value labels, type conversions, missing value declaration), data transformation (recoding, dummy coding, standardization, centering), descriptive statistics, codebook generation, hypothesis testing, correlation analysis, post-hoc comparisons, weighted statistics, scale analysis, regression, non-parametric tests, exact tests, factorial ANOVA, and ANCOVA. Every analysis offers compact print() and detailed summary() output with toggleable sections. Statistical results are checked against reference output of 'SPSS' version 29 (see the compatibility vignette for the status of each function). Methods follow the published algorithms of IBM Corp. (2023, ``IBM SPSS Statistics Algorithms''), the Lilliefors-corrected normality test of Dallal and Wilkinson (1986) <doi:10.1080/00031305.1986.10475419>, and the adjusted standardized residuals of Haberman (1973) <doi:10.2307/2529686>. Designed for survey researchers, social scientists, and students working with complex survey designs.

License MIT + file LICENSE

Language en

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cli (>= 3.0.0), htmltools (>= 0.5.0), stats, utils, dplyr (>= 1.0.0), rlang (>= 1.0.0), tidyselect (>= 1.1.0), tibble (>= 3.0.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, covr, haven (>= 2.4.0), GPArotation, MASS, openxlsx2, survey, withr, broom, psych, lavaan, semTools, nortest

VignetteBuilder knitr

LazyData true

Config/testthat/edition 3

URL https://YannickDiehl.github.io/mariposa/,
https://github.com/YannickDiehl/mariposa

BugReports https://github.com/YannickDiehl/mariposa/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Yannick Diehl [aut, cre]

Maintainer Yannick Diehl <yannick.diehl@gmail.com>

Repository CRAN

Date/Publication 2026-10-22 09:00:02 UTC

Contents

ancova . . . 6
anova.linear_regression . . . 9
anova.logistic_regression . . . 10
as.data.frame.mariposa . . . 10
binomial_test . . . 14
center . . . 17
chisq_gof . . . 19
chi_square . . . 21
codebook . . . 24
coef.linear_regression . . . 27
confint.linear_regression . . . 27
confint.logistic_regression . . . 28
copy_labels . . . 29
crosstab . . . 30
describe . . . 33
df.residual.linear_regression . . . 35
drop_labels . . . 36
dunn_test . . . 37
efa . . . 40
factorial_anova . . . 44
find_var . . . 47
fisher_test . . . 49
fitted.linear_regression . . . 52
frequency . . . 52
friedman_test . . . 55
kendall_tau . . . 58
kruskal_wallis . . . 61
levene_test . . . 64
linear_regression . . . 67
logistic_regression . . . 71

logistic_regression-accessors . . . 75
longitudinal_data . . . 76
longitudinal_data_wide . . . 78
mann_whitney . . . 79
marginal_effects . . . 83
mcnemar_test . . . 85
multiple_response . . . 87
na_frequencies . . . 89
nobs.linear_regression . . . 90
normality_test . . . 91
oneway_anova . . . 93
pairwise_wilcoxon . . . 96
partial_cor . . . 99
pearson_cor . . . 101
phi . . . 105
pomps . . . 106
predict.linear_regression . . . 107
predict.logistic_regression . . . 108
print.ancova . . . 109
print.binomial_test . . . 109
print.chisq_gof . . . 110
print.chi_square . . . 111
print.codebook . . . 111
print.crosstab . . . 112
print.describe . . . 113
print.dunn_test . . . 113
print.efa . . . 114
print.factorial_anova . . . 115
print.fisher_test . . . 115
print.frequency . . . 116
print.friedman_test . . . 117
print.kendall_tau . . . 117
print.kruskal_wallis . . . 118
print.levene_test . . . 119
print.linear_regression . . . 119
print.logistic_regression . . . 120
print.mann_whitney . . . 121
print.marginal_effects . . . 122
print.mcnemar_test . . . 122
print.multiple_response . . . 123
print.normality_test . . . 124
print.oneway_anova . . . 125
print.pairwise_wilcoxon . . . 125
print.partial_cor . . . 126
print.pearson_cor . . . 127
print.reliability . . . 127
print.scheffe_test . . . 128
print.spearman_rho . . . 129

print.summary.ancova . . . 129
print.summary.binomial_test . . . 130
print.summary.chisq_gof . . . 131
print.summary.chi_square . . . 132
print.summary.codebook . . . 132
print.summary.crosstab . . . 133
print.summary.describe . . . 134
print.summary.dunn_test . . . 135
print.summary.efa . . . 136
print.summary.factorial_anova . . . 136
print.summary.fisher_test . . . 137
print.summary.frequency . . . 138
print.summary.friedman_test . . . 139
print.summary.kendall_tau . . . 139
print.summary.kruskal_wallis . . . 140
print.summary.levene_test . . . 141
print.summary.linear_regression . . . 142
print.summary.logistic_regression . . . 142
print.summary.mann_whitney . . . 143
print.summary.marginal_effects . . . 144
print.summary.mcnemar_test . . . 145
print.summary.multiple_response . . . 146
print.summary.normality_test . . . 146
print.summary.oneway_anova . . . 147
print.summary.pairwise_wilcoxon . . . 148
print.summary.partial_cor . . . 149
print.summary.pearson_cor . . . 150
print.summary.reliability . . . 150
print.summary.scheffe_test . . . 151
print.summary.spearman_rho . . . 152
print.summary.tukey_test . . . 153
print.summary.t_test . . . 154
print.summary.wilcoxon_test . . . 154
print.summary.w_quantile . . . 155
print.summary.w_statistic . . . 156
print.tukey_test . . . 156
print.t_test . . . 157
print.wilcoxon_test . . . 158
print.w_modus . . . 158
profile.logistic_regression . . . 159
read_por . . . 159
read_sas . . . 160
read_spss . . . 162
read_stata . . . 164
read_xlsx . . . 165
read_xpt . . . 167
rec . . . 168
reliability . . . 171

residuals.linear_regression . . . 174
row_count . . . 175
row_means . . . 176
row_sums . . . 178
scheffe_test . . . 179
set_na . . . 182
spearman_rho . . . 184
std . . . 187
strip_tags . . . 189
summary.ancova . . . 190
summary.binomial_test . . . 191
summary.chisq_gof . . . 192
summary.chi_square . . . 193
summary.codebook . . . 194
summary.crosstab . . . 195
summary.describe . . . 196
summary.dunn_test . . . 197
summary.efa . . . 198
summary.factorial_anova . . . 199
summary.fisher_test . . . 200
summary.frequency . . . 201
summary.friedman_test . . . 202
summary.kendall_tau . . . 203
summary.kruskal_wallis . . . 204
summary.levene_test . . . 205
summary.linear_regression . . . 206
summary.logistic_regression . . . 207
summary.mann_whitney . . . 208
summary.marginal_effects . . . 209
summary.mcnemar_test . . . 210
summary.multiple_response . . . 211
summary.normality_test . . . 212
summary.oneway_anova . . . 212
summary.pairwise_wilcoxon . . . 213
summary.partial_cor . . . 214
summary.pearson_cor . . . 215
summary.reliability . . . 216
summary.scheffe_test . . . 217
summary.spearman_rho . . . 218
summary.tukey_test . . . 219
summary.t_test . . . 220
summary.wilcoxon_test . . . 221
survey_data . . . 222
to_character . . . 224
to_dummy . . . 225
to_label . . . 227
to_labelled . . . 229
to_numeric . . . 230

tukey_test . . . 232
t_test . . . 234
unlabel . . . 238
untag_na . . . 240
update.linear_regression . . . 241
update.logistic_regression . . . 241
val_labels . . . 242
var_label . . . 243
vcov.linear_regression . . . 245
wilcoxon_test . . . 246
write_spss . . . 248
write_stata . . . 250
write_xlsx . . . 251
write_xpt . . . 254
w_iqr . . . 256
w_kurtosis . . . 258
w_mean . . . 260
w_median . . . 262
w_modus . . . 264
w_quantile . . . 266
w_range . . . 268
w_sd . . . 270
w_se . . . 272
w_skew . . . 274
w_var . . . 276

Index **279**

`ancova` *Analysis of Covariance: ANCOVA*

Description

`ancova()` tests whether group means differ after controlling for one or more continuous covariates. It performs a factorial ANCOVA using Type III Sum of Squares, matching SPSS UNIANOVA output with the WITH keyword.

Think of it as:

- An ANOVA that removes the effect of confounding variables first
- Testing group differences on "adjusted" means
- Combining regression (covariates) and ANOVA (factors) in one model

The test tells you:

- Whether each factor has a significant effect AFTER controlling for covariates
- The relationship between each covariate and the outcome
- Effect sizes for factors and covariates (partial eta squared)
- Estimated marginal means (group means adjusted for covariates)

Usage

```
ancova(data, dv, between, covariate, weights = NULL, ss_type = 3)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`dv`	The numeric dependent variable to analyze (unquoted)
`between`	Character vector or unquoted variable names specifying the between-subjects factors (1-3 factors). These must be categorical variables (factor, character, or labelled numeric).
`covariate`	Unquoted variable names of the continuous covariates to control for. Use `c(age, income)` for multiple covariates.
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`ss_type`	Deprecated; only 3 (Type III, the SPSS default) is implemented. Passing 2 issues a warning and computes Type III.

Details

Understanding the Results:

Adjusted Means (Estimated Marginal Means): These are the group means after statistically removing the effect of the covariate(s). They answer: "What would the group means be if all groups had the same covariate values?"

Covariate Effects: The covariate row in the ANOVA table shows whether the covariate significantly predicts the DV after adjusting for the factors.

Factor Effects: These show whether the factor affects the DV after controlling for the covariate. This is the primary test of interest.

Partial Eta Squared (Effect Size):

- Less than 0.01: Negligible
- 0.01 to 0.06: Small
- 0.06 to 0.14: Medium
- 0.14 or greater: Large

When to Use This:

Use ANCOVA when:

- You have group comparisons (ANOVA) but want to control for a confound
- Your covariate is continuous and linearly related to the DV
- You want to increase statistical power by removing known variance sources
- The covariate's relationship with the DV is the same across groups (homogeneity of regression slopes assumption)

Value

An object of class "ancova" containing:

anova_table Tibble with Source, SS, df, MS, F, p, Partial Eta Squared

parameter_estimates Tibble with regression coefficients (B, SE, t, p, CI, partial eta squared) in SPSS coding: one row per category, the last category of each factor is the reference and reported as a redundant 0 (`redundant = TRUE`)

descriptives Tibble with unadjusted cell means, SDs, and Ns

estimated_marginal_means Tibble with adjusted cell means (covariates at grand mean)

emm_main_effects For 2+ factors: named list of tibbles with the adjusted main-effect means (unweighted average of the cell means, as SPSS `/EMMEANS=TABLES(factor)`); NULL for one factor

levene_test Tibble with Levene's test of equality of error variances (f, df1, df2, p): as in SPSS UNIANOVA, a one-way ANOVA of the absolute residuals of the ANCOVA model across the cells of the design. Also available as `levene_test(result)`.

r_squared R-squared and Adjusted R-squared

model The underlying lm model object

call_info List with metadata (dv, factors, covariates, weighted, etc.)

Use `summary()` for the full SPSS-style output with toggleable sections. For data grouped with `group_by()`, one ANCOVA is computed per group: the tables carry the group keys as leading columns and `group_results` holds the complete result of each group (`NULL`, with a warning, for a group that cannot be analysed).

References

Huitema, B. E. (2011). The Analysis of Covariance and Alternatives (2nd ed.). Wiley.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`factorial_anova` for ANOVA without covariates.

`linear_regression` for regression analysis.

`oneway_anova` for single-factor ANOVA.

`summary.ancova` for detailed output with toggleable sections.

Other hypothesis_tests: `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# One-way ANCOVA: income by education, controlling for age
```

```
survey_data %>%
  ancova(dv = income, between = c(education), covariate = c(age))

# Two-way ANCOVA with weights
survey_data %>%
  ancova(dv = income, between = c(gender, education),
         covariate = c(age), weights = sampling_weight)

# Multiple covariates
survey_data %>%
  ancova(dv = income, between = c(education),
         covariate = c(age, political_orientation))

# --- Three-layer output ---
result <- ancova(survey_data, dv = income, between = c(education),
                 covariate = c(age))
result              # compact overview
summary(result)     # full detailed output with all sections
summary(result, marginal_means = FALSE)  # hide estimated marginal means
```

`anova.linear_regression` *ANOVA for a linear_regression model*

Description

For listwise + ungrouped results, dispatches to `stats::anova.lm` (sequential Type-I sum of squares per term). For the SPSS-style overall-model ANOVA table, use `object$anova_table`.

Usage

```
## S3 method for class 'linear_regression'
anova(object, ...)
```

Arguments

`object`	A `linear_regression` result.
`...`	Passed to `stats::anova.lm`.

Value

An anova table (sequential Type-I sums of squares), as returned by `stats::anova.lm`.

`anova.logistic_regression`

ANOVA for a logistic_regression model

Description

For ungrouped results, dispatches to `stats::anova.glm` (sequential deviance per term). For the SPSS-style omnibus chi-square test, use `object$omnibus_test`.

Usage

```
## S3 method for class 'logistic_regression'
anova(object, ...)
```

Arguments

`object`	A `logistic_regression` result.
`...`	Passed to `stats::anova.glm`.

Value

An anova table (sequential analysis of deviance per term), as returned by `stats::anova.glm`.

`as.data.frame.mariposa`

Convert Analysis Results to Data Frames

Description

Turns the result of any mariposa analysis into a plain data frame that can be saved with `utils::write.csv()`, combined with `dplyr::bind_rows()` or used for plotting. `tibble::as_tibble()` works the same way (it calls `as.data.frame()`), `write_xlsx()` writes the table (plus the secondary tables SPSS shows next to it) to Excel, and with the **broom** package loaded, `broom::tidy()` returns the table with broom's column names.

Usage

```
## S3 method for class 'ancova'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'binomial_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'chi_square'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

```
## S3 method for class 'chisq_gof'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'codebook'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'crosstab'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'describe'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'dunn_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'efa'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'factorial_anova'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'fisher_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'frequency'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'friedman_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'kendall_tau'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'kruskal_wallis'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'levene_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'linear_regression'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'logistic_regression'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'mann_whitney'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

```
## S3 method for class 'marginal_effects'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'mcnemar_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'multiple_response'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'normality_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'oneway_anova'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'pairwise_wilcoxon'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'partial_cor'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'pearson_cor'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'reliability'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'scheffe_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'spearman_rho'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 't_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'tukey_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_iqr'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_kurtosis'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_mean'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

```
## S3 method for class 'w_median'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_modus'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_quantile'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_range'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_sd'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_se'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_skew'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'w_var'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'wilcoxon_test'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A result of one of mariposa's analysis functions, e.g. t_test(), describe(), crosstab() or linear_regression().
row.names, optional	Ignored; present for compatibility with base::as.data.frame().
...	Ignored.

Details

broom::tidy() (available once **broom** is loaded) renames the common columns to broom's conventions: statistic, p.value, parameter (or num.df/den.df for F tests), estimate, conf.low, conf.high, std.error, adj.p.value (post-hoc tests), term (ANOVA tables), plus method and, where it applies, alternative. normality_test() results are tidied to one row per variable and test. The regressions keep their own tidy(), glance() and augment() methods.

Value

A plain data.frame (no list-columns, no row names) holding the main result table:

- one row per variable (per test) for tests and descriptive statistics, with the columns of `x$results`; group-level details stored as list-columns are flattened into columns (for example `group1`, `group2`, `mean1`, `mean2` for `t_test()`) or dropped (the observed/expected tables of `chi_square()`);
- one row per pair for correlations and post-hoc tests;
- one row per term for ANOVA tables and regression coefficients, and one row per item for `efa()` loadings;
- one row per cell for `crosstab()` and per category for `frequency()` (summary rows dropped, a `missing` column flags missing values);
- one row per scale for `reliability()`.

For grouped analyses (`dplyr::group_by()`) the grouping variables are the leading columns; labelled grouping variables show their value labels.

See Also

`write_xlsx()` for Excel export of results.

Examples

```
tt <- t_test(survey_data, age, income, group = gender)
as.data.frame(tt)

# Grouped analyses get the grouping variables as leading columns
survey_data |>
  dplyr::group_by(region) |>
  describe(age, income) |>
  as.data.frame()

# One row per cell of a crosstab
as.data.frame(crosstab(survey_data, gender, region))

# Save as CSV or convert to a tibble
csv <- tempfile(fileext = ".csv")
utils::write.csv(as.data.frame(tt), csv, row.names = FALSE)
tibble::as_tibble(chi_square(survey_data, gender, region))
unlink(csv)

# broom-style columns
if (requireNamespace("broom", quietly = TRUE)) {
  broom::tidy(oneway_anova(survey_data, age, group = education))
}
```

`binomial_test` *Test Whether a Proportion Matches an Expected Value*

Description

`binomial_test()` tests whether the observed proportion of a binary variable differs from a hypothesized proportion. It uses the exact binomial test, making it valid for any sample size.

Think of it as:

- Testing whether a coin is fair (proportion of heads = 50 percent)
- Checking if your sample's gender ratio matches the population
- Verifying if satisfaction rates meet a target proportion

The test tells you:

- Whether the observed proportion differs significantly from the expected
- The exact p-value (based on the binomial distribution)
- A confidence interval for the true proportion

Usage

```
binomial_test(data, ..., p = 0.5, weights = NULL, conf.level = 0.95)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	One or more binary variables to test. Each must have exactly 2 categories (e.g., Yes/No, Male/Female, 0/1, TRUE/FALSE)
`p`	The hypothesized proportion to test against (Default: 0.50 = 50 percent). It refers to Group 1, the category of the first valid case in the data (as in SPSS). With 0.5 the test is two-tailed; with any other value it is one-tailed in the direction of the observed proportion, as SPSS reports it.
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`conf.level`	Confidence level for intervals (Default: 0.95 = 95 percent)

Details

Understanding the Results:

P-value: If $p < 0.05$, the proportion differs significantly from the test proportion

- $p < 0.001$: Very strong evidence the proportion differs
- $p < 0.01$: Strong evidence the proportion differs
- $p < 0.05$: Moderate evidence the proportion differs
- $p > 0.05$: No significant difference from expected proportion

Observed Proportion: The actual proportion in your data. Compare this with the test proportion to see the direction of any difference.

Confidence Interval: The range likely to contain the true population proportion. If the test proportion falls outside this range, the result is significant.

When to Use This:

Use the binomial test when:

- You want to compare an observed proportion to a known value
- Your variable has exactly 2 categories
- You need an exact test (not relying on normal approximation)
- Sample size is small (where chi-square may not be reliable)

Relationship to Other Tests:

- For testing association between two categorical variables: Use `chi_square()` instead
- For comparing proportions between groups: Use chi-square or z-test for proportions
- For larger samples with normal approximation: results will be very similar to a one-sample z-test for proportions

Group 1 and one-tailed tests:

As in SPSS `NPAR TESTS /BINOMIAL`, Group 1 is the category of the first valid case in the data (per group with `group_by()`), and `p` is its hypothesized proportion. For `p = 0.5` the p-value is two-tailed. For any other `p` SPSS reports a one-tailed p-value in the direction of the observed proportion ("the proportion of cases in the first group < p"); `binomial_test()` does the same. To test the other category, put a case of it first or use `1 - p`.

Weighted analyses:

Each case weight is rounded to a whole number before the categories are counted. With all weights equal to 1 the weighted test reproduces the unweighted one exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

Value

Test results showing whether the proportion differs from expected, including:

- Category counts and observed proportions
- Test proportion (null hypothesis)
- Exact p-value (two-tailed for `p = 0.5`, otherwise one-tailed; column `alternative`: "two.sided", "less" or "greater")
- Confidence interval for the true proportion of Group 1 (Clopper-Pearson)

References

Conover, W. J. (1999). Practical nonparametric statistics (3rd ed.). John Wiley & Sons.

See Also

`binom.test` for the base R exact binomial test.

`chi_square` for testing associations between categorical variables.

Other hypothesis_tests: `ancova()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Test whether gender split is 50/50
survey_data %>%
  binomial_test(gender, p = 0.50)

# Test whether East region proportion is 50%
survey_data %>%
  binomial_test(region, p = 0.50)

# Multiple variables at once
survey_data %>%
  binomial_test(gender, region, p = 0.50)

# Weighted analysis
survey_data %>%
  binomial_test(gender, p = 0.50, weights = sampling_weight)

# Grouped analysis (separate test per region)
survey_data %>%
  group_by(region) %>%
  binomial_test(gender, p = 0.50)
```

---

center *Center Variables (Mean Centering)*

Description

Centers variables by subtracting the mean. When used on a grouped data frame (via `dplyr::group_by()`), this becomes group-mean centering — the R equivalent of separate centering within each group.

Usage

```
center(data, ..., weights = NULL, suffix = NULL, na.rm = TRUE)
```

Arguments

`data`	A data frame or numeric vector.
`...`	Variables to center (tidyselect). Only used when `data` is a data frame.
`weights`	Optional survey weights. When provided, the weighted mean is subtracted instead of the unweighted mean. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.

suffix	A character string appended to column names (e.g., "_c"). If NULL (default), original columns are overwritten.
na.rm	Remove missing values before computing the mean? Default: TRUE.

Details

Grand-Mean vs. Group-Mean Centering:

- **Grand-mean centering** (ungrouped): Subtracts the overall mean. A centered value of 0 means the respondent is at the sample average.
- **Group-mean centering** (grouped): Subtracts the group mean. Useful in multilevel models to separate within-group and between-group effects. This replaces sjmisc's separate de_mean() function.

Weighted Centering:

When weights is provided, the weighted mean is used for centering. This accounts for survey design in the centering computation.

Missing Values:

The result is a plain numeric variable (value labels are dropped, the variable label is kept with " (centered)" appended). Missing values of imported data, including their SPSS missing types, become plain NA, as in an SPSS COMPUTE.

Value

If data is a vector, a centered numeric vector. If data is a data frame, the modified data frame (invisibly).

See Also

std() for full standardization (centering + scaling)

Other transform: std()

Examples

```
library(dplyr)
data(survey_data)

# Grand-mean centering
data <- center(survey_data, income, age, suffix = "_c")

# Weighted centering
data <- center(survey_data, income, age,
               weights = sampling_weight, suffix = "_c")

# Group-mean centering (replaces sjmisc::de_mean)
data <- survey_data %>%
  group_by(region) %>%
  center(income, age, suffix = "_gmc")
```

chisq_gof *Chi-Square Goodness-of-Fit Test*

Description

chisq_gof() tests whether observed frequencies of a categorical variable match expected frequencies. By default, it tests against equal proportions (uniform distribution). You can also specify custom expected proportions.

Think of it as:

- Testing whether categories are equally distributed
- Comparing observed distribution to a theoretical distribution
- The one-sample version of the chi-square test

The test tells you:

- Whether observed frequencies differ from expected frequencies
- How strong the deviation is (chi-square statistic)
- A frequency table with observed, expected, and residual counts

Usage

```
chisq_gof(data, ..., expected = NULL, weights = NULL)
```

Arguments

data	Your survey data (data frame or tibble)
...	One or more categorical variables to test (tidyselect supported)
expected	Optional numeric vector of expected proportions, one per category. If NULL (default), equal proportions are assumed (SPSS /EXPECTED=EQUAL). Applied to every selected variable, so each variable must have that many categories. A named vector is matched to the categories by name (category label, or the code of a labelled variable); an unnamed vector is used in category order. As in SPSS /EXPECTED=50 30 20, values may also be counts or any relative frequencies: they are divided by their sum. Proportions (all values below 1) must sum to 1; a sum within 0.01 of 1 (e.g. 0.995 from rounding) is rescaled with a message.
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.

Details

Understanding the Results:

P-value: If $p < 0.05$, the distribution differs from expected

- $p < 0.001$: Very strong evidence the distribution differs

- p < 0.01: Strong evidence the distribution differs
- p < 0.05: Moderate evidence the distribution differs
- p >= 0.05: No significant deviation from expected distribution

Residuals: The difference between observed and expected counts. Large positive residuals indicate a category has more cases than expected; large negative residuals indicate fewer cases than expected.

When to Use This:

Use this test when:

- You want to check whether a categorical variable follows a specific distribution
- You want to test if categories are equally distributed (uniform)
- You have a single categorical variable and a hypothesised distribution

The Chi-Square Goodness-of-Fit Statistic:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where O_i = observed frequency, E_i = expected frequency.

Degrees of freedom = number of categories - 1.

Relationship to Other Tests:

- For testing association between two categorical variables: Use `chi_square()` instead
- For testing a single binary proportion: Use `binomial_test()` instead
- For small samples where expected frequencies are below 5: Use `fisher_test()` instead

SPSS Equivalent:

SPSS: `NPAR TESTS /CHISQUARE=variable /EXPECTED=EQUAL` or: `NPAR TESTS /CHISQUARE=variable /EXPECTED=50 30 20`

Value

Test results showing whether observed frequencies match expected, including:

- Chi-square statistic (`chi_squared`) and p-value for each variable
- Degrees of freedom
- Frequency table with observed, expected, and residual counts (`frequencies`; expected counts and residuals unrounded, printed with one decimal like SPSS; for grouped data one long table with the group columns and `Variable`)
- Sample size (N)

References

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50(302), 157-175.

See Also

chi_square for chi-square test of independence (two variables).

binomial_test for testing a single proportion.

Other hypothesis_tests: ancova(), binomial_test(), chi_square(), factorial_anova(), fisher_test(), friedman_test(), kruskal_wallis(), mann_whitney(), mcnemar_test(), oneway_anova(), t_test(), wilcoxon_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Test whether gender is equally distributed
survey_data %>%
  chisq_gof(gender)

# Test multiple variables at once
survey_data %>%
  chisq_gof(gender, region, education)

# Custom expected proportions
survey_data %>%
  chisq_gof(interview_mode, expected = c(0.5, 0.3, 0.2))

# With weights
survey_data %>%
  chisq_gof(gender, weights = sampling_weight)

# Grouped analysis
survey_data %>%
  group_by(region) %>%
  chisq_gof(education)
```

---

chi_square *Test If Two Categories Are Related*

Description

chi_square() helps you discover if two categorical variables are related or independent. For example, is education level related to voting preference? Or are they independent of each other?

The test tells you:

- Whether the relationship is statistically significant
- How strong the relationship is (effect sizes)
- What patterns exist in your data

Usage

```
chi_square(data, ..., weights = NULL, correct = FALSE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	Two categorical variables to test (e.g., gender, region)
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`correct`	Apply Yates' continuity correction to a 2x2 table? (Default: FALSE). The corrected statistic is reported as `chi_squared`; as in SPSS (which prints "Pearson Chi-Square" and "Continuity Correction" side by side), the uncorrected Pearson value is kept in `pearson_chi_squared` and Phi / Cramer's V are always computed from it.

Details

Understanding the Results:

P-value: If p < 0.05, the variables are likely related (not independent)

- p < 0.001: Very strong evidence of relationship
- p < 0.01: Strong evidence of relationship
- p < 0.05: Moderate evidence of relationship
- p >= 0.05: No significant relationship found

Effect Sizes (How strong is the relationship?):

- **Cramer's V**: Works for any table size (0 = no relationship, 1 = perfect relationship)
  - < 0.1: Negligible relationship
  - 0.1-0.3: Small relationship
  - 0.3-0.5: Medium relationship
  - 0.5 or higher: Large relationship
- **Phi**: sqrt(chi-squared / N). Reported for every table, as SPSS does; in a 2x2 table it equals Cramer's V, in larger tables it can exceed 1 (use Cramer's V there)
- **Gamma**: For two ordinal variables (-1 to +1, shows the direction of the relationship). Shown by `summary()` only when both variables are ordered factors or numeric; for nominal variables its sign depends on the arbitrary category order. `goodman_gamma()` always computes it.

When to Use This:

Use chi-squared test when:

- Both variables are categorical (gender, region, education level, etc.)
- You want to know if they're related or independent
- You have at least 5 observations in most cells

Reading the Frequency Tables:

- **Observed**: What you actually found in your data

- **Expected**: What you'd expect if variables were independent
- Large differences suggest a relationship exists

Tips for Success:

- Check that most cells have at least 5 observations
- Use weights for population estimates
- Look at both significance (p-value) and strength (effect sizes)
- Consider using crosstab() for detailed percentage breakdowns

Value

Test results showing whether the variables are related, including:

- Chi-squared statistic and p-value
- Observed vs expected frequencies
- Effect sizes to measure relationship strength Use `summary()` for the full SPSS-style output with toggleable sections.

References

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50(302), 157–175.

Cramer, H. (1946). *Mathematical Methods of Statistics*. Princeton University Press.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`chisq.test` for the base R chi-squared test.

`crosstab` for detailed cross-tabulation tables.

`frequency` for single-variable frequency tables.

`summary.chi_square` for detailed output with toggleable sections.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic chi-squared test for independence
survey_data %>% chi_square(gender, region)

# With weights
survey_data %>% chi_square(gender, education, weights = sampling_weight)
```

```
# Grouped analysis
survey_data %>%
  group_by(region) %>%
  chi_square(gender, employment)

# With continuity correction
survey_data %>% chi_square(gender, region, correct = TRUE)

# --- Three-layer output ---
result <- chi_square(survey_data, gender, education)
result              # compact one-line overview
summary(result)     # full detailed output with all sections
summary(result, cross_tabulation = FALSE)  # hide cross-tabulation
```

---

codebook *Create a Codebook for Your Data*

Description

codebook() creates an interactive HTML data dictionary that opens in the RStudio Viewer. It shows you everything about your dataset at a glance: variable names, types, labels, the empirical values found in the data, value labels, and frequency counts.

Think of it as a professional "cheat sheet" for your dataset – especially useful when working with labelled survey data imported from SPSS, Stata, or SAS.

Usage

```
codebook(
  data,
  ...,
  weights = NULL,
  show_id = TRUE,
  show_type = TRUE,
  show_labels = TRUE,
  show_values = TRUE,
  show_freq = TRUE,
  show_na = TRUE,
  show_unused = FALSE,
  max_values = 10,
  max_len = 50,
  sort_by_name = FALSE,
  file = NULL,
  view = interactive()
)
```

Arguments

data	Your survey data (a data frame or tibble)
...	Optional: specific variables to include. If empty, all variables are shown. Supports tidyselect helpers like starts_with("trust").
weights	Optional survey weights for weighted frequency calculations. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
show_id	Show variable position number? (Default: TRUE)
show_type	Show data type? (Default: TRUE)
show_labels	Show variable labels? (Default: TRUE)
show_values	Show empirical values and value labels? (Default: TRUE)
show_freq	Show frequency counts? (Default: TRUE)
show_na	Show tagged missing value types in the codebook? (Default: TRUE). When data was imported with read_spss(), read_stata(), read_sas(), or read_xpt() using tagged NAs, they are displayed with their original missing value codes, labels, and frequencies below the valid values, separated by a thin gray line.
show_unused	Show all defined value labels, even those with zero observations? (Default: FALSE). Useful for seeing the full codebook including response options that no respondent selected.
max_values	Maximum number of values to display per variable before truncating or showing a range (Default: 10)
max_len	Maximum character width for labels before truncation (Default: 50)
sort_by_name	Sort variables alphabetically instead of by position? (Default: FALSE)
file	Path to save the HTML codebook. If NULL (default), no file is written unless the codebook is opened in the Viewer (then a temporary file is used). The directory of file must already exist.
view	Open the HTML codebook in the RStudio Viewer (or browser)? Defaults to interactive(), so interactive sessions open the Viewer and scripts/tests do not; while an R Markdown/Quarto document is knitted the default is FALSE (the codebook is embedded in the document instead). Set view = FALSE to suppress the Viewer side effect entirely; file = writing is unaffected by this argument.

Details

What the Codebook Shows:

For each variable, the codebook displays (depending on options):

- **ID**: Column position in the dataset
- **Name**: Variable name (in monospace font)
- **Type**: Data type (numeric, factor, ordered factor, haven_labelled, etc.)
- **Label**: Variable label (from SPSS/Stata/SAS imports or manual assignment)
- **Values**: The empirical values found in the data
- **Value Labels**: Labels assigned to those values (if any)

- **Freq.**: Frequency count for each value

When to Use This:

Use `codebook()` when you:

- First receive a new dataset and want to understand its structure
- Work with labelled data (SPSS, Stata, SAS) and need to see all value labels
- Want to document your dataset for colleagues or publications
- Need to quickly see value distributions across variables

In R Markdown and Quarto:

A chunk containing `codebook(data)` embeds the HTML codebook in HTML output (its styles only apply to the codebook, not to the rest of the document); PDF and Word output show the compact console overview. Use `write_xlsx()` for an Excel version of the codebook.

Value

A list of class `"codebook"`. Printing it shows a compact console overview; in a knitted HTML document (R Markdown, Quarto) the full HTML codebook is embedded, in other formats the overview is shown. The list contains:

codebook Tibble with one row per variable and all metadata

data_info List with dataset-level information (name, nrow, ncol, etc.)

html The generated HTML as an htmltools object

weights Name of the weight variable, or NULL

options List of all display options

frequencies Named list of frequency tables per variable

See Also

`describe()` for detailed numeric summaries, `frequency()` for detailed frequency tables, `read_spss()`, `read_por()`, `read_stata()`, `read_sas()`, `read_xpt()` for importing data with tagged NAs

Examples

```
data(survey_data)

# Compact console overview (no Viewer)
cb <- codebook(survey_data)
print(cb)

# Detailed console output
summary(cb)


# Full codebook (opens in RStudio Viewer when interactive)
codebook(survey_data)

# Only trust-related variables
```

```
codebook(survey_data, starts_with("trust"))

# Save to file for sharing
codebook(survey_data, file = tempfile(fileext = ".html"))
```

`coef.linear_regression`

Coefficients of a linear_regression model

Description

The unstandardized coefficients B. Also available for pairwise results (taken from the coefficients table); grouped results hold one model per group in `$groups`.

Usage

```
## S3 method for class 'linear_regression'
coef(object, ...)
```

Arguments

`object`	A `linear_regression` result.
`...`	Not used.

Value

A named numeric vector.

`confint.linear_regression`

Confidence intervals for linear_regression coefficients

Description

t-based intervals for B. For weighted models they use the SPSS frequency-weight standard errors and df (`sum(w) - rank`) and equal the CI columns of `summary()`.

Usage

```
## S3 method for class 'linear_regression'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	A linear_regression result (ungrouped, listwise).
parm	Coefficients to compute intervals for (names or indices; default all).
level	Confidence level (default 0.95).
...	Not used.

Value

A matrix with one row per coefficient and columns for the lower and upper limits.

---

confint.logistic_regression

Confidence intervals for logistic regression coefficients

Description

Confidence intervals for the coefficients B (log-odds scale) of a logistic_regression model.

The default method = "wald" gives Wald intervals ($B \pm z_{1-\alpha/2} \cdot SE$), the interval SPSS LOGISTIC REGRESSION reports as "95\ summary() prints: exp(confint(model)) reproduces the Lower/Upper columns of the coefficients table. method = "profile" gives the profile-likelihood intervals of stats::confint() for glm objects.

Usage

```
## S3 method for class 'logistic_regression'
confint(object, parm, level = 0.95, method = c("wald", "profile"), ...)
```

Arguments

object	A logistic_regression result (ungrouped).
parm	Coefficients to compute intervals for (names or indices; default all).
level	Confidence level (default 0.95).
method	"wald" (default, SPSS) or "profile".
...	Passed to the underlying confint method.

Value

A matrix with one row per coefficient and columns for the lower and upper limits (log-odds scale).

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction >= 4)
model <- logistic_regression(survey_data, high_satisfaction ~ age + gender)
confint(model)        # Wald (SPSS)
exp(confint(model))   # SPSS "95% C.I. for EXP(B)"
```

copy_labels *Copy Labels from One Data Frame to Another*

Description

Copies variable labels, value labels, and tagged NA metadata from a source data frame to a target data frame. dplyr verbs such as `dplyr::filter()`, `dplyr::select()` or `dplyr::arrange()` keep labels; they are lost by base-R conversions and computations (e.g. `as.numeric()`, `ifelse()`, arithmetic on labelled vectors), by `merge()` / `rbind()` of plain data frames, or by a detour through CSV. `copy_labels()` restores them from the original data.

Usage

```
copy_labels(data, source)
```

Arguments

`data`	The target data frame (e.g., after filtering or subsetting).
`source`	The source data frame with the original labels.

Details

The variable label (`"label"`) is always copied. The following are copied only when the target column still holds the source's codes (a numeric column whose values are observed values or labelled codes of the source column):

- `"labels"` — value labels
- `"na_tag_map"` — tagged NA mapping
- `"na_tag_format"` — tagged NA format (spss/stata/sas)
- `"na_values"`, `"na_range"` — SPSS missing-value definitions
- `"class"` — vector class (e.g., `haven_labelled`)

A converted column (e.g. a factor from `to_label()`) or a summarised one (e.g. group means) keeps its own type and gets the variable label only.

Value

The target data frame with labels copied from the source. Only columns present in both data frames are affected. Columns only in the target are left unchanged.

See Also

`var_label()`, `val_labels()`

Other labels: `drop_labels()`, `find_var()`, `set_na()`, `to_character()`, `to_label()`, `to_labelled()`, `to_numeric()`, `unlabel()`, `val_labels()`, `var_label()`

Examples

```
# as.numeric() drops the variable label
data_plain <- dplyr::mutate(survey_data, age = as.numeric(age))
attr(data_plain$age, "label")

# Restore it
data_plain <- copy_labels(data_plain, survey_data)
attr(data_plain$age, "label")
```

---

crosstab *Compare Two Categories: See How They Relate*

Description

`crosstab()` shows you how two categorical variables relate to each other. It creates a table that reveals patterns - like whether education level differs by region, or if gender influences product preferences.

Think of it as a two-way frequency table that shows:

- How many people fall into each combination of categories
- What percentage each cell represents
- Whether there are patterns or associations

Usage

```
crosstab(
  data,
  row,
  col,
  weights = NULL,
  percentages = c("row", "none", "col", "total", "all"),
  na.rm = TRUE,
  digits = 1
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`row`	The variable for table rows (e.g., education, age_group)
`col`	The variable for table columns (e.g., region, gender)
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`percentages`	Which percentages to show:

- `"row"` (default): Percentages across each row (adds to 100% horizontally)
- `"col"`: Percentages down each column (adds to 100% vertically)
- `"total"`: Percentage of the entire table
- `"all"`: Show all three types
- `"none"`: Just counts, no percentages

`na.rm` Remove missing values before calculating? (Default: TRUE). With `FALSE`, missing values of either variable form their own `"NA"` row or column.

`digits` Decimal places for percentages (Default: 1)

Details

Understanding the Results:

The crosstab table shows:

- **Cell counts**: Number of people in each combination
- **Row %**: Distribution within each row (e.g., "Among those with high school education, X% live in the East")
- **Column %**: Distribution within each column (e.g., "Among those in the East, X% have high school education")
- **Total %**: Percentage of the entire sample (e.g., "X% of all respondents have high school education AND live in the East")
- **Adjusted residuals** (via `summary(result, residuals = TRUE)`, SPSS `/CELLS=ASRESID`): which cells deviate from independence. After a significant `chi_square` test, cells with |adj. residual| > 2 are the ones driving the association. For weighted tables the residuals are computed on the rounded weighted cell counts (see below). They follow Haberman's formula (as `chisq.test()$stdres`) and are not compared with SPSS output.

Weighted Tables:

As SPSS CROSSTABS does by default (`/COUNT ROUND CELL`), each weighted cell count is rounded to a whole number first; the row and column totals are sums of the rounded cells and all percentages come from the rounded counts. The table total can therefore differ by a few cases from the rounded sum of weights shown as "N (valid)" (SPSS: 2518 vs. 2516 for education x employment in `survey_data`).

When to Use This:

Use crosstab when you want to:

- See if two categorical variables are related
- Compare distributions across groups
- Find patterns in survey responses
- Create demographic breakdowns

Choosing Percentages:

- **Row %**: Use when your row variable is the grouping factor (e.g., "How does region vary BY education level?")
- **Column %**: Use when your column variable is the grouping factor (e.g., "How does education vary BY region?")

- **Total %**: Use to understand the overall sample composition

Tips for Success:

- Start with row or column percentages, not both at once
- Use chi-squared test to check if the relationship is statistically significant
- Watch for small cell counts (< 5) which may be unreliable
- Consider combining sparse categories if many cells are empty

Value

A cross-tabulation table showing the relationship between two variables. The result also carries the expected cell counts (`$expected`) and adjusted standardized residuals (`$adj_residuals`); display the residuals with `summary(result, residuals = TRUE)`.

See Also

`table` for base R contingency tables.

`frequency` for single-variable frequency tables.

`chi_square` for testing if the cross-tabulated variables are related.

Other descriptive: `describe()`, `frequency()`, `multiple_response()`, `normality_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic crosstab
survey_data %>% crosstab(gender, region)

# With weights and all percentages
survey_data %>% crosstab(gender, education,
                         weights = sampling_weight,
                         percentages = "all")

# Grouped analysis
survey_data %>%
  group_by(employment) %>%
  crosstab(gender, region, weights = sampling_weight)

# Column percentages only
survey_data %>% crosstab(education, employment, percentages = "col")
```

describe *Get to Know Your Numeric Data*

Description

`describe()` gives you a complete summary of numeric variables - like age, income, or satisfaction scores. It's your first step in any analysis, helping you understand what's typical, what's unusual, and how spread out your data is.

Think of it as a health check for your data that reveals:

- What's the average value?
- What's the middle value?
- How spread out are the responses?
- Are there unusual patterns or outliers?

Usage

```
describe(
  data,
  ...,
  weights = NULL,
  show = "short",
  probs = c(0.25, 0.5, 0.75),
  na.rm = TRUE,
  excess = TRUE
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to summarize. List them separated by commas, or use helpers like `starts_with("trust")`
`weights`	Optional survey weights for population-representative results. Without weights, you describe your sample. With weights, you describe the population. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`show`	Which statistics to display: • `"short"` (default): Essential stats (mean, median, SD, range, IQR, skewness) • `"all"`: Everything including min/max, variance, kurtosis, mode, quantiles • Custom list: Choose specific stats like `c("mean", "sd", "range")` from `"mean"`, `"median"`, `"sd"`, `"se"`, `"var"`, `"min"`, `"max"` (SPSS DESCRIPTIVES' Minimum/Maximum), `"range"`, `"iqr"`, `"skew"`, `"kurtosis"`, `"mode"`, `"quantiles"`. Unknown names are an error.

probs	For quantiles, which percentiles to show (default: 25th, 50th, 75th). The 50th percentile is the median; when the median is shown it is not printed a second time (the result keeps both columns).
na.rm	Remove missing values before calculating? (Default: TRUE). With FALSE, the result for a variable that contains missing values is NA (as in base R).
excess	For kurtosis, show excess kurtosis? (Default: TRUE, easier to interpret)

Details

Understanding the Results:

Key statistics and what they tell you:

- **n**: How many valid responses (watch for too many missing)
- **Mean**: The average value
- **Median**: The middle value (half above, half below)
- **SD**: Standard deviation - how spread out values are
- **Range**: The minimum and maximum values
- **IQR**: Interquartile range - the middle 50% of values
- **Skewness**: Whether data leans left (negative) or right (positive)
- **Kurtosis**: Whether you have unusual outliers
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as in SPSS FREQUENCIES with WEIGHT BY. Kish's effective sample size is kept in the result (<variable>_Effective_N) but not printed.

When to Use This:

Always start here! Use describe() to:

- Check data quality (impossible values?)
- Understand distributions before testing
- Spot outliers that might affect analyses
- Compare groups side by side

Interpreting Patterns:

- **Mean close to Median**: Data is roughly symmetric
- **Mean > Median**: Right-skewed (tail extends right)
- **Mean < Median**: Left-skewed (tail extends left)
- **Large SD**: Responses vary widely
- **Small SD**: Responses are similar

Value

A summary table with descriptive statistics for each variable

See Also

summary for base R summary statistics.

frequency for categorical variable summaries.

w_mean, w_sd, w_median for individual weighted statistics.

t_test and oneway_anova for group comparisons.

Other descriptive: crosstab(), frequency(), multiple_response(), normality_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic unweighted analysis
survey_data %>% describe(age)

# Weighted analysis
survey_data %>% describe(age, weights = sampling_weight)

# Multiple variables with custom statistics
survey_data %>% describe(age, income, life_satisfaction,
                         weights = sampling_weight,
                         show = c("mean", "sd", "skew"))

# Grouped analysis
survey_data %>%
  group_by(region) %>%
  describe(age, weights = sampling_weight)
```

---

df.residual.linear_regression

Residual degrees of freedom of a linear_regression model

Description

Weighted models: sum(w) - rank (SPSS frequency weights, non-integer); unweighted: the lm value.

Usage

```
## S3 method for class 'linear_regression'
df.residual(object, ...)
```

Arguments

object	A linear_regression result (ungrouped, listwise).
...	Not used.

Value

A single number.

`drop_labels` *Remove Unused Value Labels*

Description

Removes value labels for values that are not present in the data, and unused levels of factors. This is useful after filtering or subsetting, when some categories may no longer exist but their labels remain attached.

Usage

```
drop_labels(data, ..., drop_na = FALSE)
```

Arguments

`data`	A data frame or a single vector.
`...`	Optional: unquoted variable names (tidyselect supported). If empty, applies to all labelled and factor columns.
`drop_na`	If `TRUE`, also removes tagged NA labels. Default: `FALSE` (tagged NA labels are preserved even if no tagged NAs of that type exist).

Details

This is useful after subsetting data (e.g., filtering out a category). The removed category's label still exists on the variable, which can cause confusing output in `frequency()` or `codebook()`. `drop_labels()` cleans this up by keeping only labels for values actually present in the data. For factors, unused levels are dropped; the variable label and the original codes of a `to_label()` factor are kept.

By default, tagged NA labels are preserved (`drop_na = FALSE`) because they represent missing value types, not substantive categories.

Value

The data with unused labels removed.

See Also

`val_labels()`, `copy_labels()`

Other labels: `copy_labels()`, `find_var()`, `set_na()`, `to_character()`, `to_label()`, `to_labelled()`, `to_numeric()`, `unlabel()`, `val_labels()`, `var_label()`

Examples

```
# After filtering, "Basic Secondary" no longer occurs but remains a level
data_subset <- dplyr::filter(survey_data, education != "Basic Secondary")
levels(data_subset$education)
data_clean <- drop_labels(data_subset)
levels(data_clean$education)

# Value labels of labelled vectors
if (requireNamespace("haven", quietly = TRUE)) {
  x <- haven::labelled(c(1, 2, 2), labels = c(low = 1, mid = 2, high = 3))
  drop_labels(x)   # the unused label "high" is removed
}
```

---

dunn_test *Find Which Specific Groups Differ After Kruskal-Wallis*

Description

`dunn_test()` tells you exactly which groups are different from each other after Kruskal-Wallis finds overall differences. It's the non-parametric equivalent of a Tukey or Scheffe post-hoc test.

Think of it as:

- Kruskal-Wallis says "there are differences somewhere"
- Dunn test says "specifically, Group A differs from Group C"
- A way to make all possible pairwise comparisons using rank-based statistics

Usage

```
dunn_test(x, ...)

## Default S3 method:
dunn_test(x, ...)
```

Arguments

`x`	Kruskal-Wallis results from `kruskal_wallis()`
`...`	Additional arguments passed to methods. The method for `kruskal_wallis` objects accepts `p_adjust` (character): method for adjusting p-values for multiple comparisons. Options: `"bonferroni"` (default, most conservative), `"holm"`, `"BH"`, `"hochberg"`, `"hommel"`, `"BY"`, `"fdr"`, `"none"`.

Details

Understanding the Results:

Z-Statistics: Based on differences in mean ranks between groups

- Large absolute Z values indicate big rank differences
- Positive Z: First group has higher mean rank than second
- Negative Z: Second group has higher mean rank than first

Adjusted P-values: Control for multiple comparisons

- p < 0.05: Groups are significantly different
- p >= 0.05: No significant difference between these groups

The Dunn Test Formula:

For each pair of groups (i, j):

$$Z_{ij} = \frac{\bar{R}_i - \bar{R}_j}{\sqrt{\frac{N(N+1)}{12}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

where $\bar{R}_i$ is the mean rank for group i, N is the total sample size, and n_i is the size of group i.

P-Value Adjustment Methods:

- **Bonferroni** (default): Most conservative, multiplies p by number of comparisons
- **Holm**: Step-down method, less conservative than Bonferroni
- **BH**: Controls false discovery rate, good for many comparisons

When to Use This:

Use Dunn test when:

- Your Kruskal-Wallis test shows significant differences (p < 0.05)
- You want to know which specific groups differ
- Your data violates normality assumptions
- You have ordinal data or skewed distributions

Relationship to Other Tests:

- Non-parametric equivalent of `tukey_test()` and `scheffe_test()`
- Follow-up to `kruskal_wallis()`, just like Tukey follows ANOVA

Weighted analyses:

When the `kruskal_wallis()` result is weighted, the pairwise z statistics use the same frequency-weighted midranks. With all weights equal to 1 they reproduce the unweighted comparisons exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- Uses ranks instead of raw values, making it robust to outliers

Value

Pairwise comparison results showing:

- Which group pairs are significantly different
- Z-statistics based on rank differences
- Adjusted p-values (controlling for multiple comparisons)

The comparison table (one row per pair) is stored as `$results` (also available as `$comparisons`).

References

Dunn, O. J. (1964). Multiple comparisons using rank sums. Technometrics, 6(3), 241-252.

See Also

`kruskal_wallis` for performing Kruskal-Wallis tests.

`tukey_test` for parametric post-hoc comparisons after ANOVA.

`scheffe_test` for conservative parametric post-hoc comparisons.

Other posthoc: `levene_test()`, `pairwise_wilcoxon()`, `scheffe_test()`, `tukey_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Perform Kruskal-Wallis followed by Dunn post-hoc test
kw_result <- survey_data %>%
  kruskal_wallis(life_satisfaction, group = education)

# Dunn post-hoc comparisons (default: Bonferroni)
kw_result %>% dunn_test()

# With Holm correction (less conservative)
kw_result %>% dunn_test(p_adjust = "holm")

# With Benjamini-Hochberg (controls false discovery rate)
kw_result %>% dunn_test(p_adjust = "BH")

# With weights
kw_weighted <- survey_data %>%
  kruskal_wallis(life_satisfaction, group = education,
                 weights = sampling_weight)

kw_weighted %>% dunn_test()

# Multiple variables
kw_multi <- survey_data %>%
  kruskal_wallis(life_satisfaction, trust_government,
                 group = education)
```

```
kw_multi %>% dunn_test()

# Grouped analysis
kw_grouped <- survey_data %>%
  group_by(region) %>%
  kruskal_wallis(life_satisfaction, group = education)

kw_grouped %>% dunn_test()
```

---

efa *Explore the Structure Behind Your Survey Items*

Description

efa() performs Exploratory Factor Analysis (EFA) to discover underlying patterns in your survey items. Supports both Principal Component Analysis (PCA) and Maximum Likelihood (ML) extraction. This is the R equivalent of SPSS's FACTOR procedure.

For example, if you have 6 items measuring different attitudes, EFA can reveal whether they group into 2-3 underlying dimensions (factors or components).

Usage

```
efa(
  data,
  ...,
  n_factors = NULL,
  rotation = "varimax",
  extraction = "pca",
  weights = NULL,
  use = "pairwise",
  sort = TRUE,
  blank = 0.4,
  na.rm = TRUE
)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The items to analyze. Use bare column names separated by commas, or tidyselect helpers like starts_with("trust").
n_factors	Number of components to extract (a whole number). Default NULL uses the Kaiser criterion (eigenvalue > 1). A single component cannot be rotated; it is shown unrotated with SPSS's note.

rotation	Rotation method: "varimax" (default, orthogonal), "oblimin" (oblique direct oblimin, delta = 0, allows correlated factors), "promax" (oblique, power 4), or "none". All rotations use Kaiser normalization and SPSS FACTOR's own algorithms and stopping rules (at most 25 iterations, as SPSS /CRITERIA ITERATE(25)), so they reproduce SPSS's rotated matrices; they differ slightly from stats::varimax(), stats::promax() and GPArotation::oblimin().
extraction	Extraction method: "pca" (default, Principal Component Analysis) or "ml" (Maximum Likelihood, enables goodness-of-fit testing, assumes multivariate normality). ML uses SPSS's starting values, bounds a Heywood variable's communality at .999 as SPSS does and keeps SPSS's factor order.
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
use	How to handle missing data for correlation computation: "pairwise" (default, matches SPSS) or "complete" (listwise; "listwise" is accepted as an alias).
sort	Logical. Sort loadings by size within each component? Default TRUE.
blank	Numeric. Suppress (hide) loadings with absolute value below this threshold in the print output. Default 0.40 (matches SPSS BLANK(.40)). Set to 0 to show all loadings.
na.rm	Logical. Remove missing values? Default TRUE.

Details

Understanding the Results:

KMO (Kaiser-Meyer-Olkin) measures sampling adequacy:

- KMO > 0.90: Marvelous
- KMO 0.80 - 0.90: Meritorious
- KMO 0.70 - 0.80: Middling
- KMO 0.60 - 0.70: Mediocre
- KMO 0.50 - 0.60: Miserable
- KMO < 0.50: Unacceptable - don't use factor analysis

Bartlett's Test of Sphericity tests whether correlations are significantly different from zero. A significant result ($p < .05$) means the correlation matrix is suitable for factor analysis.

Eigenvalues indicate how much variance each component explains. The Kaiser criterion retains components with eigenvalue > 1.

Factor Loadings show how strongly each item relates to each component:

- |loading| > 0.70: Strong association
- |loading| 0.40 - 0.70: Moderate association
- |loading| < 0.40: Weak (suppressed by default)

Communalities show how much of each item's variance is explained by the extracted components. Low communalities (< 0.40) suggest the item doesn't fit well with the others.

Choosing an Extraction Method:

- **PCA** (default): Extracts components explaining maximum total variance. Simple and robust. Does not assume normality.
- **ML**: Extracts factors explaining shared variance only. Assumes multivariate normality. Provides a goodness-of-fit test to evaluate model fit. A non-significant chi-square ($p > .05$) suggests adequate fit.

Choosing a Rotation:

- **Varimax** (default): Assumes factors are uncorrelated. Produces simpler, easier-to-interpret results.
- **Oblimin**: Allows factors to be correlated. More realistic for social science data. Produces both a Pattern Matrix (unique contributions) and Structure Matrix (total correlations).
- **Promax**: Oblique rotation based on a power transformation of Varimax results. Like Oblimin, produces Pattern and Structure matrices. Common alternative to Oblimin in SPSS.
- **None**: No rotation. Rarely useful for interpretation.

Value

An `efa` result object containing:

loadings Component loading matrix (rotated if rotation applied)

unrotated_loadings Unrotated component matrix

eigenvalues All eigenvalues from the correlation matrix

variance_explained Tibble with the initial eigenvalues: Total, % of Variance, Cumulative % (one row per variable)

extraction_variance Tibble with the extraction sums of squared loadings of the extracted components/factors (SPSS "Extraction Sums of Squared Loadings"). For ML this is the variance the common factors explain - the figure the compact print reports.

rotation_variance Tibble with rotation sums of squared loadings (for the oblique rotations the column sums of squares of the structure matrix, as SPSS reports them)

communalities Extraction communalities for each variable

kmo List with overall KMO and per-item MSA values

bartlett List with chi_sq, df, and p_value

rotation Rotation method used

extraction Extraction method used

n_factors Number of components extracted

correlation_matrix Correlation matrix used for analysis

initial_communalities Initial communalities (1.0 for PCA, SMC for ML)

goodness_of_fit Goodness-of-fit test (ML only): chi_sq, df, p_value. NULL for PCA.

uniquenesses Unique variances per variable (ML only, NULL for PCA)

pattern_matrix Pattern matrix (oblimin/promax only, NULL otherwise)

structure_matrix Structure matrix (oblimin/promax only, NULL otherwise)

factor_correlations Factor correlation matrix (oblimin/promax only, NULL otherwise)

rotation_iterations, rotation_converged Iterations of the rotation as SPSS counts them ("Rotation converged in 4 iterations"; for promax those of its varimax step) and whether it converged; NULL without rotation

variables Character vector of variable names

variable_labels Named character vector with the variable labels (`NA` where a variable has none); `summary()` shows them next to the names, shortened to the console width in the tables

weights Weights variable name or NULL

item_statistics Tibble with mean, SD, analysis N and missing N per item. With `use = "pairwise"` each item uses its own valid cases; with `use = "complete"` all items use the complete cases.

n Sample size: the smallest pairwise N (`use = "pairwise"`, the N of Bartlett's test as in SPSS) or the number of complete cases (`use = "complete"`); the sum of weights when weighted

use The missing-data handling used (`"pairwise"` or `"complete"`)

col_prefix Column name prefix: `"PC"` for PCA, `"Factor"` for ML

sort Whether loadings are sorted

blank Suppression threshold

Use `summary()` for the full SPSS-style output with toggleable sections.

See Also

`reliability` for checking scale reliability before creating indices.

`row_means` for creating mean indices after identifying factors.

`summary.efa` for detailed output with toggleable sections.

Other scale: `pomps()`, `reliability()`, `row_count()`, `row_means()`, `row_sums()`

Examples

```
library(dplyr)
data(survey_data)

# Basic EFA with Varimax rotation
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science)

# With Oblimin rotation
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science,
    rotation = "oblimin")

# Maximum Likelihood extraction
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science,
    extraction = "ml")
```

```
# Promax rotation (oblique)
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science,
    rotation = "promax")

# Fix number of factors
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science,
    n_factors = 2)

# With survey weights
efa(survey_data,
    political_orientation, environmental_concern, life_satisfaction,
    trust_government, trust_media, trust_science,
    weights = sampling_weight)

# Grouped by region
survey_data %>%
  group_by(region) %>%
  efa(political_orientation, environmental_concern, life_satisfaction,
      trust_government, trust_media, trust_science)

# --- Three-layer output ---
result <- efa(survey_data, political_orientation, environmental_concern,
              life_satisfaction, trust_government, trust_media, trust_science)
result              # compact overview
summary(result)     # full detailed output with all sections
summary(result, communalities = FALSE)  # hide communalities table
```

`factorial_anova` *Compare Groups Across Multiple Factors: Factorial ANOVA*

Description

`factorial_anova()` tests whether group means differ across two or more factors simultaneously, including their interactions. It performs a factorial (two-way, three-way) ANOVA using Type III Sum of Squares, matching SPSS UNIANOVA output.

Think of it as:

- Testing multiple grouping variables at once
- Detecting interaction effects (do factor combinations matter?)
- An extension of one-way ANOVA to multiple factors

The test tells you:

- Whether each factor has a main effect on the outcome
- Whether factors interact (the effect of one depends on the other)
- How much variance each factor explains (partial eta squared)

Usage

```
factorial_anova(data, dv, between, weights = NULL, ss_type = 3)
```

Arguments

data	Your survey data (a data frame or tibble)
dv	The numeric dependent variable to analyze (unquoted)
between	Character vector or unquoted variable names specifying the between-subjects factors (2-3 factors). These must be categorical variables (factor, character, or labelled numeric).
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
ss_type	Deprecated; only 3 (Type III, the SPSS default) is implemented. Passing 2 issues a warning and computes Type III.

Details

Understanding the Results:

Main Effects: Does each factor independently affect the outcome?

- Significant main effect = group means differ for that factor
- Example: Education affects income regardless of gender

Interaction Effects: Does the effect of one factor depend on another?

- Significant interaction = the pattern differs across factor combinations
- Example: The gender gap in income varies by education level

Partial Eta Squared (Effect Size):

- Less than 0.01: Negligible
- 0.01 to 0.06: Small
- 0.06 to 0.14: Medium
- 0.14 or greater: Large

Type III Sum of Squares:

Type III SS tests each effect after adjusting for all other effects. This is the standard in SPSS and recommended for unbalanced designs (unequal cell sizes). It uses orthogonal contrasts (contr.sum) internally.

When to Use This:

Use factorial ANOVA when:

- You have one numeric outcome variable
- You have 2-3 categorical grouping factors
- You want to test main effects AND interactions
- Your data is approximately normally distributed within cells

What Comes Next?:

If the ANOVA is significant:

1. Check which effects are significant (main effects vs. interactions)
2. Use `tukey_test()` for post-hoc comparisons on main effects
3. Examine cell means to interpret interaction patterns
4. Consider effect sizes for practical significance

Value

An object of class `"factorial_anova"` containing:

anova_table Tibble with Source, SS, df, MS, F, p, Partial Eta Squared

descriptives Tibble with cell means, SDs, and Ns for each factor combination

levene_test Tibble with Levene's test results (F, df1, df2, p)

r_squared R-squared and Adjusted R-squared

model The underlying model object for S3 dispatch

call_info List with metadata (dv, factors, weighted, n_total, n_missing)

Use `summary()` for the full SPSS-style output with toggleable sections. For data grouped with `group_by()`, one ANOVA is computed per group: the tables carry the group keys as leading columns, `group_results` holds the complete result of each group (`NULL`, with a warning, for a group that cannot be analysed), and `print()`, `summary()`, `tukey_test()`, `scheffe_test()` and `levene_test()` report per group.

References

Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences (2nd ed.). Lawrence Erlbaum Associates.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

Maxwell, S. E., & Delaney, H. D. (2004). Designing Experiments and Analyzing Data (2nd ed.). Lawrence Erlbaum Associates.

See Also

`oneway_anova` for single-factor ANOVA.

`tukey_test` for post-hoc pairwise comparisons.

`levene_test` for testing homogeneity of variances.

`ancova` for ANOVA with covariates.

`summary.factorial_anova` for detailed output with toggleable sections.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Two-way ANOVA: income by gender and education
survey_data %>%
  factorial_anova(dv = income, between = c(gender, education))

# Two-way ANOVA with weights
survey_data %>%
  factorial_anova(dv = life_satisfaction, between = c(gender, region),
                  weights = sampling_weight)

# Separate ANOVA for each region
survey_data %>%
  group_by(region) %>%
  factorial_anova(dv = life_satisfaction, between = c(gender, education))

# Three-way ANOVA
survey_data %>%
  factorial_anova(dv = income, between = c(gender, region, education))

# Follow up with post-hoc tests
result <- survey_data %>%
  factorial_anova(dv = income, between = c(gender, education))
result %>% tukey_test()
result %>% levene_test()

# --- Three-layer output ---
result              # compact overview
summary(result)     # full detailed output with all sections
summary(result, descriptives = FALSE)  # hide the cell descriptives
```

---

find_var *Find Variables by Name or Label*

Description

Searches for variables in your data by matching a pattern against variable names, variable labels, or both. This is especially useful for SPSS datasets where variable names are often cryptic codes (e.g., `v104`, `q23a_1`) and the actual meaning is stored in variable labels.

Usage

```
find_var(
  data,
  pattern,
```

```
  search = c("name_label", "name", "label"),
  fixed = FALSE
)
```

Arguments

data	A data frame (typically imported from SPSS with read_spss).
pattern	A search term or regular expression to match against variable names and/or labels. Case-insensitive by default.
search	Where to search: "name_label" (default) searches both variable names and labels, "name" searches only names, "label" searches only labels.
fixed	If TRUE, pattern is searched as literal text (case-insensitive), e.g. find_var(data, "BEFRAGTE(R)", fixed = TRUE) for label text with parentheses. Default FALSE: regular expression. A pattern that is not a valid regular expression (e.g. "(") is searched as literal text automatically, and a message points to fixed = TRUE when the literal text would give other matches.

Details

When to Use This:

- You imported an SPSS file and need to find which variable contains "trust" or "satisfaction"
- You want to quickly identify all variables related to a topic
- You know the German/English label text but not the variable code

Pattern Matching:

The pattern argument supports regular expressions. Matching is case-insensitive. Some examples:

- "trust" — matches "trust", "Trust", "distrust", "trustworthy"
- "^trust" — matches only names/labels starting with "trust"
- "^q[0-9]+" — matches variable names like q1, q23, q104
- "zufried" — finds German labels containing "Zufriedenheit"

Label text often contains characters with a special meaning in regular expressions, such as parentheses: "BEFRAGTE(R)" as a regular expression matches "BEFRAGTER". Use fixed = TRUE to search for the text exactly as written.

Value

A data frame with columns:

col Column position in the data

name Variable name

label Variable label (or "" if none)

Without matches, an empty data frame is returned invisibly (with a message).

See Also

var_label() for getting/setting variable labels, val_labels() for value labels

Other labels: copy_labels(), drop_labels(), set_na(), to_character(), to_label(), to_labelled(), to_numeric(), unlabel(), val_labels(), var_label()

Examples

```
library(dplyr)
data(survey_data)

# Find all variables related to "trust"
find_var(survey_data, "trust")

# Search only in variable labels
find_var(survey_data, "satisfaction", search = "label")

# Use regex to find numbered items
find_var(survey_data, "^q[0-9]+", search = "name")

# Literal text with regex characters, e.g. parentheses in a label
find_var(survey_data, "(1=left", fixed = TRUE)
```

---

fisher_test *Fisher's Exact Test for Small Samples*

Description

fisher_test() performs Fisher's exact test for independence in a contingency table. Use this instead of chi_square() when your sample is small or when expected cell frequencies are below 5.

Think of it as:

- An exact alternative to the chi-square test of independence
- Best for small samples where the chi-square approximation may be inaccurate
- SPSS automatically reports Fisher's Exact Test for 2x2 tables

The test tells you:

- Whether two categorical variables are related (exact p-value)
- The contingency table of observed frequencies
- Results that are valid even with very small samples

Usage

```
fisher_test(
  data,
  row,
  col,
  weights = NULL,
  simulate.p.value = FALSE,
  B = 10000,
  ...
)
```

Arguments

data	Your survey data (data frame or tibble)
row	The row variable (categorical)
col	The column variable (categorical)
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
simulate.p.value	Compute the p-value by Monte Carlo simulation instead of exactly? (Default: FALSE). For large tables (many rows and columns, large N) the exact network algorithm can run out of memory ("FEXACT error"); fisher_test() then switches to the Monte Carlo p-value automatically with a warning. SPSS offers the same choice (Exact Tests: "Exact" or "Monte Carlo"). Call set.seed() first for reproducible Monte Carlo results.
B	Number of Monte Carlo replicates (Default: 10000, the SPSS default number of samples).
...	Additional arguments (currently unused)

Details

Understanding the Results:

P-value: If p < 0.05, the variables are likely related (not independent)

- p < 0.001: Very strong evidence of relationship
- p < 0.01: Strong evidence of relationship
- p < 0.05: Moderate evidence of relationship
- p >= 0.05: No significant relationship found

Unlike the chi-square test which uses a large-sample approximation, Fisher's exact test computes the exact probability of observing the given table (or a more extreme one) under the null hypothesis of independence.

For tables larger than 2x2, the Fisher-Freeman-Halton extension is used.

When to Use This:

Use Fisher's exact test when:

- Any expected cell frequency is less than 5
- Your total sample size is small (typically N < 30)
- You have a 2x2 table (Fisher's exact is standard here)
- You want an exact p-value rather than an approximation

Relationship to Other Tests:

- For large samples with all expected frequencies >= 5: Use `chi_square()` instead
- For paired binary data (before/after): Use `mcnemar_test()` instead
- For testing a single proportion: Use `binomial_test()` instead

SPSS Equivalent:

SPSS: `CROSSTABS /STATISTICS=CHISQ` (Fisher's exact test is automatically reported for 2x2 tables)

Value

Test results showing whether two categorical variables are related, including:

- Exact p-value (two-sided)
- Contingency table of observed frequencies
- Sample size (N)

References

Fisher, R. A. (1922). On the interpretation of chi-square from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1), 87-94.

Freeman, G. H., & Halton, J. H. (1951). Note on an exact treatment of contingency, goodness of fit and other problems of significance. *Biometrika*, 38(1/2), 141-149.

See Also

`chi_square` for chi-square test of independence (large samples).

`mcnemar_test` for paired proportions.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic 2x2 Fisher test
survey_data %>%
  fisher_test(row = gender, col = region)

# Fisher test for larger table
```

```
survey_data %>%
  fisher_test(row = gender, col = interview_mode)

# With weights
survey_data %>%
  fisher_test(row = gender, col = region, weights = sampling_weight)

# Grouped analysis
survey_data %>%
  group_by(education) %>%
  fisher_test(row = gender, col = region)
```

`fitted.linear_regression` *Fitted values of a linear_regression model*

Description

Dispatches to `stats::fitted()` for the fitted `lm`; grouped and pairwise results raise an informative error.

Usage

```
## S3 method for class 'linear_regression'
fitted(object, ...)
```

Arguments

`object`	A `linear_regression` result (ungrouped, listwise).
`...`	Passed to the `lm` method.

Value

A numeric vector.

`frequency` *Count How Many People Chose Each Option*

Description

`frequency()` helps you understand categorical data by showing how many people chose each option. It's perfect for survey questions with fixed choices like education level, yes/no questions, or rating scales.

Think of it as creating a summary table that shows:

- How many people chose each option
- What percentage that represents
- Running totals to see cumulative patterns

Usage

```
frequency(
  data,
  ...,
  weights = NULL,
  sort_frq = "none",
  show_na = TRUE,
  show_prc = TRUE,
  show_valid = TRUE,
  show_sum = TRUE,
  show_labels = "auto",
  show_unused = FALSE,
  sort.frq = NULL,
  show.na = NULL,
  show.prc = NULL,
  show.valid = NULL,
  show.sum = NULL,
  show.labels = NULL,
  show.unused = NULL
)

fre(data, ..., weights = NULL, sort_frq = "none", show_na = TRUE,
  show_prc = TRUE, show_valid = TRUE, show_sum = TRUE, show_labels = "auto",
  show_unused = FALSE, sort.frq = NULL, show.na = NULL, show.prc = NULL,
  show.valid = NULL, show.sum = NULL, show.labels = NULL, show.unused = NULL)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The categorical variables you want to analyze. You can list multiple variables separated by commas, or use helpers like `starts_with("trust")`
weights	Optional survey weights for population-representative results. Without weights, you get sample frequencies. With weights, you get population estimates. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
sort_frq	How to order the results: • `"none"` (default): Keep original order • `"asc"`: Sort from lowest to highest frequency • `"desc"`: Sort from highest to lowest frequency
show_na	Include missing values in the table? (Default: TRUE)
show_prc	Show raw percentages including missing values? (Default: TRUE)
show_valid	Show percentages excluding missing values? (Default: TRUE). The cumulative percentages are cumulative valid percentages, so `show_valid = FALSE` hides them too.
show_sum	Show the cumulative (valid) percentages? (Default: TRUE)

`show_labels` Show category labels if available? (Default: "auto" - shows labels when they exist)

`show_unused` Show all defined value labels, even those with zero observations? (Default: FALSE). When TRUE, values that have labels defined (e.g., from statistical software files) but no cases in the data are included with frequency 0. This is useful for labelled datasets where unused categories should still appear in the output. The same applies to empty factor levels, which are hidden by default (as SPSS lists only observed values). Automatically enables label display.

`sort.frq`, `show.na`, `show.prc`, `show.valid`, `show.sum`, `show.labels`, `show.unused`

Defunct dot-case argument names, removed in mariposa 0.6.9. Calling the function with any of them is an error; use the snake_case equivalents instead. (The formals are retained only so that the old names error clearly instead of being swallowed by ....)

Details

Understanding the Results:

The frequency table follows the SPSS FREQUENCIES layout:

- **N**: Number of responses in each category (weighted: sum of weights, displayed rounded)
- **Raw %**: Percentage including missing values (use for "response rate")
- **Valid %**: Percentage excluding missing values (use for "among those who answered")
- **Cum. %**: Running total of the valid percentages (helps identify cutoff points)
- **Total valid**, the missing categories, **Total missing** (with two or more missing categories) and the grand **Total**; without missing values a single **Total** row ends the table.

Factors, character and logical variables show their categories in the Value column; labelled numeric variables show the code and its label. Long labels are never cut; they wrap when the table would be wider than the console.

When to Use This:

Use `frequency()` when you have:

- Categorical variables (gender, region, education level)
- Yes/No questions
- Rating scales (satisfied/neutral/dissatisfied)
- Any question with a fixed set of options

Weights Make a Difference:

Without weights, you're describing your sample. With weights, you're estimating population values. Always use weights for population inference.

Tagged Missing Values:

When data is imported with tagged NAs (e.g., via `read_spss()` with `tag_na = TRUE`, or `read_stata()`, `read_sas()`, `read_xpt()` with the `tag_na` parameter), `frequency()` automatically expands the missing value section to show each missing type individually (with its original missing value code and label), plus summary rows for **Total Valid** and **Total Missing**.

Value

A frequency table showing counts and percentages for each category

See Also

table for base R frequency tables.

crosstab for cross-tabulation of two variables.

chi_square for testing relationships between categories.

describe for numeric variable summaries.

Other descriptive: crosstab(), describe(), multiple_response(), normality_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic categorical analysis
survey_data %>% frequency(gender)

# Multiple variables with weights
survey_data %>% frequency(gender, region, weights = sampling_weight)

# Grouped analysis by region
survey_data %>%
  group_by(region) %>%
  frequency(gender, weights = sampling_weight)

# Education levels with sorting
survey_data %>% frequency(education, sort_frq = "desc")

# Employment status with custom display options
survey_data %>% frequency(employment, weights = sampling_weight,
                          show_na = TRUE, show_sum = TRUE)
```

---

friedman_test *Compare Three or More Related Measurements Without Assuming Normality*

Description

friedman_test() compares three or more related measurements from the same subjects when your data isn't normally distributed. It's the non-parametric alternative to repeated-measures ANOVA.

Think of it as:

- Comparing ratings of multiple items by the same respondents

- Testing whether scores change across three or more time points
- A robust repeated-measures comparison that works with any data shape

The test tells you:

- Whether at least one measurement is significantly different from the others
- Which measurements tend to be rated higher or lower (via mean ranks)
- The strength of the overall effect (Kendall's W)

Usage

```
friedman_test(data, ..., weights = NULL, conf.level = 0.95)
```

Arguments

`data`	Your survey data (a data frame or tibble) in wide format, with one row per subject and each measurement in a separate column
`...`	The measurement variables to compare (at least 3). You can list them individually or use helpers like `starts_with("trust_")`
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`conf.level`	Not used. Rank tests report no confidence interval (SPSS `NPAR TESTS` neither); kept for backward compatibility.

Details

Understanding the Results:

P-value: If $p < 0.05$, at least one measurement is significantly different

- $p < 0.001$: Very strong evidence of differences
- $p < 0.01$: Strong evidence of differences
- $p < 0.05$: Moderate evidence of differences
- $p > 0.05$: No significant differences found

Kendall's W (Effect size: How consistent is the pattern?):

- < 0.1: Negligible agreement/effect
- 0.1 - 0.3: Weak agreement
- 0.3 - 0.5: Moderate agreement
- 0.5 or higher: Strong agreement

Mean Ranks:

- Higher mean rank = measurement tends to have higher values
- Lower mean rank = measurement tends to have lower values
- Compare mean ranks to see which measurements stand out

When to Use This:

Use the Friedman test when:

- You have 3 or more related measurements from the same subjects
- Your data is not normally distributed
- You have ordinal data (ratings, rankings)
- You want a robust alternative to repeated-measures ANOVA
- You're comparing multiple ratings by the same respondents

Relationship to Other Tests:

- For 2 related measurements: Use `wilcoxon_test()` instead
- For normally distributed repeated measures: Use repeated-measures ANOVA

Weighted analyses:

Weights are treated as frequency weights. With all weights equal to 1 the weighted test reproduces the unweighted one exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- For independent groups: Use `kruskal_wallis()` instead

Value

Test results showing whether the measurements differ, including:

- Chi-Square statistic (`chi_squared`, the Friedman test statistic)
- Degrees of freedom (number of measurements minus 1)
- P-value (are measurements different?)
- Kendall's W (effect size: how strong is the overall pattern?)
- Mean rank for each measurement (which measurements are higher/lower?)

References

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 32(200), 675-701.

Kendall, M. G., & Babington Smith, B. (1939). The problem of m rankings. The Annals of Mathematical Statistics, 10(3), 275-287.

See Also

`friedman.test` for the base R Friedman test.

`wilcoxon_test` for comparing two related measurements.

`kruskal_wallis` for comparing independent groups.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Compare three trust items (rated by same respondents)
survey_data %>%
  friedman_test(trust_government, trust_media, trust_science)

# Using tidyselect helpers
survey_data %>%
  friedman_test(starts_with("trust_"))

# Weighted analysis
survey_data %>%
  friedman_test(trust_government, trust_media, trust_science,
                weights = sampling_weight)

# Grouped analysis (separate test per region)
survey_data %>%
  group_by(region) %>%
  friedman_test(trust_government, trust_media, trust_science)
```

---

kendall_tau *Kendall's Tau Correlation Analysis*

Description

Calculates Kendall's tau-b rank correlation coefficients between variables with support for weighted correlations, grouped data, and multiple variable pairs. Provides significance testing and SPSS-compatible output formatting.

The function computes tau-b, which is adjusted for ties and is particularly suitable for ordinal data or when the assumptions of Pearson correlation are not met. For weighted correlations, it uses survey-weighted rank calculations.

Usage

```
kendall_tau(
  data,
  ...,
  weights = NULL,
  alternative = c("two.sided", "less", "greater"),
  use = c("pairwise", "listwise"),
  na.rm = NULL
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The variables you want to correlate. List two for a single correlation or more for a correlation matrix. You can use helpers like `starts_with("trust")`. Numeric variables or ordered factors (ranked by their level order).
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`alternative`	Direction of the test: • `"two.sided"` (default): Two-tailed test • `"less"`: One-tailed test (negative correlation) • `"greater"`: One-tailed test (positive correlation)
`use`	How to handle missing values: • `"pairwise"` (default): Pairwise deletion - each correlation uses all available cases • `"listwise"`: Listwise deletion - only complete cases across all variables
`na.rm`	Deprecated. Use `use` instead.

Details

Understanding the Results:

Kendall's tau measures how often pairs of observations are in the same order (concordant) versus different order (discordant). It is particularly useful for ordinal data, data with outliers, small sample sizes, and non-linear but monotonic relationships.

The tau value ranges from -1 to +1:

- **Strong positive** (0.5 to 1.0): High values of one variable tend to go with high values of the other
- **Moderate positive** (0.3 to 0.5): Some tendency for values to increase together
- **Weak positive** (0 to 0.3): Slight tendency for values to increase together
- **No correlation** (near 0): No relationship between the variables
- **Negative values**: As one variable increases, the other tends to decrease

The output also provides:

- **p-value**: Probability of seeing this correlation by chance (smaller = stronger evidence)
- **n**: Number of observations used
- **significance stars**: Quick visual indicator of statistical significance

When to Use This:

Choose Kendall's tau when:

- Your data is ordinal (ranked categories)
- You have a small sample size (< 30 observations)
- Your data has outliers that might affect Pearson correlation

Weighted analyses:
The weighted tau-b is an exact frequency-weighted computation and reproduces the unweighted tau when all weights equal 1; its z statistic and p-value use a normal approximation without tie correction. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- You want a more conservative measure than Spearman's rho

Value

Correlation results showing rank-based relationships between variables, including the tau-b coefficient, p-value, z-score, and sample size for each pair. For multiple variables, correlation, significance, and sample size matrices are also provided. Use `summary()` for the full SPSS-style output with toggleable sections.

References

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, 30(1/2), 81–93.

Kendall, M. G. (1945). The treatment of ties in ranking problems. *Biometrika*, 33(3), 239–251.

Agresti, A. (2010). *Analysis of Ordinal Categorical Data* (2nd ed.). John Wiley & Sons.

See Also

`cor` with `method = "kendall"` for the base R implementation.

`spearman_rho` for Spearman's rank correlation.

`pearson_cor` for Pearson correlation analysis.

`summary.kendall_tau` for detailed output with toggleable sections.

Other correlation: `partial_cor()`, `pearson_cor()`, `spearman_rho()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Kendall's tau is O(n^2); the examples run on a subset for speed.
# On your own data, simply drop the subsetting.
mini_survey <- survey_data[1:300, ]

# Basic correlation between two variables
mini_survey %>%
  kendall_tau(life_satisfaction, political_orientation)

# Correlation matrix for multiple variables
mini_survey %>%
  kendall_tau(life_satisfaction, political_orientation, trust_media)

# Weighted correlations
mini_survey %>%
  kendall_tau(age, income, weights = sampling_weight)
```

```
# Listwise deletion for missing data
mini_survey %>%
  kendall_tau(age, income, use = "listwise")

# One-tailed test
mini_survey %>%
  kendall_tau(age, income, alternative = "greater")


# Grouped correlations
mini_survey %>%
  group_by(region) %>%
  kendall_tau(age, income, life_satisfaction)

# Using tidyselect helpers for ordinal variables
mini_survey %>%
  kendall_tau(starts_with("trust"), weights = sampling_weight)

# --- Three-layer output ---
result <- mini_survey %>%
  kendall_tau(life_satisfaction, political_orientation, trust_media,
              weights = sampling_weight)
result              # compact one-line overview
summary(result)     # full correlation, p-value, and N matrices
summary(result, pvalue_matrix = FALSE)  # hide p-values
```

`kruskal_wallis` *Compare Multiple Groups Without Assuming Normal Data*

Description

`kruskal_wallis()` compares three or more groups when your data isn't normally distributed or when you have ordinal data (like ratings or rankings). It's the non-parametric alternative to one-way ANOVA.

Think of it as:

- An extension of the Mann-Whitney test for more than two groups
- A robust way to compare groups that works with any data shape
- Perfect for Likert scales, rankings, or skewed distributions

The test tells you:

- Whether at least one group is different from the others
- How strong the overall group effect is (effect size)
- Which groups tend to have higher or lower values (via mean ranks)

Usage

```
kruskal_wallis(data, ..., group, weights = NULL, conf.level = 0.95)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The variables you want to compare between groups. You can list multiple variables or use helpers like `starts_with("satisfaction")`
`group`	The categorical variable that defines your groups (e.g., education, employment). Must have at least 2 groups (3+ for meaningful use).
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`conf.level`	Not used. Rank tests report no confidence interval (SPSS `NPAR TESTS` neither); kept for backward compatibility.

Details

Understanding the Results:

P-value: If $p < 0.05$, at least one group is significantly different

- $p < 0.001$: Very strong evidence of group differences
- $p < 0.01$: Strong evidence of group differences
- $p < 0.05$: Moderate evidence of group differences
- $p > 0.05$: No significant group differences found

Effect Size Epsilon-squared (How much do groups matter?):

- < 0.01: Negligible effect
- 0.01-0.06: Small effect
- 0.06-0.14: Medium effect
- 0.14 or higher: Large effect

Mean Ranks:

- Higher mean rank = group tends to have higher values
- Lower mean rank = group tends to have lower values
- Compare mean ranks to see the pattern of group differences

When to Use This:

Use Kruskal-Wallis test when:

- Your data is not normally distributed (skewed, outliers)
- You have ordinal data (rankings, Likert scales)
- Sample sizes are small or very unequal across groups
- You want a robust alternative to one-way ANOVA
- You're comparing satisfaction ratings, income, or other skewed variables

What Comes Next?:

If the Kruskal-Wallis test is significant:

1. Look at mean ranks to see the pattern
2. Use pairwise Mann-Whitney tests with Bonferroni correction to find which specific groups differ
3. Consider effect sizes to judge practical importance

Relationship to Other Tests:

- For 2 groups: Use `mann_whitney()` instead
- For normally distributed data: Use `oneway_anova()` instead

Weighted analyses:

Weights are treated as frequency weights. With all weights equal to 1 the weighted test reproduces the unweighted one exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- For repeated measures (same subjects): Use `friedman_test()` instead

Value

Test results showing whether groups differ, including:

- H statistic (Kruskal-Wallis chi-square test statistic)
- Degrees of freedom (number of groups minus 1)
- P-value (are groups different?)
- Effect size epsilon-squared (how big is the group effect?)
- Mean rank for each group (which groups are higher/lower?)

References

Kruskal, W. H., & Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association, 47(260), 583-621.

Tomczak, M., & Tomczak, E. (2014). The need to report effect size estimates revisited. An overview of some recommended measures of effect size. Trends in Sport Sciences, 1(21), 19-25.

See Also

`kruskal.test` for the base R Kruskal-Wallis test.

`mann_whitney` for comparing exactly two groups.

`oneway_anova` for parametric one-way ANOVA.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic Kruskal-Wallis test (comparing across education levels)
survey_data %>%
  kruskal_wallis(life_satisfaction, group = education)

# Multiple variables
survey_data %>%
  kruskal_wallis(life_satisfaction, income, trust_government,
                 group = education)

# Using tidyselect helpers
survey_data %>%
  kruskal_wallis(starts_with("trust_"), group = education)

# Weighted analysis
survey_data %>%
  kruskal_wallis(life_satisfaction, group = education,
                 weights = sampling_weight)

# Grouped analysis (separate test for each region)
survey_data %>%
  group_by(region) %>%
  kruskal_wallis(life_satisfaction, group = education)

# Compare across employment status (5 groups)
survey_data %>%
  kruskal_wallis(income, group = employment)
```

---

levene_test *Test If Groups Vary Similarly*

Description

`levene_test()` checks if different groups have similar amounts of variation. This is an important assumption for many statistical tests - groups should spread out in similar ways.

The test tells you:

- Whether variance is consistent across groups
- If you can trust standard ANOVA and t-test results
- When to use alternative tests that don't assume equal variance

Usage

```
levene_test(x, ...)

## Default S3 method:
levene_test(x, ...)

## S3 method for class 'data.frame'
levene_test(x, ..., group, weights = NULL, center = c("mean", "median"))

## S3 method for class 'oneway_anova'
levene_test(x, center = c("mean", "median"), ...)

## S3 method for class 't_test'
levene_test(x, center = c("mean", "median"), ...)

## S3 method for class 'ancova'
levene_test(x, center = c("mean", "median"), ...)

## S3 method for class 'mann_whitney'
levene_test(x, ...)

## S3 method for class 'grouped_df'
levene_test(x, ..., group, weights = NULL, center = c("mean", "median"))
```

Arguments

x	Either your data (a data frame, optionally grouped with group_by()) or test results from t_test(), oneway_anova(), factorial_anova() or ancova()
...	Variables to test (when using a data frame). List several variables or use tidyselect helpers like starts_with("trust").
group	The grouping variable for comparison (unquoted or a string)
weights	Optional survey weights for population-representative results. Must be numeric and non-negative (the package-wide weights policy). Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
center	How to measure center: "mean" (default) or "median" (more robust). For ancova() results only "mean" is available: as in SPSS UNIANOVA, the test is computed on the absolute residuals of the ANCOVA model (covariates and factors), compared across the cells of the design.

Details

Understanding the Results:

P-value interpretation:

- p > 0.05: Good! Groups have similar variance (assumption met)
- p <= 0.05: Problem - groups vary differently (assumption violated)

Think of it like checking if all groups are equally "spread out":

- Similar spread = can use standard tests
- Different spread = need special methods

When to Use This:

Check variance equality when:

- Before running t-tests or ANOVA
- Comparing groups with different sizes
- Your statistical test assumes equal variances
- You see very different standard deviations

What If Variances Are Unequal?:

If Levene's test is significant (p <= 0.05):

- For t-tests: Use Welch's t-test (var.equal = FALSE)
- For ANOVA: Use Welch's ANOVA
- Consider transforming your data
- Use non-parametric alternatives
- Report that equal variance assumption was violated

Usage Flexibility:

You can use this function two ways:

- **Standalone**: Check any variables for equal variance
- **After tests**: Pipe after t_test() or oneway_anova() to verify assumptions

Tips for Success:

- Always check this assumption for group comparisons
- Visual inspection (boxplots) can supplement the test
- Large samples make the test very sensitive
- Use median-based test for skewed data (center = "median")
- Don't panic if violated - alternatives exist!

Value

Test results showing:

- Whether groups have equal variances (p-value)
- F-statistic measuring variance differences
- Which variables meet the assumption

References

Levene, H. (1960). Robust tests for equality of variances. In I. Olkin (Ed.), *Contributions to Probability and Statistics* (pp. 278–292). Stanford University Press.

Brown, M. B., & Forsythe, A. B. (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346), 364–367.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

oneway_anova for one-way ANOVA (which assumes equal variances).

t_test for group mean comparisons.

var.test for the base R F-test of variance equality.

Other posthoc: dunn_test(), pairwise_wilcoxon(), scheffe_test(), tukey_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Standalone Levene test (test homogeneity of variances)
survey_data %>% levene_test(life_satisfaction, group = region)

# Multiple variables
survey_data %>% levene_test(life_satisfaction, trust_government, group = region)

# Weighted analysis
survey_data %>% levene_test(income, group = education, weights = sampling_weight)

# Piped after ANOVA (common workflow)
result <- survey_data %>%
  oneway_anova(life_satisfaction, group = education)
result %>% levene_test()

# Piped after t-test
survey_data %>%
  t_test(age, group = gender) %>%
  levene_test()

# Using mean instead of median as center
survey_data %>% levene_test(income, group = region, center = "mean")
```

---

linear_regression *Run a Linear Regression*

Description

linear_regression() performs bivariate or multiple linear regression with SPSS-compatible output. Wraps stats::lm() and adds standardized coefficients (Beta), a formatted ANOVA table, and a model summary matching SPSS REGRESSION output.

Supports two interface styles:

- **Formula interface:** linear_regression(data, life_satisfaction ~ age + education)
- **SPSS-style:** linear_regression(data, dependent = life_satisfaction, predictors = c(age, education))

Usage

```
linear_regression(
  data,
  formula = NULL,
  dependent = NULL,
  predictors = NULL,
  weights = NULL,
  use = c("listwise", "pairwise"),
  standardized = TRUE,
  conf.level = 0.95,
  factors = c("dummy", "numeric")
)
```

Arguments

data	Your survey data (a data frame or tibble). If grouped (via dplyr::group_by()), separate regressions are run for each group.
formula	A formula specifying the model (e.g., y ~ x1 + x2). If provided, dependent and predictors are ignored. The outcome may be transformed (log(income) ~ age); y ~ . uses all other columns except the weights and grouping variables.
dependent	The dependent variable (unquoted). Used with predictors when no formula is given.
predictors	Predictor variable(s) (unquoted, supports tidyselect). Used with dependent when no formula is given. The dependent, weights and grouping variables are never used as predictors (a selection such as where(is.numeric) drops them with a message). Character predictors are entered as factors.
weights	Optional survey weights. When specified, weighted least squares (WLS) is used, matching SPSS WEIGHT BY. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
use	How to handle missing data: "listwise" (default) drops any case with a missing value on any variable (matching SPSS /MISSING LISTWISE). "pairwise" computes the regression from a pairwise covariance/correlation matrix, retaining more cases (matching SPSS /MISSING PAIRWISE).
standardized	Logical. If TRUE (default), standardized coefficients (Beta) are calculated and included in the output.
conf.level	Confidence level for coefficient intervals (default 0.95).
factors	How factor predictors are entered into the model: "dummy" (default, matches base R lm()) expands a factor with L levels into L - 1 contrasts; "numeric" silently coerces factor levels to their integer codes, matching SPSS REGRESSION default behavior (ordinal-as-scale). The "numeric" mode emits a one-line cli::cli_inform() listing the coerced variables. The "numeric" mode is required to reproduce SPSS results when factor predictors carry ordered meaning (e.g., 4-level education). Note that for *ordered* factors, "dummy" applies R's default polynomial contrasts (terms suffixed .L, .Q, .C), not treatment dummies; convert with factor(x, ordered = FALSE) first if you want dummy coding. factors applies to factors

only: labelled predictors from SPSS files (haven_labelled) are numeric and enter with their numeric codes, exactly as in SPSS REGRESSION; summary() notes them. Convert them with to_label first to get dummy coding.

Details

Understanding the Results:

The output includes four sections matching SPSS REGRESSION output:

- **Model Summary**: R, R-squared, Adjusted R-squared, and Standard Error of the Estimate. R-squared tells you how much variance in the dependent variable is explained by the predictors.
- **ANOVA**: Tests whether the overall model is significant. A significant F-test means at least one predictor matters.
- **Coefficients**: B (unstandardized), Beta (standardized), t-value, p-value, and confidence intervals for each predictor.
- **Descriptives**: Mean, SD, and N for all variables in the model.

Interpreting coefficients:

- **B (unstandardized)**: For each 1-unit increase in the predictor, the dependent variable changes by B units
- **Beta (standardized)**: Allows comparison across predictors with different scales. Larger absolute Beta = stronger effect
- **p-value**: Values below 0.05 indicate statistically significant predictors

When to Use This:

Use linear_regression() when:

- Your dependent variable is continuous (e.g., income, satisfaction score)
- You want to predict an outcome from one or more predictors
- You need standardized coefficients to compare predictor importance

For binary outcomes (yes/no, 0/1), use logistic_regression instead.

Technical Details:

Missing Data: By default, listwise deletion is used (matching SPSS REGRESSION /MISSING LISTWISE). Set use = "pairwise" to match SPSS /MISSING PAIRWISE, which computes the regression from a pairwise covariance matrix. Pairwise deletion retains more cases and produces results closer to SPSS output when data has varying patterns of missingness.

Weights: When weights are specified, they are treated as frequency weights (matching SPSS WEIGHT BY behavior). The model is fitted using weighted least squares via lm(weights = ...); the coefficients are those of lm(), but N is sum(w) and the residual df are sum(w) - rank (unrounded), as in SPSS. lm() itself treats weights as analytic (precision) weights with df = cases - rank, so the inherited generics are adjusted to the SPSS convention: vcov(), confint(), nobs() (unrounded sum(w)), df.residual(), anova() (single model), predict() (standard errors and intervals), broom::tidy() and broom::glance() all agree with summary(). stats::summary.lm(), logLik(), AIC() and BIC() keep lm's analytic-weight definitions.

Standardized Coefficients: Beta = B * (SD_x / SD_y). This matches the SPSS standardized coefficient output. Not available for the intercept. For dummy-coded factor terms (factors = "dummy"), the SD of the contrast column from the design matrix is used.

Factor Predictors: By default (`factors = "dummy"`), factor predictors are expanded into `L - 1` contrasts via R's `stats::model.matrix()`, matching base R `lm()`: unordered factors get treatment (dummy) contrasts against the first level, while *ordered* factors get R's default polynomial contrasts (`.L`/`.Q`/`.C` terms). Pass `factors = "numeric"` to silently coerce factor levels to their integer codes (SPSS `REGRESSION` default). The "numeric" mode is required to reproduce SPSS results for ordinal predictors like education or Likert scales that SPSS treats as continuous.

Grouped Analysis: When `data` is grouped via `dplyr::group_by()`, a separate regression is run for each group (matching SPSS SPLIT FILE BY).

Value

For ungrouped + listwise data, an object of class `c("linear_regression", "lm")` — **the fitted `lm` itself**, with mariposa-specific slots attached:

coef_table SPSS-style tibble with B, Std.Error, Beta, t, p, CI_lower, CI_upper. For weighted models, SE / t / p are adjusted to SPSS's frequency-weight df (see Technical Details).

anova_table SPSS-style overall-model ANOVA tibble (Source × Sum_of_Squares / df / Mean_Square / F_statistic / Sig).

model_summary List with R, R_squared, adj_R_squared, std_error.

descriptives Tibble with Mean, Std.Deviation, N for all variables.

n Sample size (listwise complete cases; weighted N when weighted).

formula, dependent, predictor_names, weighted, weight_name, use, is_grouped, standardized, conf.level Call metadata.

Because the object inherits from `"lm"`, all standard generics (`predict()`, `anova()`, `vcov()`, `confint()`, `residuals()`, `fitted()`, `coef()`, `model.matrix()`, `broom::tidy()`, `broom::glance()`, `broom::augment()`) dispatch natively without unwrapping. `summary()` returns the SPSS-style mariposa summary; for the raw lm summary use `stats::summary.lm()` on the same object.

For `use = "pairwise"` (no single fitted lm available) or for grouped data, returns a list of class `"linear_regression"`. Pairwise results expose the same SPSS-style tables but not the lm generics; grouped results hold one fitted lm-inheriting model per group under `$groups`.

See Also

`logistic_regression` for binary outcome variables.

`describe` for checking variable distributions before regression.

`pearson_cor` for checking bivariate correlations.

`summary.linear_regression` for detailed output with toggleable sections.

Other regression: `logistic_regression()`, `marginal_effects()`

Examples

```
library(dplyr)
data(survey_data)

# Bivariate regression
linear_regression(survey_data, life_satisfaction ~ age)
```

```
# Multiple regression
linear_regression(survey_data, income ~ age + education + life_satisfaction)

# SPSS-style interface
linear_regression(survey_data,
                  dependent = life_satisfaction,
                  predictors = c(trust_government, trust_media, trust_science))

# Weighted regression
linear_regression(survey_data, life_satisfaction ~ age, weights = sampling_weight)

# Grouped by region
survey_data |>
  dplyr::group_by(region) |>
  linear_regression(life_satisfaction ~ age)

# Factor predictors: dummy-coding (default, matches base R lm())
linear_regression(survey_data, income ~ age + education)

# Factor predictors: SPSS-style ordinal-as-scale
linear_regression(survey_data, income ~ age + education,
                  factors = "numeric")

# --- Three-layer output ---
result <- linear_regression(survey_data, life_satisfaction ~ age + income)
result                                  # compact one-line overview
summary(result)                         # full detailed SPSS-style output
summary(result, descriptives = FALSE)   # hide descriptives section
```

`logistic_regression` *Run a Logistic Regression*

Description

`logistic_regression()` performs binary logistic regression with SPSS-compatible output. Wraps `stats::glm(family = binomial)` and adds odds ratios, pseudo R-squared measures, classification table, and model tests matching SPSS LOGISTIC REGRESSION output.

Supports two interface styles:

- **Formula interface:** `logistic_regression(data, high_satisfaction ~ age + income)`
- **SPSS-style:** `logistic_regression(data, dependent = high_satisfaction, predictors = c(age, income))`

Usage

```
logistic_regression(
  data,
```

```
  formula = NULL,
  dependent = NULL,
  predictors = NULL,
  weights = NULL,
  conf.level = 0.95,
  factors = c("dummy", "numeric")
)
```

Arguments

data	Your survey data (a data frame or tibble). If grouped (via dplyr::group_by()), separate regressions are run for each group.
formula	A formula specifying the model (e.g., y ~ x1 + x2). If provided, dependent and predictors are ignored.
dependent	The dependent variable (unquoted). Used with predictors when no formula is given. Must have exactly two distinct values (e.g. 0/1, 1/2, a two-level factor); see Technical Details.
predictors	Predictor variable(s) (unquoted, supports tidyselect). Used with dependent when no formula is given. The dependent, weights and grouping variables are never used as predictors (a selection such as where(is.numeric) drops them with a message). Character predictors are entered as factors.
weights	Optional survey weights. When specified, weighted maximum likelihood estimation is used, matching SPSS WEIGHT BY behavior. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
conf.level	Confidence level for odds ratio intervals (default 0.95).
factors	How factor predictors are entered into the model: "dummy" (default, matches base R glm()) expands a factor with L levels into L - 1 contrasts; "numeric" silently coerces factor levels to their integer codes, matching SPSS LOGISTIC REGRESSION default behavior when no /CATEGORICAL subcommand is given. The "numeric" mode emits a one-line cli::cli_inform() listing the coerced variables. Note that for *ordered* factors, "dummy" applies R's default polynomial contrasts (terms suffixed .L, .Q, .C), not treatment dummies; convert with factor(x, ordered = FALSE) first if you want dummy coding. factors applies to factors only: labelled predictors from SPSS files (haven_labelled) are numeric and enter with their numeric codes, as in SPSS LOGISTIC REGRESSION without /CATEGORICAL; summary() notes them. Convert them with to_label first to get dummy coding.

Details

Understanding the Results:

The output includes five sections matching SPSS LOGISTIC REGRESSION output:

- **Omnibus Test**: Tests whether the model as a whole is significant. A significant chi-square means the model predicts better than chance.
- **Model Summary**: -2 Log Likelihood and pseudo R-squared values. Lower -2LL = better fit. Higher R-squared = more variance explained.

- **Hosmer-Lemeshow Test**: Goodness-of-fit test. A non-significant result ($p > 0.05$) means the model fits the data well.
- **Classification Table**: How well the model classifies cases. Shows percentage correctly predicted for each group and overall.
- **Coefficients**: B, Wald test, odds ratios (Exp(B)), and CIs.

Interpreting odds ratios (Exp(B)):

- **Exp(B) > 1**: Predictor increases the odds of the outcome
- **Exp(B) < 1**: Predictor decreases the odds of the outcome
- **Exp(B) = 1**: Predictor has no effect on the odds

When to Use This:

Use `logistic_regression()` when:

- Your dependent variable is binary (yes/no, 0/1, pass/fail)
- You want to predict group membership from one or more predictors
- You need odds ratios to interpret predictor effects

For continuous outcomes, use `linear_regression` instead.

Technical Details:

Dependent Variable: Any variable with exactly two distinct observed values, coded internally as 0/1 like SPSS LOGISTIC REGRESSION does: for numeric and labelled variables the lower value becomes 0 and the higher 1 (so 1/2 and 0/1 codings give the same model); for factors unused levels are dropped and the first remaining level becomes 0; character values are ordered alphabetically; logicals code `FALSE` = 0, `TRUE` = 1. The model predicts the probability of the category coded 1 - the compact print names it (`[P(y = category)]`), and `summary()` shows the SPSS "Dependent Variable Encoding" table (also stored as `$dv_encoding`). A grouped analysis uses one encoding for all groups.

Missing Data: Listwise deletion is used (matching SPSS LOGISTIC REGRESSION default behavior).

Weights: When weights are specified, they are treated as frequency weights (matching SPSS WEIGHT BY behavior).

Pseudo R-squared: Three measures are reported:

- Cox & Snell R-squared (bounded below 1)
- Nagelkerke R-squared (adjusted to reach 1)
- McFadden R-squared (1 - LL_model/LL_null)

Factor Predictors: By default (`factors = "dummy"`), factor predictors are expanded into `L - 1` contrasts via R's `stats::model.matrix()`, matching base R `glm()`: unordered factors get treatment (dummy) contrasts against the first level, while *ordered* factors get R's default polynomial contrasts (`.L`/`.Q`/`.C` terms). Pass `factors = "numeric"` to silently coerce factor levels to their integer codes (SPSS `LOGISTIC REGRESSION` default without an explicit `/CATEGORICAL` subcommand).

Grouped Analysis: When `data` is grouped via `dplyr::group_by()`, a separate regression is run for each group (matching SPSS SPLIT FILE BY).

Value

For ungrouped data, an object of class `c("logistic_regression", "glm", "lm")` — **the fitted `glm` itself**, with mariposa-specific slots attached:

coef_table Tibble with B, S.E., Wald, df, Sig., Exp(B), CI_lower, CI_upper

model_summary List with minus2LL, cox_snell_r2, nagelkerke_r2, mcfadden_r2

omnibus_test List with chi_sq, df, p for overall model test

classification List with table, overall_pct, pct_correct_0, pct_correct_1

hosmer_lemeshow List with chi_sq, df, p (goodness-of-fit test)

dv_encoding Tibble Original -> Internal (0/1) of the outcome categories (SPSS "Dependent Variable Encoding")

n Sample size (listwise complete cases; weighted N when weighted)

formula, dependent, predictor_names, weighted, weight_name, is_grouped, conf.level Call metadata.

Because the object inherits from `"glm"`, all standard generics (`predict()`, `anova()`, `vcov()`, `confint()`, `residuals()`, `fitted()`, `coef()`, `broom::tidy()`, `broom::glance()`, `broom::augment()`) dispatch natively without unwrapping. `summary()` returns the SPSS-style mariposa summary; for the raw glm summary use `stats::summary.glm()` on the same object. `confint()` returns Wald intervals on the log-odds scale (the SPSS method; `exp(confint(model))` gives the "C.I. for EXP(B)" of the summary), profile-likelihood intervals via `confint(model, method = "profile")`; see confint.logistic_regressior

For grouped data, returns a list of class `"logistic_regression"` with `$groups` holding one fitted glm-inheriting result per group.

See Also

linear_regression for continuous outcome variables.

chi_square for testing associations between categorical variables.

summary.logistic_regression for detailed output with toggleable sections.

Other regression: linear_regression(), marginal_effects()

Examples

```
library(dplyr)
data(survey_data)

# Create binary DV
survey_data$high_satisfaction <- ifelse(survey_data$life_satisfaction >= 4, 1, 0)

# Bivariate logistic regression
logistic_regression(survey_data, high_satisfaction ~ age)

# Multiple logistic regression
logistic_regression(survey_data, high_satisfaction ~ age + income + education)

# SPSS-style interface
logistic_regression(survey_data,
```

```
                    dependent = high_satisfaction,
                    predictors = c(age, income))

# Weighted logistic regression
logistic_regression(survey_data, high_satisfaction ~ age,
                    weights = sampling_weight)

# Grouped by region
survey_data |>
  dplyr::group_by(region) |>
  logistic_regression(high_satisfaction ~ age)

# Factor predictors: dummy-coding (default, matches base R glm())
logistic_regression(survey_data, high_satisfaction ~ age + education)

# Factor predictors: SPSS-style ordinal-as-scale
logistic_regression(survey_data, high_satisfaction ~ age + education,
                    factors = "numeric")

# --- Three-layer output ---
result <- logistic_regression(survey_data, high_satisfaction ~ age + income)
result                                  # compact one-line overview
summary(result)                         # full detailed SPSS-style output
summary(result, classification = FALSE)   # hide classification table
```

`logistic_regression-accessors`
Standard glm accessors for logistic_regression results

Description

`coef()`, `residuals()`, `fitted()` and `vcov()` dispatch to the `glm` methods for an ungrouped model; a grouped result raises an informative error (each element of `$groups` is a fitted model). `nobs()` returns the number of cases, or for a weighted model the unrounded sum of the frequency weights (SPSS N, shown rounded in the summary).

Usage

```
## S3 method for class 'logistic_regression'
coef(object, ...)

## S3 method for class 'logistic_regression'
residuals(object, ...)

## S3 method for class 'logistic_regression'
fitted(object, ...)

## S3 method for class 'logistic_regression'
```

```
vcov(object, ...)

## S3 method for class 'logistic_regression'
nobs(object, ...)
```

Arguments

object	A logistic_regression result.
...	Passed to the glm methods.

Value

As the corresponding glm method.

---

longitudinal_data *Longitudinal Study Data (Synthetic)*

Description

A synthetic repeated measures dataset suitable for testing longitudinal analysis functions. Contains realistic treatment effects, missing data patterns, and within-subject correlations.

Usage

```
longitudinal_data
```

Format

A data frame with 480 rows and 10 variables:

subject_id Subject identifier (factor, 1-120)

group Treatment group (Control, Treatment)

age Age in years (numeric)

gender Gender (Male, Female)

time Time point (T1, T2, T3, T4)

time_numeric Time point as numeric (1-4)

outcome_score Primary outcome measure (continuous)

secondary_outcome Secondary outcome measure (continuous)

physio_measure Physiological measure (continuous)

questionnaire_score Questionnaire rating (ordered factor, 1-7)

Details

Study design features:

- 120 subjects (60 per group)
- 4 measurement occasions
- Balanced treatment assignment
- Realistic dropout patterns (0% at T1, 35% by T4)
- Treatment effect grows over time (interaction effect)
- Autoregressive error structure
- Effect size approximately Cohen's d = 0.6

Missing data patterns reflect realistic longitudinal study attrition with higher dropout in later time points. The treatment effect demonstrates a clear Group x Time interaction suitable for repeated measures testing.

Source

Generated synthetically with realistic longitudinal correlation patterns and treatment effects. No real study data was used.

See Also

longitudinal_data_wide for the wide format version.

survey_data for the cross-sectional survey dataset.

Examples

```
# Load required packages and data
library(dplyr)
data(longitudinal_data)

# Repeated measurements are analysed in the wide format
# (see longitudinal_data_wide), e.g. with friedman_test()

# Descriptive statistics by group and time
longitudinal_data %>%
  group_by(group, time) %>%
  describe(outcome_score)
```

`longitudinal_data_wide` *Longitudinal Study Data - Wide Format (Synthetic)*

Description

Wide format version of the longitudinal_data dataset with one row per subject and separate columns for each time point. Useful for certain repeated measures analyses and data visualization.

Usage

```
longitudinal_data_wide
```

Format

A data frame with 120 rows and 8 variables:

subject_id Subject identifier (factor, 1-120)

group Treatment group (Control, Treatment)

age Age in years (numeric)

gender Gender (Male, Female)

score_T1 Outcome score at Time 1

score_T2 Outcome score at Time 2

score_T3 Outcome score at Time 3

score_T4 Outcome score at Time 4

Details

This is the wide format version of `longitudinal_data` with outcome scores spread across columns by time point. Missing values represent subject dropout or missed assessments.

Source

Generated synthetically from the long format longitudinal_data.

See Also

`longitudinal_data` for the long format version

Examples

```
# Load required packages and data
library(dplyr)
data(longitudinal_data_wide)

# Change over the four time points
friedman_test(longitudinal_data_wide, score_T1, score_T2, score_T3, score_T4)

# Change between two time points
wilcoxon_test(longitudinal_data_wide, score_T1, score_T2)
```

---

mann_whitney *Compare Two Groups Without Assuming Normal Data*

Description

`mann_whitney()` compares two groups when your data isn't normally distributed or when you have ordinal data (like ratings or rankings). It's the go-to alternative when t-tests aren't appropriate.

Think of it as:

- A robust way to compare groups that works with any data shape
- Perfect for Likert scales, ratings, or skewed distributions
- A test that compares the typical values between groups

The test tells you:

- Whether one group tends to have higher values than the other
- How strong the difference is (effect size)
- Which group has higher average ranks

Usage

```
mann_whitney(
  data,
  ...,
  group,
  weights = NULL,
  mu = 0,
  alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The variables you want to compare between groups. You can list multiple variables or use helpers like `starts_with("satisfaction")`
`group`	The categorical variable that defines your two groups (e.g., gender, treatment/control). Must have exactly two groups.
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`mu`	The hypothesized location shift of group 1 against group 2 (Default: 0, meaning no difference; this is the SPSS test). As in `wilcox.test()`, group-1 values are shifted by `mu` before ranking, so U, W, Z, r and p all refer to the same hypothesis. Only `mu = 0` is available with `weights`.
`alternative`	Direction of the test: • `"two.sided"` (default): Test if groups are different. Z is reported as SPSS does, from the smaller U (never positive). • `"greater"`: Test if group 1 > group 2 • `"less"`: Test if group 1 < group 2 For a one-sided test Z is directional (positive when group 1 tends to have higher values), so its sign matches the p-value.
`conf.level`	Not used. Rank tests report no confidence interval (SPSS `NPAR TESTS` neither); kept for backward compatibility.

Details

Understanding the Results:

P-value: If p < 0.05, the groups are significantly different

- p < 0.001: Very strong evidence of difference
- p < 0.01: Strong evidence of difference
- p < 0.05: Moderate evidence of difference
- p >= 0.05: No significant difference found

Effect Size r (How big is the difference?):

- |r| < 0.1: Negligible difference
- 0.1 to < 0.3: Small difference
- 0.3 to < 0.5: Medium difference
- 0.5 or higher: Large difference

Rank Mean Difference:

- Positive: Group 1 tends to have higher values
- Negative: Group 2 tends to have higher values
- Zero: Groups have similar distributions

When to Use This:

Use Mann-Whitney test when:

- Your data is not normally distributed (skewed, outliers)
- You have ordinal data (rankings, Likert scales)
- Sample sizes are small (< 30 per group)
- You want a robust alternative to the t-test
- You're comparing satisfaction ratings, income, or other skewed variables

Advantages Over t-test:

- Works with any data distribution
- Not affected by outliers
- Valid for ordinal data
- No assumptions about variance
- More robust for real-world data

Tips for Success:

- Each group should have at least 5 observations
- The test compares distributions, not just means
- Look at both p-values and effect sizes
- Consider plotting the data to see the pattern

Weighted analyses:

The weighted test is the design-based Lumley-Scott (2013) rank test (Horvitz-Thompson midranks with a sandwich variance) and agrees with `survey::svyranktest()`. Its U and W are design-based rank quantities; Z and the p-value are the test result. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- Use this for Likert scales and rating data

Value

Test results showing whether groups differ, including:

- U and W statistics (test statistics)
- Z-score and p-value (are groups different?)
- Effect size r (`r_effect`, how big is the difference?)
- Rank means for each group (which group is higher?) Use `summary()` for the full SPSS-style output with toggleable sections.

References

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics, 18(1), 50-60.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. Biometrics Bulletin, 1(6), 80-83.

Lumley, T., & Scott, A. (2013). Two-sample rank tests under complex sampling. Biometrika, 100(4), 831-842.

See Also

wilcox.test for the base R Wilcoxon test function.

svyranktest for survey-weighted rank tests.

t_test for parametric t-tests.

summary.mann_whitney for detailed output with toggleable sections.

Other hypothesis_tests: ancova(), binomial_test(), chi_square(), chisq_gof(), factorial_anova(), fisher_test(), friedman_test(), kruskal_wallis(), mcnemar_test(), oneway_anova(), t_test(), wilcoxon_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic Mann-Whitney test (non-parametric comparison)
survey_data %>%
  mann_whitney(age, group = gender)

# Multiple variables
survey_data %>%
  mann_whitney(age, income, life_satisfaction, group = region)

# Using tidyselect helpers
survey_data %>%
  mann_whitney(starts_with("trust_"), group = gender)

# Weighted analysis
survey_data %>%
  mann_whitney(income, group = region, weights = sampling_weight)

# Grouped analysis (separate tests for each education level)
survey_data %>%
  group_by(education) %>%
  mann_whitney(life_satisfaction, group = gender)

# One-sided test
survey_data %>%
  mann_whitney(life_satisfaction, group = region, alternative = "greater")

# --- Three-layer output ---
result <- survey_data %>%
  mann_whitney(income, group = gender, weights = sampling_weight)
result              # compact one-line overview
summary(result)     # full detailed output with all sections
summary(result, effect_sizes = FALSE)  # hide effect sizes
```

`marginal_effects` *Average Marginal Effects*

Description

`marginal_effects()` computes average marginal effects (AMEs) for a fitted `logistic_regression` model — the Stata `margins, dydx(*)` workhorse. AMEs translate logit coefficients into the language reports actually use: *percentage-point changes in the predicted probability*.

For each continuous predictor, the AME is the derivative of the predicted probability with respect to that predictor, averaged over the observed sample. For each factor level, it is the average *discrete change* in predicted probability compared to the reference level.

Usage

```
marginal_effects(model, conf.level = 0.95, ...)
```

Arguments

`model`	A fitted model object. Currently implemented for `logistic_regression` results (ungrouped or grouped).
`conf.level`	Confidence level for the AME intervals (default 0.95).
`...`	Additional arguments (not used).

Details

Understanding the Output:

An AME of 0.03 for `age` means: averaged over the sample, one additional year of age increases the predicted probability of the outcome by 3 percentage points. Unlike odds ratios, AMEs are directly comparable across models and across groups, which is why they are the preferred reporting style in much of the social sciences.

When to Use This:

- To report logistic regression results on the probability scale instead of (or alongside) odds ratios
- To compare predictor effects across groups (run on a `group_by()`-fitted model)

Technical Details:

Continuous predictors use a centered numerical derivative of the predicted probability; factor, character and logical predictors use the average discrete change against the reference level (the first level; `FALSE` for logicals). Effects are computed per *variable* from the original data: a variable that enters through several terms (`x + I(x^2)`, `poly(x, 2)`, `log(x)`, interactions) gets one AME that moves all of those terms together. A numeric variable that enters only as a categorical term (e.g. `factor(x)`) cannot be differentiated; it is skipped with a warning. Standard errors come from the delta method with the analytic gradient of the AME with respect to the coefficients and the model's Wald covariance matrix, matching the default in Stata's `margins` and R's **margins**

package. Weighted models average with their frequency weights (unrounded); with all weights equal to 1 the weighted AME equals the unweighted one.

Linear regression: AMEs are not needed there — for a linear model without interactions or transformations, the unstandardized coefficient B *is* the marginal effect, already shown in the `linear_regression` coefficients table. Calling `marginal_effects()` on a `linear_regression` result says so instead of duplicating the table.

Validation: SPSS has no AME procedure. The results are verified against the analytic logit-AME formula and an independent finite-difference delta-method computation.

Value

An object of class `"marginal_effects"` whose `$results` tibble holds one row per predictor term (and group combination) with:

Term Predictor name; factor rows read `"var: level vs. ref"`

Type `"dydx"` (continuous derivative) or `"discrete"` (factor-level contrast)

AME Average marginal effect on the probability scale

SE Delta-method standard error

z, p_value Wald test of the AME

CI_lower, CI_upper Confidence interval at `conf.level`

See Also

`logistic_regression` for fitting the model.

`summary.marginal_effects` for detailed output.

Other regression: `linear_regression()`, `logistic_regression()`

Examples

```
data(survey_data)
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction >= 4)

model <- logistic_regression(survey_data,
                             high_satisfaction ~ age + income + education)
marginal_effects(model)

# Weighted model
model_w <- logistic_regression(survey_data, high_satisfaction ~ age + income,
                               weights = sampling_weight)
marginal_effects(model_w)

# --- Three-layer output ---
ame <- marginal_effects(model)
ame                 # compact overview
summary(ame)        # full detailed output
```

mcnemar_test *McNemar's Test for Paired Proportions*

Description

mcnemar_test() tests whether paired proportions have changed between two dichotomous measurements. Use this for before/after comparisons of categorical outcomes.

Think of it as:

- A paired comparison test for binary (yes/no) data
- The categorical equivalent of a paired t-test
- Tests whether the proportion of "changers" is symmetric

The test tells you:

- Whether paired proportions changed significantly
- Both asymptotic and exact p-values for maximum reliability
- The number of discordant pairs (who actually changed)

Usage

```
mcnemar_test(data, var1, var2, weights = NULL, correct = TRUE, ...)
```

Arguments

data	Your survey data (data frame or tibble)
var1	First dichotomous variable (0/1 or two-level factor)
var2	Second dichotomous variable (0/1 or two-level factor)
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
correct	Logical, whether to apply continuity correction (default: TRUE)
...	Additional arguments (currently unused)

Details

Understanding the Results:

P-value: If $p < 0.05$, the paired proportions are significantly different

- $p < 0.001$: Very strong evidence of change
- $p < 0.01$: Strong evidence of change
- $p < 0.05$: Moderate evidence of change
- $p >= 0.05$: No significant change found

Exact vs Asymptotic: The exact binomial p-value is more reliable for small samples. For large samples, both p-values will be very similar.
Discordant Pairs: Only pairs where the two measurements differ (b and c) contribute to the test. If b approximately equals c, there is no evidence of systematic change.

When to Use This:
Use McNemar's test when:

- You have paired or matched binary data
- You're comparing before/after proportions
- Both variables must be dichotomous (exactly 2 levels)

The McNemar Statistic:
For a 2x2 table with discordant cells b and c:

$$\chi^2 = \frac{(|b - c| - 1)^2}{b + c}$$

(with continuity correction)
The exact test uses a binomial test on the discordant pairs.

Relationship to Other Tests:

- For unpaired categorical data: Use `chi_square()` instead
- For paired ordinal/continuous data: Use `wilcoxon_test()` instead
- For paired data with more than 2 levels: Consider the Bowker test of symmetry

SPSS Equivalent:
SPSS: `CROSSTABS /STATISTICS=MCNEMAR`

Value

Test results showing whether paired proportions changed, including:

- McNemar chi-square statistic (`chi_squared`, with continuity correction)
- Asymptotic p-value
- Exact binomial p-value (two-sided)
- 2x2 contingency table
- Discordant pair counts (b and c)

References

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157.

See Also

`chi_square` for independence tests.

`wilcoxon_test` for paired non-parametric tests on ordinal data.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `oneway_anova()`, `t_test()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Create dichotomous variables
test_data <- survey_data %>%
  mutate(
    trust_gov_high = as.integer(trust_government >= 4),
    trust_media_high = as.integer(trust_media >= 4)
  )

# McNemar test
test_data %>%
  mcnemar_test(var1 = trust_gov_high, var2 = trust_media_high)

# Grouped analysis
test_data %>%
  group_by(region) %>%
  mcnemar_test(var1 = trust_gov_high, var2 = trust_media_high)
```

---

multiple_response *Analyze Multiple Response Sets*

Description

`multiple_response()` analyzes "check all that apply" survey questions — the SPSS `MULT RESPONSE` procedure. It takes a set of 0/1 indicator variables (one per answer option) and produces the two tables SPSS users know:

- Without `by`: the **frequencies table** — number of mentions per option, *percent of responses* (sums to 100%), and *percent of cases* (sums above 100%, because respondents can tick several options).
- With `by`: the **crosstab** of the set against a categorical variable — mentions and case-based column percentages per group.

Usage

```
multiple_response(data, ..., by = NULL, counted = 1, weights = NULL)
```

Arguments

`data`	Your survey data (a data frame or tibble). If grouped (via `dplyr::group_by()`), separate tables are produced per group (SPSS SPLIT FILE).
`...`	The indicator variables of the set (unquoted, supports tidyselect, e.g. `starts_with("info_")`). Each is checked against `counted`.

by	Optional categorical variable (unquoted) to cross the set against (SPSS MULT RESPONSE ... BY factor).
counted	The value that counts as a mention (default 1, matching SPSS dichotomy sets with counted value 1). For factor or character indicators give the level that marks a mention, e.g. counted = "yes"; a value that does not occur among their levels is an error.
weights	Optional survey weights, treated as frequency weights matching SPSS WEIGHT BY. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.

Details

Understanding the Output:

The two percentage columns answer different questions:

- **Percent of responses**: "Of all boxes ticked, how many were this option?" — describes the mix of answers.
- **Percent of cases**: "What share of respondents ticked this option?" — usually the number reports need. It sums above 100% whenever respondents tick more than one box.

Case handling:

Following SPSS MULT RESPONSE, a case is **valid** if it has at least one non-missing indicator in the set; cases missing on *all* indicators are excluded and reported as missing. Within a valid case, missing indicators simply contribute no mention. With by, cases missing on the by variable are excluded too.

Technical Details:

Weighted analyses count mentions and cases as unrounded sums of weights; displayed Ns are rounded. With weights == 1 the weighted table reduces exactly to the unweighted one. Only dichotomy sets (indicator + counted value) are supported; SPSS's category-range sets are not.

The counts and percentages are checked by direct computation from the indicator matrix and are not compared with SPSS output (see the SPSS compatibility vignette).

Value

An object of class "multiple_response" whose $results tibble holds one row per answer option (and group combination) with:

Option Variable name of the option

Label The option's variable label (falls back to the name)

n Number of mentions (weighted sum when weighted)

pct_responses Share of all mentions — sums to 100%

pct_cases Share of valid cases mentioning the option — can sum above 100%

With by, $by_results additionally holds the long-form crosstab (columns by_level, Option, Label, n, pct_cases). $n_cases is the number of valid cases (at least one non-missing indicator), $n_responses the total number of mentions.

See Also

frequency for single-variable frequency tables.

crosstab for ordinary two-variable crosstabs.

summary.multiple_response for detailed output.

Other descriptive: crosstab(), describe(), frequency(), normality_test()

Examples

```
library(dplyr)
data(survey_data)

# Build an example set: which institutions does a respondent
# trust highly (rating of 4 or 5)?
trust <- survey_data %>%
  mutate(
    gov     = as.integer(trust_government >= 4),
    media   = as.integer(trust_media >= 4),
    science = as.integer(trust_science >= 4)
  )

# Frequencies table: % of responses vs. % of cases
multiple_response(trust, gov, media, science)

# Crossed against gender, with weights
multiple_response(trust, gov, media, science,
                  by = gender, weights = sampling_weight)

# --- Three-layer output ---
result <- multiple_response(trust, gov, media, science)
result              # full tables
summary(result)     # same tables, with section toggles
```

---

na_frequencies *Frequency Table of Missing Value Types*

Description

Shows a breakdown of the different types of missing values in a variable that was read with read_spss(), read_stata(), read_sas(), or read_xpt() and contains tagged NAs.

Usage

```
na_frequencies(x, ...)
```

Arguments

x	A numeric vector with tagged NAs, or a data frame.
...	For a data frame: the variables to tabulate (tidyselect). If empty, every numeric variable with missing values is tabulated.

Value

A data frame with one row per missing type, ordered by code like the missing block of `frequency()` (system missing last, listed only when it occurs):

code	The original missing value code: numeric for SPSS codes (e.g., -9, -8), character for native format codes (e.g., ".a" for Stata, ".A" for SAS)
label	The value label for this missing type (if available)
n	Number of cases with this missing type
prc	Percent of all cases
tag	The internal tag character of the tagged NA

For a data frame, a first column `variable` names the variable. Without any missing values the (empty) result is returned invisibly with a message.

See Also

`read_spss()`, `read_stata()`, `read_sas()`, `read_xpt()`, `untag_na()`, `strip_tags()`

Other data-import: `read_por()`, `read_sas()`, `read_spss()`, `read_stata()`, `read_xlsx()`, `read_xpt()`, `strip_tags()`, `untag_na()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Declare -9/-8 as distinct tagged missing types, then inspect them
  x <- set_na(c(1, 2, -9, 3, -8, -9), -9, -8)
  na_frequencies(x)
  #   code label n      prc tag
  # 1   -9  <NA> 2 33.33333   a
  # 2   -8  <NA> 1 16.66667   b
}
```

---

nobs.linear_regression

Number of observations of a linear_regression model

Description

Unweighted: the number of complete cases. Weighted: the unrounded sum of the frequency weights of the complete cases - SPSS's N (the summary shows it rounded).

Usage

```
## S3 method for class 'linear_regression'
nobs(object, ...)
```

Arguments

object	A linear_regression result (ungrouped, listwise).
...	Not used.

Value

A single number.

---

normality_test *Test Variables for Normality*

Description

normality_test() checks whether numeric variables follow a normal distribution, producing the two tests SPSS prints in its EXAMINE "Tests of Normality" table:

- **Kolmogorov-Smirnov** with Lilliefors significance correction (the "Kolmogorov-Smirnov(a)" column in SPSS)
- **Shapiro-Wilk** (computed for n between 3 and 5000, matching the SPSS convention)

Use it before a t-test, ANOVA, or Pearson correlation to check the normality assumption, together with levene_test for the equal-variances assumption.

Usage

```
normality_test(data, ...)
```

Arguments

data	Your survey data (a data frame or tibble). If grouped (via dplyr::group_by()), the tests are run separately per group — matching SPSS EXAMINE ... BY factor.
...	Variables to test (unquoted, supports tidyselect). All selected variables must be numeric.

Details

Understanding the Output:

For both tests the null hypothesis is "the variable is normally distributed":

- **p >= 0.05**: No significant deviation from normality detected.
- **p < 0.05**: The variable deviates significantly from a normal distribution.

With large survey samples these tests flag even tiny, practically irrelevant deviations. Combine them with the skewness and kurtosis from `describe` before deciding against a parametric test.

When to Use This:

- Before `t_test`, `oneway_anova`, or `pearson_cor` to check the normality assumption
- Grouped (via `group_by()`) to check normality *within* each comparison group — the form of the assumption that actually matters for group comparisons

If normality is clearly violated, consider the rank-based alternatives: `mann_whitney`, `kruskal_wallis`, or `spearman_rho`.

Technical Details:

The Lilliefors-corrected p-value uses the Dallal-Wilkinson (1986) approximation, the same correction SPSS applies in EXAMINE ("Lilliefors Significance Correction"). Shapiro-Wilk uses `stats::shapiro.test()` and is reported for 3 <= n <= 5000, as in SPSS. Cases with missing values are excluded per variable.

Weights: `normality_test()` deliberately takes no `weights` argument. Normality checks are a diagnostic of the *sample* distribution, and neither Shapiro-Wilk nor the Lilliefors correction has a well-defined fractional-frequency-weight form. Run the test on the unweighted sample.

The statistics are checked against independent R implementations of the same published formulas and are not compared with SPSS output (see the SPSS compatibility vignette).

Value

An object of class `"normality_test"` whose `$results` tibble holds one row per variable (and group combination) with:

n Number of valid (non-missing) cases

ks_statistic, ks_df, ks_p Kolmogorov-Smirnov statistic with Lilliefors-corrected p-value (Dallal-Wilkinson approximation); df equals n as in SPSS

shapiro_w, shapiro_p Shapiro-Wilk W and p-value (`NA` when n < 3 or n > 5000)

References

Dallal, G. E., & Wilkinson, L. (1986). An analytic approximation to the distribution of Lilliefors's test statistic for normality. *The American Statistician*, 40(4), 294-296.

See Also

`levene_test` for the equal-variances assumption.

`describe` for skewness and kurtosis.

`summary.normality_test` for detailed output.

Other descriptive: `crosstab()`, `describe()`, `frequency()`, `multiple_response()`

Examples

```
library(dplyr)
data(survey_data)

# Test a single variable
normality_test(survey_data, age)

# Several variables at once
normality_test(survey_data, age, income, life_satisfaction)

# Within comparison groups (SPSS: EXAMINE ... BY gender)
survey_data %>%
  group_by(gender) %>%
  normality_test(age, income)

# --- Three-layer output ---
result <- normality_test(survey_data, age, income)
result              # compact overview
summary(result)     # full SPSS-style table
```

---

oneway_anova *Compare Multiple Groups: Are Their Averages Different?*

Description

`oneway_anova()` helps you determine if average values differ across three or more groups. For example, does average income vary by education level? Or is customer satisfaction different across regions?

Think of it as:

- An extension of t-test for more than two groups
- A way to test if group membership affects outcomes
- A tool to identify meaningful group differences

The test tells you:

- Whether at least one group is different from the others
- How much variation is explained by group membership
- Which variance assumption fits your data best

Usage

```
oneway_anova(
  data,
  ...,
  group,
```

```
  weights = NULL,
  var.equal = TRUE,
  conf.level = 0.95
)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The numeric variables you want to analyze. You can list multiple variables or use helpers like starts_with("income")
group	The categorical variable that defines your groups (e.g., education, region, age_group). Must have at least 3 groups for ANOVA.
weights	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
var.equal	Deprecated and ignored (a warning is issued when set to FALSE). Like SPSS ONEWAY, the classical ANOVA table and Welch's robust test are always both computed and displayed.
conf.level	Confidence level for intervals (Default: 0.95 = 95%)

Details

Understanding the Results:

P-value: If p < 0.05, at least one group average is different

- p < 0.001: Very strong evidence of group differences
- p < 0.01: Strong evidence of group differences
- p < 0.05: Moderate evidence of group differences
- p >= 0.05: No significant group differences found

Effect Sizes (How much do groups matter?):

- **Eta-squared**: Proportion of variance explained by groups
  - < 0.01: Negligible effect
  - 0.01-0.06: Small effect
  - 0.06-0.14: Medium effect
  - 0.14 or higher: Large effect
- **Omega-squared**: More conservative estimate (usually preferred)
- Omega-squared and epsilon-squared are truncated at 0; a negative raw estimate (which occurs when F < 1) is reported as 0.

When to Use This:

Use ANOVA when:

- You have one numeric outcome (income, satisfaction score, etc.)
- You have one categorical grouping variable with 3+ groups
- You want to know if group averages differ
- Your data is roughly normally distributed within groups

Variance Assumptions:

Like SPSS ONEWAY, both results are always shown for comparison:

- **Standard ANOVA**: valid when group variances are similar
- **Welch's ANOVA**: robust when group variances differ
- Use `levene_test` to check which situation applies

What Comes Next?:

If ANOVA is significant:

1. Look at group means to see the pattern
2. Use `tukey_test()` to find which specific groups differ
3. Consider effect sizes to judge practical importance

Tips for Success:

- Check that each group has sufficient observations (ideally 20+)
- Look at both p-values and effect sizes
- Use Welch's ANOVA if group sizes are very unequal
- Follow up with post-hoc tests to identify specific differences
- Welch's robust test for equality of means

Value

ANOVA results showing whether groups differ, including:

- F-statistic (`F_statistic`) and p-value (are groups different?)
- Effect sizes (how much do groups matter?)
- Group statistics (means and standard deviations)
- Both standard and Welch ANOVA results Use `summary()` for the full SPSS-style output with toggleable sections.

References

Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences (2nd ed.). Lawrence Erlbaum Associates.

Welch, B. L. (1951). On the comparison of several mean values: an alternative approach. Biometrika, 38(3/4), 330-336.

Olejnik, S., & Algina, J. (2003). Generalized eta and omega squared statistics: measures of effect size for some common research designs. Psychological Methods, 8(4), 434-447.

See Also

`aov` for the base R ANOVA function.

`oneway.test` for Welch's ANOVA.

`tukey_test` for post-hoc pairwise comparisons.

`levene_test` for testing homogeneity of variances.

`summary.oneway_anova` for detailed output with toggleable sections.

Other hypothesis_tests: ancova(), binomial_test(), chi_square(), chisq_gof(), factorial_anova(), fisher_test(), friedman_test(), kruskal_wallis(), mann_whitney(), mcnemar_test(), t_test(), wilcoxon_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic one-way ANOVA (comparing across education levels)
survey_data %>%
  oneway_anova(life_satisfaction, group = education)

# Multiple dependent variables
survey_data %>%
  oneway_anova(life_satisfaction, trust_government, group = education)

# Using tidyselect helpers
survey_data %>%
  oneway_anova(starts_with("trust_"), group = education)

# Weighted analysis
survey_data %>%
  oneway_anova(income, group = education, weights = sampling_weight)

# Grouped analysis (separate ANOVA for each region)
survey_data %>%
  group_by(region) %>%
  oneway_anova(life_satisfaction, group = education)

# Store results for post-hoc analysis
result <- survey_data %>%
  oneway_anova(life_satisfaction, group = education)

# Follow up with post-hoc tests
result %>% tukey_test()
result %>% levene_test()  # Check homogeneity of variances

# --- Three-layer output ---
result              # compact one-line overview
summary(result)     # full detailed output with all sections
summary(result, descriptives = FALSE)  # hide group statistics
```

`pairwise_wilcoxon` *Find Which Specific Measurements Differ After Friedman Test*

Description

`pairwise_wilcoxon()` tells you exactly which pairs of measurements differ from each other after the Friedman test finds overall differences. It performs all pairwise Wilcoxon signed-rank tests with p-value correction.

Think of it as:

- Friedman says "there are differences somewhere among the measurements"
- Pairwise Wilcoxon says "specifically, Measurement A differs from Measurement C"
- A way to make all possible pairwise comparisons for repeated measures

Usage

```
pairwise_wilcoxon(x, ...)

## Default S3 method:
pairwise_wilcoxon(x, ...)
```

Arguments

`x`	Friedman test results from `friedman_test()`
`...`	Additional arguments passed to methods. The method for `friedman_test` objects accepts `p_adjust` (character): method for adjusting p-values for multiple comparisons. Options: `"bonferroni"` (default, most conservative), `"holm"`, `"BH"`, `"hochberg"`, `"hommel"`, `"BY"`, `"fdr"`, `"none"`.

Details

Understanding the Results:

Z-Statistics: Based on the Wilcoxon signed-rank test for each pair

- Large absolute Z values indicate big differences between two measurements
- Z is computed from the differences `var2 - var1`, like `wilcoxon_test(x = var1, y = var2)` and the SPSS pair "var2 - var1"
- As in SPSS, Z comes from the smaller rank sum and is never positive; `z_based_on` names that sum (SPSS's footnote)
- Based on negative ranks: values in var2 (Var 2) tend to be higher
- Based on positive ranks: values in var1 (Var 1) tend to be higher

Adjusted P-values: Control for multiple comparisons

- $p < 0.05$: Measurements are significantly different
- $p >= 0.05$: No significant difference between these measurements

The Wilcoxon Signed-Rank Test:

For each pair of measurements, the Wilcoxon signed-rank test:

1. Computes differences between the two measurements
2. Ranks the absolute differences
3. Computes a Z-statistic based on the rank sums

4. Uses normal approximation with tie correction

Which Cases Are Used:

Each pair uses all cases with valid values on both of its variables (pairwise deletion), exactly like the individual SPSS `NPAR TESTS /WILCOXON` tests these results are validated against. `friedman_test()` uses only cases complete on all variables (listwise, as SPSS `/FRIEDMAN`), so a pair's `n` can be larger than the Friedman N.

P-Value Adjustment Methods:

- **Bonferroni** (default): Most conservative, multiplies p by number of comparisons
- **Holm**: Step-down method, less conservative than Bonferroni
- **BH**: Controls false discovery rate, good for many comparisons

When to Use This:

Use pairwise Wilcoxon when:

- Your Friedman test shows significant differences ($p < 0.05$)
- You want to know which specific measurements differ
- Your data are ordinal or violate normality assumptions
- You have repeated measures or matched groups

Relationship to Other Tests:

- Non-parametric post-hoc for repeated measures (like Dunn is for independent groups)
- Follow-up to `friedman_test()`, like `dunn_test()` follows `kruskal_wallis()`

Weighted analyses:

When the `friedman_test()` result is weighted, each pairwise test uses the same frequency-weighted signed-rank formulas. With all weights equal to 1 they reproduce the unweighted tests exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- Each pairwise comparison uses `wilcoxon_test()` logic internally

Value

Pairwise comparison results showing:

- Which measurement pairs are significantly different
- Z-statistics from Wilcoxon signed-rank tests
- Adjusted p-values (controlling for multiple comparisons)
- The number of cases of each pair (`n`)

The comparison table (one row per pair) is stored as `$results` (also available as `$comparisons`).

References

Wilcoxon, F. (1945). Individual comparisons by ranking methods. Biometrics Bulletin, 1(6), 80-83.

See Also

friedman_test for performing Friedman tests.

wilcoxon_test for individual paired Wilcoxon tests.

dunn_test for post-hoc comparisons after Kruskal-Wallis.

Other posthoc: dunn_test(), levene_test(), scheffe_test(), tukey_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Perform Friedman followed by pairwise Wilcoxon post-hoc
friedman_result <- survey_data %>%
  friedman_test(trust_government, trust_media, trust_science)

# Pairwise Wilcoxon comparisons (default: Bonferroni)
friedman_result %>% pairwise_wilcoxon()

# With Holm correction (less conservative)
friedman_result %>% pairwise_wilcoxon(p_adjust = "holm")

# With Benjamini-Hochberg (controls false discovery rate)
friedman_result %>% pairwise_wilcoxon(p_adjust = "BH")

# With weights
fw_weighted <- survey_data %>%
  friedman_test(trust_government, trust_media, trust_science,
                weights = sampling_weight)

fw_weighted %>% pairwise_wilcoxon()

# Grouped analysis
fw_grouped <- survey_data %>%
  group_by(region) %>%
  friedman_test(trust_government, trust_media, trust_science)

fw_grouped %>% pairwise_wilcoxon()
```

---

partial_cor *Partial Correlation*

Description

partial_cor() computes partial correlations between variables while controlling for one or more other variables — the SPSS PARTIAL CORR procedure (Stata: pcorr). It answers the question "how strongly are x and y related once the influence of the control variables is removed?"

Usage

```
partial_cor(data, ..., controls, weights = NULL)
```

Arguments

data	Your survey data (a data frame or tibble). If grouped (via dplyr::group_by()), separate partial correlations are computed per group (SPSS SPLIT FILE).
...	Variables to correlate (unquoted, supports tidyselect). At least two.
controls	Control variable(s) to partial out (unquoted, supports tidyselect; SPSS BY list). At least one, must not overlap with the analysis variables.
weights	Optional survey weights, treated as frequency weights matching SPSS WEIGHT BY. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.

Details

Understanding the Output:

Comparing partial_r against zero_order_r tells you what the controls contribute:

- **Partial clearly smaller than zero-order**: much of the original association runs through the control variables (confounding or mediation).
- **Partial similar to zero-order**: the association is largely independent of the controls.
- **Partial larger than zero-order**: a suppressor situation — the controls masked part of the association.

When to Use This:

- Check whether a bivariate correlation survives controlling for demographics (age, education, ...)
- Separate the direct association of two attitudes from what a shared cause explains

For full multivariate control with several predictors, use linear_regression instead.

Technical Details:

Cases are deleted **listwise** across the analysis and control variables (SPSS /MISSING=LISTWISE, the PARTIAL CORR default). The partial correlation is computed from the Pearson correlation matrix of all variables via the inverse of the control-variable block; the same Pearson formula as pearson_cor is used throughout, so weighted results follow the SPSS frequency-weight convention with unrounded sum(w) in df and test statistics. Significance is two-tailed (the SPSS default).

The statistics are checked against the equivalent computation from regression residuals and are not compared with SPSS output (see the SPSS compatibility vignette).

Value

An object of class "partial_cor" whose $correlations tibble holds one row per variable pair (and group combination) with:

partial_r Partial correlation controlling for the controls

zero_order_r The ordinary (zero-order) Pearson correlation of the pair, for comparison — SPSS /STATISTICS=CORR

df Degrees of freedom, n - 2 - k with k control variables (non-integer for weighted data)

t_stat, p_value t-test of the partial correlation (two-tailed, the SPSS default)

n Listwise-complete sample size (rounded weighted N when weighted)

For three or more analysis variables, `$matrices` additionally holds the full partial-correlation matrix per group.

See Also

`pearson_cor` for zero-order correlations.

`linear_regression` for multivariate control.

`summary.partial_cor` for detailed output.

Other correlation: `kendall_tau()`, `pearson_cor()`, `spearman_rho()`

Examples

```
library(dplyr)
data(survey_data)

# Does the satisfaction-income correlation survive controlling for age?
partial_cor(survey_data, life_satisfaction, income, controls = age)

# Several variables, several controls
partial_cor(survey_data, trust_government, trust_media, trust_science,
            controls = c(age, political_orientation))

# Weighted (SPSS WEIGHT BY)
partial_cor(survey_data, life_satisfaction, income,
            controls = age, weights = sampling_weight)

# Grouped (SPSS SPLIT FILE)
survey_data %>%
  group_by(gender) %>%
  partial_cor(life_satisfaction, income, controls = age)

# --- Three-layer output ---
result <- partial_cor(survey_data, life_satisfaction, income, controls = age)
result              # compact overview
summary(result)     # full detailed output
```

`pearson_cor` *Measure How Strongly Variables Are Related*

Description

`pearson_cor()` shows you how strongly numeric variables are related to each other. For example, is age related to income? Does satisfaction increase with experience? This helps you understand patterns in your data.

The correlation tells you:

- **Direction**: Positive (both increase together) or negative (one increases as other decreases)
- **Strength**: How closely the variables move together (from 0 = no relationship to 1 = perfect relationship)
- **Significance**: Whether the relationship is real or could be due to chance

Usage

```
pearson_cor(
  data,
  ...,
  weights = NULL,
  conf.level = 0.95,
  alternative = c("two.sided", "less", "greater"),
  use = c("pairwise", "listwise"),
  na.rm = NULL
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to correlate. List two for a single correlation or more for a correlation matrix.
`weights`	Optional survey weights for population-representative results. Following SPSS CORRELATIONS, the weighted degrees of freedom use `n = sum(w)` (not Kish's effective sample size). This is appropriate for normalized survey weights with mean ≈ 1. For raw expansion weights (e.g., summing to millions of population units), the resulting standard errors and confidence intervals will be drastically too narrow — in that case, normalize weights so that `sum(w) == n`, or use the survey package for design-based inference. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`conf.level`	Confidence level for intervals (Default: 0.95 = 95%)
`alternative`	Direction of the test: `"two.sided"` (default), `"less"`, or `"greater"`. A one-sided test gets the matching one-sided confidence interval (`[-1, upper]` for `"less"`, `[lower, 1]` for `"greater"`), as `stats::cor.test()`.
`use`	How to handle missing values: • `"pairwise"` (default): Use all available data for each pair • `"listwise"`: Only use complete cases across all variables
`na.rm`	Deprecated. Use `use` instead.

Details

Understanding the Results:

Correlation coefficient (r) ranges from -1 to +1:

- **+1**: Perfect positive relationship (as one goes up, the other always goes up)
- **0**: No linear relationship
- **-1**: Perfect negative relationship (as one goes up, the other always goes down)

Interpreting strength (absolute value of r):

- 0.00 - 0.10: Negligible relationship
- 0.10 - 0.30: Weak relationship
- 0.30 - 0.50: Moderate relationship
- 0.50 - 0.70: Strong relationship
- 0.70 - 0.90: Very strong relationship
- 0.90 - 1.00: Extremely strong relationship

P-value interpretation:

- $p < 0.001$: Very strong evidence of a relationship
- $p < 0.01$: Strong evidence of a relationship
- $p < 0.05$: Moderate evidence of a relationship
- $p >= 0.05$: No significant relationship found

A correlation of 0.65 with $p < 0.001$ means:

- Strong positive relationship (r = 0.65)
- As one variable increases, the other tends to increase
- Very unlikely to be due to chance ($p < 0.001$)
- About 42% of variation is shared (r-squared = 0.65 squared = 0.42)

When to Use This:

Use Pearson correlation when:

- Both variables are numeric and continuous
- You expect a linear relationship
- Data is roughly normally distributed
- You want to measure strength of linear association

Don't use when:

- Data has extreme outliers (consider Spearman instead)
- Relationship is curved/non-linear
- Variables are categorical (use chi-squared test)
- You need to establish causation (correlation does not imply causation)

Tips for Success:

- Always plot your data first to check for non-linear patterns
- Consider both statistical significance (p-value) and practical importance (r value)
- Remember: correlation does not imply causation
- Check for outliers that might inflate or deflate correlations
- Use Spearman correlation for ordinal data or non-normal distributions

Value

Correlation results showing relationships between variables, including:

- Correlation coefficient (r): Strength and direction of relationship
- P-value: Whether the relationship is statistically significant
- Confidence interval: Range of plausible correlation values
- Sample size: Number of observations used Use `summary()` for the full SPSS-style output with toggleable sections.

References

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Lawrence Erlbaum Associates.

Fisher, R. A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 10(4), 507–521.

See Also

`cor` for the base R correlation function.

`cor.test` for correlation significance testing.

`spearman_rho` for rank-based correlation (robust to outliers).

`kendall_tau` for ordinal correlation.

`summary.pearson_cor` for detailed output with toggleable sections.

Other correlation: `kendall_tau()`, `partial_cor()`, `spearman_rho()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic correlation between two variables
survey_data %>%
  pearson_cor(age, income)

# Correlation matrix for multiple variables
survey_data %>%
  pearson_cor(age, income, life_satisfaction)

# Weighted correlations
survey_data %>%
  pearson_cor(age, income, weights = sampling_weight)

# Grouped correlations
survey_data %>%
  group_by(region) %>%
  pearson_cor(age, income, life_satisfaction)
```

```
# Using tidyselect helpers
survey_data %>%
  pearson_cor(where(is.numeric), weights = sampling_weight)

# Listwise deletion for missing data
survey_data %>%
  pearson_cor(age, income, use = "listwise")

# --- Three-layer output ---
result <- survey_data %>%
  pearson_cor(age, income, life_satisfaction, weights = sampling_weight)
result              # compact one-line overview
summary(result)     # full correlation, p-value, and N matrices
summary(result, pvalue_matrix = FALSE)  # hide p-values
```

---

phi *Effect Sizes for Contingency Tables*

Description

Convenience helpers that run chi_square and return just the requested effect size as a numeric value (named by group for grouped data): phi() (sqrt(chi-squared / N); in a 2x2 table signed like SPSS's Phi, negative when the first row goes with the second column; bounded by 1 only in 2x2 tables), cramers_v() (normalized to 0-1 for any table size), and goodman_gamma() for two ordinal variables. As in SPSS, Phi and Cramer's V are computed for any table size.

For the full test output (chi-square statistic, p-value, all effect sizes), call chi_square directly.

Usage

```
phi(data, ..., weights = NULL)

cramers_v(data, ..., weights = NULL)

goodman_gamma(data, ..., weights = NULL)
```

Arguments

data	Your survey data (a data frame or tibble)
...	Exactly two categorical variables, as in chi_square()
weights	Optional survey weights. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.

Value

A numeric vector with the effect size (one element per group for grouped data).

See Also

chi_square

Examples

```
data(survey_data)
phi(survey_data, gender, region)
cramers_v(survey_data, education, region)
goodman_gamma(survey_data, education, life_satisfaction)
```

---

pomps *Transform Scores to Percent of Maximum Possible (POMPS)*

Description

`pomps()` transforms scores to a 0-100 scale using the Percent of Maximum Possible Scores method. This makes different scales directly comparable regardless of their original range.

This is the R equivalent of the SPSS formula: `COMPUTE v81p = ((v81 - 1) / (7 - 1)) * 100.`

A score of 0 means the minimum possible score, 100 means the maximum. The transformation preserves all correlations between variables.

Usage

```
pomps(x, scale_min = NULL, scale_max = NULL)
```

Arguments

`x`	A numeric vector to transform (e.g., a column from your data).
`scale_min`	The theoretical minimum of the scale. If `NULL` (default), the observed minimum of `x` is used.
`scale_max`	The theoretical maximum of the scale. If `NULL` (default), the observed maximum of `x` is used.

Details

The Formula:

POMPS = ((score - scale_min) / (scale_max - scale_min)) * 100

Why Specify scale_min and scale_max?:

By default, `pomps()` uses the observed minimum and maximum of your data. However, for Likert scales you should specify the *theoretical* range:

- A 1-5 Likert scale: `scale_min = 1, scale_max = 5`
- A 1-7 Likert scale: `scale_min = 1, scale_max = 7`
- A 0-10 scale: `scale_min = 0, scale_max = 10`

Using theoretical values ensures that the transformation is consistent across samples and time points. Values outside the range (e.g. an unrecoded "don't know" = 9 on a 1-5 scale) give scores below 0 or above 100; `pomps()` warns about them, so recode missing codes first.

When to Use This:

- Comparing variables measured on different scales
- Creating profile plots across scales with different ranges
- Reporting scale scores in an intuitive 0-100 format

Value

A numeric vector of the same length as `x`, with values rescaled to the 0-100 range. The variable label of `x` is kept; missing values (including the SPSS missing types of imported data) become plain `NA`, as in an SPSS `COMPUTE`.

See Also

`row_means` for creating mean indices across items.

Other scale: `efa()`, `reliability()`, `row_count()`, `row_means()`, `row_sums()`

Examples

```
library(dplyr)
data(survey_data)

# Transform a 1-5 Likert scale to POMPS
survey_data <- survey_data %>%
  mutate(trust_gov_pomps = pomps(trust_government, scale_min = 1, scale_max = 5))

# Transform multiple variables with the same scale
survey_data <- survey_data %>%
  mutate(across(
    c(trust_government, trust_media, trust_science),
    ~ pomps(.x, scale_min = 1, scale_max = 5),
    .names = "{.col}_pomps"
  ))

# Auto-detect range (uses observed min/max)
survey_data <- survey_data %>%
  mutate(age_pomps = pomps(age))
```

`predict.linear_regression` *Predict from a linear_regression model*

Description

For listwise + ungrouped results, dispatches to `stats::predict.lm` (the result inherits from `"lm"`). For grouped or pairwise results, raises an informative error.

Usage

```
## S3 method for class 'linear_regression'
predict(object, ...)
```

Arguments

`object`	A `linear_regression` result.
`...`	Passed to `stats::predict.lm`.

Value

A numeric vector of predictions (or a matrix/list, depending on the arguments), as returned by `stats::predict.lm`.

`predict.logistic_regression`

Predict from a logistic_regression model

Description

For ungrouped results, dispatches to `stats::predict.glm` (the result inherits from `"glm"`). For grouped results, raises an informative error pointing at `object$groups`.

Usage

```
## S3 method for class 'logistic_regression'
predict(object, ...)
```

Arguments

`object`	A `logistic_regression` result.
`...`	Passed to `stats::predict.glm`.

Value

A numeric vector of predictions on the scale requested via `type` (link scale by default), as returned by `stats::predict.glm`.

print.ancova *Print ANCOVA results (compact)*

Description

Compact print method for objects of class "ancova". Shows factor effects and covariates with F statistics, p-values, and effect sizes.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'ancova'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "ancova" returned by ancova.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- ancova(survey_data, dv = life_satisfaction, between = gender, covariate = age)
result              # compact overview
summary(result)     # full detailed output
```

print.binomial_test *Print binomial test results (compact)*

Description

Compact print method for objects of class "binomial_test". Shows a one-line summary per variable with the observed vs. test proportion, p-value, and sample size.

For the full detailed output (category table, test statistics with confidence interval), use summary().

Usage

```
## S3 method for class 'binomial_test'
print(x, digits = 3, ...)
```

Arguments

x	A binomial_test object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

Examples

```
result <- binomial_test(survey_data, gender, p = 0.50)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.chisq_gof *Print chi-square goodness-of-fit test results (compact)*

Description

Compact print method for objects of class "chisq_gof". Shows a one-line summary per variable with the chi-square statistic, degrees of freedom, p-value, and sample size.

For the full detailed output (frequency tables with observed, expected, and residual counts), use summary().

Usage

```
## S3 method for class 'chisq_gof'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "chisq_gof"
digits	Number of decimal places (default: 3)
...	Additional arguments (currently unused)

Value

Invisibly returns the input object x.

Examples

```
result <- chisq_gof(survey_data, gender)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.chi_square *Print chi-squared test results (compact)*

Description

Compact print method for objects of class "chi_square". Shows a one-line summary per test with test statistic, p-value, effect size, and sample size.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'chi_square'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "chi_square" returned by chi_square.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- chi_square(survey_data, gender, education)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.codebook *Print a Compact Console Overview of the Codebook*

Description

Displays a concise one-line-per-variable table in the console, showing position, name, type, and label. The full HTML codebook with values, value labels, and frequencies is displayed in the RStudio Viewer.

Usage

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

Arguments

x	A codebook object from codebook()
...	Additional arguments (ignored)

Value

Invisibly returns the codebook object

Examples

```
data(survey_data)
cb <- codebook(survey_data)
print(cb)
```

---

print.crosstab *Print method for crosstab results*

Description

Prints the full cross-tabulation table (cell counts and percentages). The output of crosstab() is a contingency table by nature, so print() and summary() display the same table; summary() additionally offers section toggles (including cell residuals) and a digits option.

No significance test is included; use chi_square for a test of independence.

Usage

```
## S3 method for class 'crosstab'
print(x, digits = x$digits %||% 1, ...)
```

Arguments

x	A crosstab result object
digits	Number of decimal places for percentages (default: the digits given to crosstab, i.e. 1 unless set)
...	Additional arguments (currently unused)

Value

Invisibly returns the input object x.

Examples

```
result <- crosstab(survey_data, gender, region)
result              # full cross-tabulation table
summary(result)     # same table, with section toggles
```

print.describe *Print method for describe objects*

Description

Prints formatted descriptive statistics with professional layout. Automatically adjusts output based on whether analysis was weighted or unweighted. The output of `describe()` is a summary table by nature, so `print()` and `summary()` display the same statistics table; `summary()` additionally offers a `statistics` section toggle and a `digits` option.

Usage

```
## S3 method for class 'describe'
print(x, digits = 3, ...)
```

Arguments

`x`	A describe object
`digits`	Number of decimal places to display (default: 3)
`...`	Additional arguments (currently unused)

Value

Invisibly returns the input object `x`.

print.dunn_test *Print Dunn post-hoc test results (compact)*

Description

Compact print method for objects of class `"dunn_test"`. Shows one line per variable (or group combination) with the number of pairwise comparisons and how many are significant at the .05 level.

For the full comparison tables (Z-statistics, adjusted p-values), use `summary()`.

Usage

```
## S3 method for class 'dunn_test'
print(x, digits = 3, ...)
```

Arguments

`x`	An object of class `"dunn_test"` returned by `dunn_test`.
`digits`	Number of decimal places to display (default: 3)
`...`	Additional arguments passed to `print`. Currently unused.

Value

Invisibly returns the input object x.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction,
                         group = education) |> dunn_test()
result              # compact overview
summary(result)     # full comparison tables
```

`print.efa` *Print EFA results (compact)*

Description

Compact print method for objects of class "efa". Shows KMO value, number of factors, total variance explained, extraction method, and rotation in a concise format.

For the full detailed output including communalities, variance explained per factor, and rotated component matrices, use summary().

Usage

```
## S3 method for class 'efa'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "efa" returned by efa.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- efa(survey_data, political_orientation, environmental_concern,
              life_satisfaction, trust_government, trust_media, trust_science)
result              # compact overview
summary(result)     # full detailed output
```

print.factorial_anova *Print factorial ANOVA results (compact)*

Description

Compact print method for objects of class "factorial_anova". Shows main effects and interactions with F statistics, p-values, and effect sizes.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'factorial_anova'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "factorial_anova" returned by factorial_anova.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- factorial_anova(survey_data,
                          dv = life_satisfaction,
                          between = c(gender, education))
result              # compact overview
summary(result)     # full detailed output
```

print.fisher_test *Print Fisher's exact test results (compact)*

Description

Compact print method for objects of class "fisher_test". Shows a one-line summary with the exact p-value, significance stars, the odds ratio for 2x2 tables, and sample size.

For the full detailed output (contingency table, test results), use summary().

Usage

```
## S3 method for class 'fisher_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "fisher_test"
digits	Number of decimal places (default: 3)
...	Additional arguments (currently unused)

Value

Invisibly returns the input object x.

Examples

```
result <- fisher_test(survey_data, row = gender, col = region)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.frequency *Print method for frequency objects*

Description

Prints the full frequency tables (counts, raw/valid/cumulative percentages, missing value breakdowns). The output of frequency() is a frequency table by nature, so print() and summary() display the same tables; summary() additionally offers section toggles and a digits option.

Usage

```
## S3 method for class 'frequency'
print(x, digits = 2, ...)
```

Arguments

x	An object of class "frequency"
digits	Number of decimal places of the percentages (default: 2); the summary statistics line uses at least two decimals.
...	Additional arguments passed to print

Value

Invisibly returns the input object x.

Examples

```
result <- frequency(survey_data, gender)
result              # full frequency tables
summary(result)     # same tables, with section toggles
```

print.friedman_test *Print Friedman test results (compact)*

Description

Compact print method for objects of class "friedman_test". Shows a one-line summary with the chi-square statistic, p-value, Kendall's W, and sample size.

For the full detailed output (rank tables, test statistics), use summary().

Usage

```
## S3 method for class 'friedman_test'
print(x, digits = 3, ...)
```

Arguments

x	A friedman_test object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.kendall_tau *Print Kendall's tau results (compact)*

Description

Compact print method for objects of class "kendall_tau". Shows tau coefficient, p-value, and sample size per pair.

For the full detailed output including matrices, use summary().

Usage

```
## S3 method for class 'kendall_tau'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "kendall_tau" returned by kendall_tau.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- kendall_tau(survey_data[1:300, ], age, life_satisfaction)
result              # compact one-line overview
summary(result)     # full correlation matrices
```

---

print.kruskal_wallis *Print Kruskal-Wallis test results (compact)*

Description

Compact print method for objects of class "kruskal_wallis". Shows a one-line summary per variable with the H statistic, p-value, effect size (epsilon-squared), and sample size.

For the full detailed output (rank tables, test statistics), use summary().

Usage

```
## S3 method for class 'kruskal_wallis'
print(x, digits = 3, ...)
```

Arguments

x	A kruskal_wallis object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction, group = education)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.levene_test *Print Levene test results (compact)*

Description

Compact print method for objects of class "levene_test". Shows a one-line summary per variable with the F statistic, degrees of freedom, p-value, and the equal/unequal variances conclusion.

For the full detailed output (results tables, interpretation, recommendation), use summary().

Usage

```
## S3 method for class 'levene_test'
print(x, digits = 3, ...)
```

Arguments

x	A levene_test object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

Examples

```
result <- levene_test(survey_data, life_satisfaction, group = education)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.linear_regression *Print linear regression results (compact)*

Description

Compact print method for objects of class "linear_regression". Shows R-squared, adjusted R-squared, F statistic, and p-value.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'linear_regression'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "linear_regression" returned by linear_regression.
digits	Number of decimal places for R-squared and the p-value (the F statistic keeps one decimal fewer). (Default: 3)
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- linear_regression(survey_data, life_satisfaction ~ age + income)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.logistic_regression

Print logistic regression results (compact)

Description

Compact print method for objects of class "logistic_regression". Shows Nagelkerke R-squared, chi-squared test, and classification accuracy.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'logistic_regression'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "logistic_regression" returned by logistic_regression.
digits	Number of decimal places for the pseudo R-squared and the p-value (chi-square one decimal fewer). (Default: 3)
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction > 3)
result <- logistic_regression(survey_data, high_satisfaction ~ age + income)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.mann_whitney *Print Mann-Whitney test results (compact)*

Description

Compact print method for objects of class "mann_whitney". Shows a one-line summary per variable with test statistic, p-value, effect size, and sample size.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'mann_whitney'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "mann_whitney" returned by mann_whitney.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- mann_whitney(survey_data, life_satisfaction, group = gender)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.marginal_effects *Print average marginal effects (compact)*

Description

Compact print method for objects of class "marginal_effects": one line per predictor with the AME on the probability scale.

Usage

```
## S3 method for class 'marginal_effects'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "marginal_effects" returned by marginal_effects.
digits	Number of decimal places (default: 3).
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction >= 4)
model <- logistic_regression(survey_data, high_satisfaction ~ age + income)
marginal_effects(model)
```

print.mcnemar_test *Print McNemar test results (compact)*

Description

Compact print method for objects of class "mcnemar_test". Shows a one-line summary with the McNemar chi-square statistic, the asymptotic and exact p-values, and sample size.

For the full detailed output (2x2 contingency table, test results, discordant pairs), use summary().

Usage

```
## S3 method for class 'mcnemar_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "mcnemar_test"
digits	Number of decimal places (default: 3)
...	Additional arguments (currently unused)

Value

Invisibly returns the input object x.

Examples

```
test_data <- transform(survey_data,
  trust_gov_high = as.integer(trust_government >= 4),
  trust_media_high = as.integer(trust_media >= 4))
result <- mcnemar_test(test_data, var1 = trust_gov_high,
                       var2 = trust_media_high)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.multiple_response

Print multiple response results

Description

Prints the full SPSS-style MULT RESPONSE tables (frequencies and, when by= was used, the crosstab). The output of multiple_response() is a frequency table by nature, so print() and summary() display the same tables; summary() additionally offers section toggles and a digits option.

Usage

```
## S3 method for class 'multiple_response'
print(x, digits = 1, ...)
```

Arguments

x	An object of class "multiple_response" returned by multiple_response.
digits	Number of decimal places for percentages (default: 1).
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
d <- survey_data
d$gov <- as.integer(d$trust_government >= 4)
d$media <- as.integer(d$trust_media >= 4)
multiple_response(d, gov, media)
```

---

print.normality_test *Print normality test results (compact)*

Description

Compact print method for objects of class "normality_test". Shows one line per variable with both test results; grouped analyses show these lines under a [group] line for every group.

Usage

```
## S3 method for class 'normality_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "normality_test" returned by normality_test.
digits	Number of decimal places. (Default: 3)
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- normality_test(survey_data, age, income)
result              # compact overview
summary(result)     # full detailed output
```

print.oneway_anova *Print ANOVA test results (compact)*

Description

Compact print method for objects of class "oneway_anova". Shows a one-line summary per variable with F statistic, p-value, effect size, and sample size.

For the full detailed output, use summary().

Usage

```
## S3 method for class 'oneway_anova'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "oneway_anova" returned by oneway_anova.
digits	Number of decimal places to display (default: 3).
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.pairwise_wilcoxon *Print pairwise Wilcoxon post-hoc test results (compact)*

Description

Compact print method for objects of class "pairwise_wilcoxon". Shows one line per group combination with the number of pairwise comparisons and how many are significant at the .05 level.

For the full comparison tables (Z-statistics, adjusted p-values), use summary().

Usage

```
## S3 method for class 'pairwise_wilcoxon'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "pairwise_wilcoxon" returned by pairwise_wilcoxon.
digits	Number of decimal places to display (default: 3)
...	Additional arguments passed to print. Currently unused.

Value

Invisibly returns the input object x.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science) |> pairwise_wilcoxon()
result              # compact overview
summary(result)     # full comparison tables
```

---

print.partial_cor *Print partial correlation results (compact)*

Description

Compact print method for objects of class "partial_cor": one line per variable pair with the partial correlation, p-value, and the zero-order correlation for comparison.

Usage

```
## S3 method for class 'partial_cor'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "partial_cor" returned by partial_cor.
digits	Number of decimal places (default: 3).
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- partial_cor(survey_data, life_satisfaction, income, controls = age)
result              # compact overview
summary(result)     # full detailed output
```

print.pearson_cor *Print Pearson correlation results (compact)*

Description

Compact print method for objects of class "pearson_cor". Shows correlation coefficient, p-value, and sample size per pair.

For the full detailed output including matrices, use summary().

Usage

```
## S3 method for class 'pearson_cor'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "pearson_cor" returned by pearson_cor.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- pearson_cor(survey_data, age, life_satisfaction)
result              # compact one-line overview
summary(result)     # full correlation matrices
```

print.reliability *Print reliability results (compact)*

Description

Compact print method for objects of class "reliability". Shows Cronbach's Alpha (with quality interpretation), McDonald's Omega, and item count in a single line per group.

For the full detailed output including item statistics, inter-item correlations, and item-total statistics, use summary().

Usage

```
## S3 method for class 'reliability'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "reliability" returned by reliability.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- reliability(survey_data, trust_government, trust_media, trust_science)
result              # compact one-line overview
summary(result)     # full detailed output
```

---

print.scheffe_test *Print Scheffe test results (compact)*

Description

Compact print method for objects of class "scheffe_test". Shows one line per variable (or factor, or group combination) with the number of pairwise comparisons and how many are significant at the .05 level.

For the full comparison tables (mean differences, confidence intervals, adjusted p-values), use summary().

Usage

```
## S3 method for class 'scheffe_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "scheffe_test" returned by scheffe_test.
digits	Number of decimal places to display (default: 3)
...	Additional arguments passed to print. Currently unused.

Value

Invisibly returns the input object x.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> scheffe_test()
result              # compact overview
summary(result)     # full comparison tables
```

print.spearman_rho *Print Spearman correlation results (compact)*

Description

Compact print method for objects of class "spearman_rho". Shows rank correlation coefficient, p-value, and sample size per pair.

For the full detailed output including matrices, use summary().

Usage

```
## S3 method for class 'spearman_rho'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "spearman_rho" returned by spearman_rho.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- spearman_rho(survey_data, age, life_satisfaction)
result              # compact one-line overview
summary(result)     # full correlation matrices
```

print.summary.ancova *Print summary of ANCOVA results (detailed output)*

Description

Displays the detailed SPSS-style output for an ANCOVA, with sections controlled by the boolean parameters passed to summary.ancova. Sections include the ANCOVA table with Type III sums of squares, effect sizes, estimated marginal means, and Levene's test for homogeneity of variances.

Usage

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

Arguments

x	A summary.ancova object created by summary.ancova.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

ancova for the main analysis, summary.ancova for summary options.

Examples

```
result <- ancova(survey_data, dv = life_satisfaction, between = gender, covariate = age)
summary(result)                          # all sections
summary(result, marginal_means = FALSE)  # hide marginal means
```

---

print.summary.binomial_test

Print summary of binomial test results (detailed output)

Description

Displays the detailed SPSS-style output for a binomial test, with sections controlled by the boolean parameters passed to summary.binomial_test. Sections include the category table and the test statistics table with confidence interval.

Usage

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

Arguments

x	A summary.binomial_test object created by summary.binomial_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

binomial_test for the main analysis, summary.binomial_test for summary options.

Examples

```
result <- binomial_test(survey_data, gender, p = 0.50)
summary(result)                     # all sections
summary(result, categories = FALSE) # hide category tables
```

`print.summary.chisq_gof` *Print summary of chi-square goodness-of-fit results (detailed output)*

Description

Displays the detailed output for a chi-square goodness-of-fit test, with sections controlled by the boolean parameters passed to `summary.chisq_gof`. Sections include per-variable frequency tables and the test statistics table.

Usage

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

Arguments

`x`	A `summary.chisq_gof` object created by `summary.chisq_gof`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`chisq_gof` for the main analysis, `summary.chisq_gof` for summary options.

Examples

```
result <- chisq_gof(survey_data, gender)
summary(result)                          # all sections
summary(result, frequency_table = FALSE) # hide frequency tables
```

print.summary.chi_square

Print summary of chi-squared test results (detailed output)

Description

Displays the detailed SPSS-style output for a chi-squared test, with sections controlled by the boolean parameters passed to summary.chi_square. Sections include cross-tabulation, test results, and effect sizes (Cramer's V, Phi).

Usage

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

Arguments

x	A summary.chi_square object created by summary.chi_square.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

chi_square for the main analysis, summary.chi_square for summary options.

Examples

```
result <- chi_square(survey_data, gender, education)
summary(result)                          # all sections
summary(result, cross_tabulation = FALSE) # hide crosstab
```

print.summary.codebook

Print a Detailed Console Codebook Summary

Description

Renders the verbose codebook summary to the console with boolean-gated sections for overview, variable details, and value labels.

Usage

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

Arguments

x	A summary.codebook object from summary.codebook()
...	Additional arguments (ignored)

Value

Invisibly returns the summary object

Examples

```
data(survey_data)
cb <- codebook(survey_data)
print(summary(cb))
```

---

print.summary.crosstab

Print summary of crosstab results (detailed output)

Description

Displays the full cross-tabulation table for a crosstab result, with sections controlled by the boolean parameters passed to summary.crosstab. For grouped analyses, a separate table is displayed for each group combination.

Usage

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

Arguments

x	A summary.crosstab object created by summary.crosstab.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

crosstab for the main analysis, summary.crosstab for summary options.

Examples

```
result <- crosstab(survey_data, gender, region)
summary(result)                       # full table
summary(result, percentages = FALSE) # counts only
```

`print.summary.describe` *Print summary of describe results (detailed output)*

Description

Displays the descriptive statistics table for a `describe` result, controlled by the boolean parameter passed to `summary.describe`.

Usage

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

Arguments

`x`	A `summary.describe` object created by `summary.describe`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`describe` for the main analysis, `summary.describe` for summary options.

Examples

```
result <- describe(survey_data, age, income)
summary(result)
```

`print.summary.dunn_test` *Print summary of Dunn post-hoc test results (detailed output)*

Description

Displays the detailed output for Dunn pairwise comparisons, with sections controlled by the boolean parameters passed to `summary.dunn_test`. The display includes:

- Pairwise group comparisons with Z-statistics
- Adjusted p-values controlling for multiple comparisons
- Significance indicators (* p < 0.05, ** p < 0.01, *** p < 0.001)

For grouped analyses, results are displayed separately for each group combination.

Usage

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

Arguments

`x`	A `summary.dunn_test` object created by `summary.dunn_test`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`dunn_test` for the main analysis, `summary.dunn_test` for summary options.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction,
                         group = education) |> dunn_test()
summary(result)                       # all sections
summary(result, comparisons = FALSE)  # hide comparison tables
```

print.summary.efa *Print summary of EFA results (detailed output)*

Description

Displays the detailed SPSS-style output for an Exploratory Factor Analysis, with sections controlled by the boolean parameters passed to summary.efa. Sections include KMO and Bartlett's test, communalities, variance explained, and rotated component/pattern matrices.

Usage

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

Arguments

x	A summary.efa object created by summary.efa.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

efa for the main analysis, summary.efa for summary options.

Examples

```
result <- efa(survey_data, political_orientation, environmental_concern,
              life_satisfaction, trust_government, trust_media, trust_science)
summary(result)                       # all sections
summary(result, communalities = FALSE) # hide communalities
```

print.summary.factorial_anova *Print summary of factorial ANOVA results (detailed output)*

Description

Displays the detailed SPSS-style output for a factorial ANOVA, with sections controlled by the boolean parameters passed to summary.factorial_anova. Sections include the ANOVA table with Type III sums of squares and effect sizes, the cell descriptives, and Levene's test for homogeneity of variances.

Usage

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

Arguments

x	A summary.factorial_anova object created by summary.factorial_anova.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

factorial_anova for the main analysis, summary.factorial_anova for summary options.

Examples

```
result <- factorial_anova(survey_data,
                          dv = life_satisfaction,
                          between = c(gender, education))
summary(result)                          # all sections
summary(result, levene_test = FALSE)  # hide Levene's test
```

---

print.summary.fisher_test

Print summary of Fisher's exact test results (detailed output)

Description

Displays the detailed output for Fisher's exact test, with sections controlled by the boolean parameters passed to summary.fisher_test. Sections include the contingency table and the test results table.

Usage

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

Arguments

x	A summary.fisher_test object created by summary.fisher_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

fisher_test for the main analysis, summary.fisher_test for summary options.

Examples

```
result <- fisher_test(survey_data, row = gender, col = region)
summary(result)                            # all sections
summary(result, contingency_table = FALSE) # hide contingency table
```

---

print.summary.frequency

Print summary of frequency results (detailed output)

Description

Prints formatted frequency statistics with ASCII tables, with sections controlled by the boolean parameters passed to summary.frequency.

Usage

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

Arguments

x	A summary.frequency object created by summary.frequency.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

frequency for the main analysis, summary.frequency for summary options.

Examples

```
result <- frequency(survey_data, gender)
summary(result)                           # all sections
summary(result, frequency_table = FALSE) # only summary statistics
```

print.summary.friedman_test

Print summary of Friedman test results (detailed output)

Description

Displays the detailed SPSS-style output for a Friedman test, with sections controlled by the boolean parameters passed to summary.friedman_test. Sections include the mean rank table and the test statistics table with effect size interpretation.

Usage

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

Arguments

x	A summary.friedman_test object created by summary.friedman_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

friedman_test for the main analysis, summary.friedman_test for summary options.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science)
summary(result)               # all sections
summary(result, ranks = FALSE) # hide rank table
```

print.summary.kendall_tau

Print summary of Kendall's tau correlation results (detailed output)

Description

Displays the detailed SPSS-style output for a Kendall's tau correlation, with sections controlled by the boolean parameters passed to summary.kendall_tau. Sections include the correlation matrix, p-value matrix, and sample size matrix.

Usage

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

Arguments

x	A summary.kendall_tau object created by summary.kendall_tau.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

kendall_tau for the main analysis, summary.kendall_tau for summary options.

Examples

```
result <- kendall_tau(survey_data[1:300, ], age, life_satisfaction)
summary(result)                              # all matrices
summary(result, pvalue_matrix = FALSE)       # hide p-values
```

---

print.summary.kruskal_wallis

Print summary of Kruskal-Wallis test results (detailed output)

Description

Displays the detailed SPSS-style output for a Kruskal-Wallis test, with sections controlled by the boolean parameters passed to summary.kruskal_wallis. Sections include the per-group rank table and the test statistics table with effect size.

Usage

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

Arguments

x	A summary.kruskal_wallis object created by summary.kruskal_wallis.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

kruskal_wallis for the main analysis, summary.kruskal_wallis for summary options.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction, group = education)
summary(result)                # all sections
summary(result, ranks = FALSE) # hide rank tables
```

---

print.summary.levene_test

Print summary of Levene test results (detailed output)

Description

Displays the detailed output for Levene's test of homogeneity of variance, with sections controlled by the boolean parameters passed to summary.levene_test. Sections include the results tables, interpretation, and recommendation.

Usage

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

Arguments

x	A summary.levene_test object created by summary.levene_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

levene_test for the main analysis, summary.levene_test for summary options.

Examples

```
result <- levene_test(survey_data, life_satisfaction, group = education)
summary(result)                          # all sections
summary(result, interpretation = FALSE)  # hide interpretation
```

print.summary.linear_regression

Print summary of linear regression results (detailed output)

Description

Displays the detailed SPSS-style output for a linear regression, with sections controlled by the boolean parameters passed to summary.linear_regression. Sections include model summary (R-squared, F-test), coefficients table (B, SE, Beta, t, p), and collinearity diagnostics (Tolerance, VIF).

Usage

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

Arguments

x	A summary.linear_regression object created by summary.linear_regression.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

linear_regression for the main analysis, summary.linear_regression for summary options.

Examples

```
result <- linear_regression(survey_data, life_satisfaction ~ age + income)
summary(result)                          # all sections
summary(result, collinearity = FALSE)    # hide VIF/Tolerance
```

print.summary.logistic_regression

Print summary of logistic regression results (detailed output)

Description

Displays the detailed SPSS-style output for a logistic regression, with sections controlled by the boolean parameters passed to summary.logistic_regression. Sections include the omnibus test of model coefficients, model fit statistics (Nagelkerke R-squared, Hosmer-Lemeshow), classification table, and coefficients with odds ratios.

Usage

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

Arguments

x	A summary.logistic_regression object created by summary.logistic_regression.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

logistic_regression for the main analysis, summary.logistic_regression for summary options.

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction > 3)
result <- logistic_regression(survey_data, high_satisfaction ~ age + income)
summary(result)                         # all sections
summary(result, classification = FALSE) # hide classification table
```

---

print.summary.mann_whitney

Print summary of Mann-Whitney test results (detailed output)

Description

Displays the detailed SPSS-style output for a Mann-Whitney U test, with sections controlled by the boolean parameters passed to summary.mann_whitney. Sections include rank statistics, test results, and effect sizes (rank-biserial correlation).

Usage

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

Arguments

x	A summary.mann_whitney object created by summary.mann_whitney.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

mann_whitney for the main analysis, summary.mann_whitney for summary options.

Examples

```
result <- mann_whitney(survey_data, life_satisfaction, group = gender)
summary(result)                        # all sections
summary(result, effect_sizes = FALSE)  # hide effect sizes
```

---

print.summary.marginal_effects

Print summary of average marginal effects (detailed output)

Description

Displays the detailed AME table for a marginal_effects result, per group for grouped models.

Usage

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

Arguments

x	A summary.marginal_effects object created by summary.marginal_effects.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

marginal_effects for the main analysis, summary.marginal_effects for summary options.

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction >= 4)
model <- logistic_regression(survey_data, high_satisfaction ~ age + income)
summary(marginal_effects(model))
```

`print.summary.mcnemar_test` *Print summary of McNemar test results (detailed output)*

Description

Displays the detailed output for a McNemar test, with sections controlled by the boolean parameters passed to `summary.mcnemar_test`. Sections include the 2x2 contingency table, the test results table, and the discordant pairs.

Usage

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

Arguments

`x`	A `summary.mcnemar_test` object created by `summary.mcnemar_test`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`mcnemar_test` for the main analysis, `summary.mcnemar_test` for summary options.

Examples

```
test_data <- transform(survey_data,
  trust_gov_high = as.integer(trust_government >= 4),
  trust_media_high = as.integer(trust_media >= 4))
result <- mcnemar_test(test_data, var1 = trust_gov_high,
                       var2 = trust_media_high)
summary(result)                           # all sections
summary(result, discordant_pairs = FALSE)  # hide discordant pairs
```

print.summary.multiple_response

Print summary of multiple response results (detailed output)

Description

Displays the detailed SPSS-style MULT RESPONSE tables for a `multiple_response` result, per group for grouped data.

Usage

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

Arguments

`x`	A `summary.multiple_response` object created by `summary.multiple_response`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`multiple_response` for the main analysis, `summary.multiple_response` for summary options.

Examples

```
d <- survey_data
d$gov <- as.integer(d$trust_government >= 4)
d$media <- as.integer(d$trust_media >= 4)
summary(multiple_response(d, gov, media, by = gender))
```

print.summary.normality_test

Print summary of normality test results (detailed output)

Description

Displays the SPSS-style "Tests of Normality" table (Kolmogorov-Smirnov with Lilliefors correction, Shapiro-Wilk) for a `normality_test` result, per group for grouped data.

Usage

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

Arguments

x	A summary.normality_test object created by summary.normality_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

normality_test for the main analysis, summary.normality_test for summary options.

Examples

```
result <- normality_test(survey_data, age, income)
summary(result)
```

---

print.summary.oneway_anova

Print summary of one-way ANOVA results (detailed output)

Description

Displays the detailed SPSS-style output for a one-way ANOVA, with sections controlled by the boolean parameters passed to summary.oneway_anova. Sections include group descriptives, ANOVA table, homogeneity of variances (Levene's test), and effect sizes.

Usage

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

Arguments

x	A summary.oneway_anova object created by summary.oneway_anova.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

oneway_anova for the main analysis, summary.oneway_anova for summary options.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction, group = education)
summary(result)                        # all sections
summary(result, descriptives = FALSE)  # hide group statistics
```

---

print.summary.pairwise_wilcoxon

Print summary of pairwise Wilcoxon post-hoc test results (detailed output)

Description

Displays the detailed output for pairwise Wilcoxon comparisons, with sections controlled by the boolean parameters passed to summary.pairwise_wilcoxon. The display includes:

- Pairwise measurement comparisons with Z-statistics
- Adjusted p-values controlling for multiple comparisons
- Significance indicators (* p < 0.05, ** p < 0.01, *** p < 0.001)

For grouped analyses, results are displayed separately for each group.

Usage

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

Arguments

x	A summary.pairwise_wilcoxon object created by summary.pairwise_wilcoxon.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

pairwise_wilcoxon for the main analysis, summary.pairwise_wilcoxon for summary options.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science) |> pairwise_wilcoxon()
summary(result)                       # all sections
summary(result, comparisons = FALSE)  # hide comparison tables
```

`print.summary.partial_cor` *Print summary of partial correlation results (detailed output)*

Description

Displays the detailed SPSS-style output for a `partial_cor` result, with sections controlled by the boolean parameters passed to `summary.partial_cor`.

Usage

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

Arguments

`x`	A `summary.partial_cor` object created by `summary.partial_cor`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`partial_cor` for the main analysis, `summary.partial_cor` for summary options.

Examples

```
result <- partial_cor(survey_data, life_satisfaction, income, controls = age)
summary(result)
```

print.summary.pearson_cor

Print summary of Pearson correlation results (detailed output)

Description

Displays the detailed SPSS-style output for a Pearson correlation, with sections controlled by the boolean parameters passed to summary.pearson_cor. Sections include the correlation matrix, p-value matrix, and sample size matrix.

Usage

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

Arguments

x	A summary.pearson_cor object created by summary.pearson_cor.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

pearson_cor for the main analysis, summary.pearson_cor for summary options.

Examples

```
result <- pearson_cor(survey_data, age, life_satisfaction)
summary(result)                              # all matrices
summary(result, pvalue_matrix = FALSE)       # hide p-values
```

print.summary.reliability

Print summary of reliability analysis results (detailed output)

Description

Displays the detailed SPSS-style output for a reliability analysis, with sections controlled by the boolean parameters passed to summary.reliability. Sections include reliability statistics, item statistics, inter-item correlations, and item-total statistics.

Usage

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

Arguments

x	A summary.reliability object created by summary.reliability.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

reliability for the main analysis, summary.reliability for summary options.

Examples

```
result <- reliability(survey_data, trust_government, trust_media, trust_science)
summary(result)                                 # all sections
summary(result, inter_item_correlations = FALSE)  # hide correlations
```

---

print.summary.scheffe_test

Print summary of Scheffe test results (detailed output)

Description

Displays the detailed output for Scheffe post-hoc comparisons, with sections controlled by the boolean parameters passed to summary.scheffe_test. The display includes:

- Pairwise group comparisons with mean differences
- Confidence intervals for differences (widest among all post-hoc tests)
- Scheffe-adjusted p-values controlling family-wise error rate
- Significance indicators (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$)

For grouped analyses, results are displayed separately for each group combination. For weighted analyses, effective sample sizes are used in calculations.

Usage

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

Arguments

x	A summary.scheffe_test object created by summary.scheffe_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

scheffe_test for the main analysis, summary.scheffe_test for summary options.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> scheffe_test()
summary(result)                        # all sections
summary(result, comparisons = FALSE)  # hide comparison tables
```

---

print.summary.spearman_rho

Print summary of Spearman correlation results (detailed output)

Description

Displays the detailed SPSS-style output for a Spearman rank correlation, with sections controlled by the boolean parameters passed to summary.spearman_rho. Sections include the correlation matrix, p-value matrix, and sample size matrix.

Usage

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

Arguments

x	A summary.spearman_rho object created by summary.spearman_rho.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

spearman_rho for the main analysis, summary.spearman_rho for summary options.

Examples

```
result <- spearman_rho(survey_data, age, life_satisfaction)
summary(result)                            # all matrices
summary(result, pvalue_matrix = FALSE)     # hide p-values
```

---

print.summary.tukey_test

Print summary of Tukey HSD test results (detailed output)

Description

Displays the detailed output for Tukey HSD post-hoc comparisons, with sections controlled by the boolean parameters passed to `summary.tukey_test`. The display includes:

- Pairwise group comparisons with mean differences
- Confidence intervals for differences
- Tukey-adjusted p-values controlling family-wise error rate
- Significance indicators (* p < 0.05, ** p < 0.01, *** p < 0.001)

For grouped analyses, results are displayed separately for each group combination. For weighted analyses, effective sample sizes are used in calculations.

Usage

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

Arguments

`x`	A `summary.tukey_test` object created by `summary.tukey_test`.
`...`	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

See Also

`tukey_test` for the main analysis, `summary.tukey_test` for summary options.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> tukey_test()
summary(result)                      # all sections
summary(result, comparisons = FALSE)  # hide comparison tables
```

print.summary.t_test *Print summary of t-test results (detailed output)*

Description

Displays the detailed SPSS-style output for a t-test, with sections controlled by the boolean parameters passed to summary.t_test. Sections include group descriptives, test results (equal and unequal variances), and effect sizes.

Usage

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

Arguments

x	A summary.t_test object created by summary.t_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

t_test for the main analysis, summary.t_test for summary options.

Examples

```
result <- t_test(survey_data, life_satisfaction, group = gender)
summary(result)                        # all sections
summary(result, effect_sizes = FALSE)  # hide effect sizes
```

print.summary.wilcoxon_test *Print summary of Wilcoxon signed-rank test results (detailed output)*

Description

Displays the detailed SPSS-style output for a Wilcoxon signed-rank test, with sections controlled by the boolean parameters passed to summary.wilcoxon_test. Sections include the rank table and the test statistics table with effect size interpretation.

Usage

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

Arguments

x	A summary.wilcoxon_test object created by summary.wilcoxon_test.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

See Also

wilcoxon_test for the main analysis, summary.wilcoxon_test for summary options.

Examples

```
result <- wilcoxon_test(survey_data, x = trust_government, y = trust_media)
summary(result)                # all sections
summary(result, ranks = FALSE) # hide rank table
```

---

print.summary.w_quantile

Print summary of w_quantile results (with Kish's effective N)

Description

Print summary of w_quantile results (with Kish's effective N)

Usage

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

Arguments

x	A summary.w_quantile object.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

print.summary.w_statistic *Print summary of a w_* statistic (detailed output)*

Description

Print summary of a w_* statistic (detailed output)

Usage

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

Arguments

x	A `summary.w_statistic` object created by `summary()` on a `w_*` result.
...	Additional arguments (not used).

Value

Invisibly returns the input object `x`.

print.tukey_test *Print Tukey HSD test results (compact)*

Description

Compact print method for objects of class `"tukey_test"`. Shows one line per variable (or factor, or group combination) with the number of pairwise comparisons and how many are significant at the .05 level.

For the full comparison tables (mean differences, confidence intervals, adjusted p-values), use `summary()`.

Usage

```
## S3 method for class 'tukey_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class `"tukey_test"` returned by `tukey_test`.
digits	Number of decimal places to display (default: 3)
...	Additional arguments passed to `print`. Currently unused.

Value

Invisibly returns the input object x.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> tukey_test()
result              # compact overview
summary(result)     # full comparison tables
```

---

print.t_test *Print t-test results (compact)*

Description

Compact print method for objects of class "t_test". Shows a one-line summary per variable with test statistic, p-value, effect size, and sample size.

For the full detailed output, use summary().

Usage

```
## S3 method for class 't_test'
print(x, digits = 3, ...)
```

Arguments

x	An object of class "t_test" returned by t_test.
digits	Number of decimal places to display. Default is 3.
...	Additional arguments (not used).

Value

Invisibly returns the input object x.

Examples

```
result <- t_test(survey_data, life_satisfaction, group = gender)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.wilcoxon_test *Print Wilcoxon signed-rank test results (compact)*

Description

Compact print method for objects of class "wilcoxon_test". Shows a one-line summary per comparison with the Z statistic, p-value, effect size r, and sample size.

For the full detailed output (rank tables, test statistics), use summary().

Usage

```
## S3 method for class 'wilcoxon_test'
print(x, digits = 3, ...)
```

Arguments

x	A wilcoxon_test object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

Examples

```
result <- wilcoxon_test(survey_data, x = trust_government, y = trust_media)
result              # compact one-line overview
summary(result)     # full detailed output
```

print.w_modus *Print method for w_modus objects*

Description

Print method for w_modus objects

Usage

```
## S3 method for class 'w_modus'
print(x, digits = 3, ...)
```

Arguments

x	A w_modus object
digits	Number of decimal places to display (default: 3)
...	Additional arguments (not used)

Value

Invisibly returns the input object x.

---

profile.logistic_regression *Profile likelihood for a logistic_regression model*

Description

Dispatches to the glm method (stats::profile()); used by confint(model, method = "profile").

Usage

```
## S3 method for class 'logistic_regression'
profile(fitted, ...)
```

Arguments

fitted	A logistic_regression result (ungrouped).
...	Passed to stats::profile().

Value

A "profile" object, as returned for glm fits.

---

read_por *Read SPSS Portable Data with Tagged Missing Values*

Description

Reads an SPSS portable .por file and preserves user-defined missing values as tagged NAs. This is the portable format equivalent of read_spss() for .sav files.

Usage

```
read_por(path, tag_na = TRUE, verbose = FALSE)
```

Arguments

path	Path to an SPSS .por file.
tag_na	If TRUE (the default), user-defined missing values are converted to tagged NAs using haven::tagged_na(). If FALSE, the file is read with standard haven::read_por() behavior.
verbose	If TRUE, prints a message summarizing how many values were converted.

Details

The SPSS portable format (.por) is an older, platform-independent format. Unlike .sav files, the portable format does not support specifying a character encoding. Tagged NA handling is identical to read_spss().

Value

A tibble with the SPSS data. See read_spss() for details on tagged NA handling.

See Also

read_spss(), na_frequencies(), untag_na(), strip_tags(), haven::read_por()

Other data-import: na_frequencies(), read_sas(), read_spss(), read_stata(), read_xlsx(), read_xpt(), strip_tags(), untag_na()

Examples

```
# .por files cannot be produced from R, so this only runs when one
# is present in the working directory
if (requireNamespace("haven", quietly = TRUE) &&
    file.exists("survey.por")) {
  data <- read_por("survey.por")
  na_frequencies(data$satisfaction)
}
```

---

read_sas *Read SAS Data with Tagged Missing Values*

Description

Reads a SAS .sas7bdat file and integrates with mariposa's tagged NA system. Handles two scenarios:

1. **Native special missing values** (.A through .Z, ._): Automatically detected and annotated.
2. **Numeric missing codes** (e.g., -9, -42): When tag_na is provided, these regular values are converted to tagged NAs, giving the same result as read_spss() with tag_na = TRUE.

Usage

```
read_sas(
  path,
  catalog_file = NULL,
  encoding = NULL,
  catalog_encoding = NULL,
  tag_na = NULL,
  verbose = FALSE
)
```

Arguments

path	Path to a SAS .sas7bdat data file.
catalog_file	Path to a SAS catalog file (.sas7bcat) for value labels. If NULL, no catalog is used.
encoding	Character encoding for the data file. If NULL, haven's default encoding detection is used.
catalog_encoding	Character encoding for the catalog file. If NULL, the same encoding as the data file is used.
tag_na	Numeric vector of values to treat as missing (e.g., c(-9, -8, -42)). These values will be converted to tagged NAs across all numeric variables. Use this when SAS files contain missing codes stored as regular values. Default: NULL (only detect native SAS special missing values).
verbose	If TRUE, prints a message summarizing how many variables contain tagged missing values.

Details

Native Special Missing Values:

SAS supports 28 distinct missing value types: . (system missing), .A through .Z, and ._ (underscore). The haven package preserves these as tagged NAs automatically. read_sas() adds the na_tag_map attribute so that mariposa's tagged NA functions work seamlessly.

Numeric Missing Codes (tag_na):

Some SAS files store missing value codes as regular numeric values (e.g., -9 = "No answer"). The tag_na parameter converts these to tagged NAs, enabling proper handling in frequency(), codebook(), and other functions.

When tag_na is used, untag_na() can recover the original numeric codes.

Value

A tibble with the SAS data. Variables with missing value codes have:

- Tagged NAs for each missing type
- An "na_tag_map" attribute mapping tag characters to original codes
- is.na() returns TRUE for these values (standard R behavior)

See Also

read_xpt(), na_frequencies(), strip_tags(), untag_na(), read_spss(), read_stata(), haven::read_sas()

Other data-import: na_frequencies(), read_por(), read_spss(), read_stata(), read_xlsx(), read_xpt(), strip_tags(), untag_na()

Examples

```
# .sas7bdat files cannot be produced from R, so this only runs when
# one is present in the working directory
if (requireNamespace("haven", quietly = TRUE) &&
    file.exists("survey.sas7bdat")) {
  # Read SAS file with native special missing values
  data <- read_sas("survey.sas7bdat")

  # Read with catalog for value labels
  data <- read_sas("survey.sas7bdat", catalog_file = "formats.sas7bcat")

  # Read SAS file with numeric missing codes
  data <- read_sas("survey.sas7bdat", tag_na = c(-9, -8, -42))
}
```

---

read_spss *Read SPSS Data with Tagged Missing Values*

Description

Reads an SPSS .sav file and preserves user-defined missing values as tagged NAs instead of converting them to regular NA. This allows you to distinguish between different types of missing data (e.g., "no answer", "not applicable", "refused") while still treating them as NA in standard R operations.

Usage

```
read_spss(path, tag_na = TRUE, encoding = NULL, verbose = FALSE)
```

Arguments

path	Path to an SPSS .sav file.
tag_na	If TRUE (the default), user-defined missing values are converted to tagged NAs using haven::tagged_na(). If FALSE, the file is read with standard haven::read_sav() behavior (all missing values become regular NA).
encoding	Character encoding for the file. If NULL, haven's default encoding detection is used.
verbose	If TRUE, prints a message summarizing how many values were converted.

Details

SPSS allows defining specific values as "user-defined missing values" (e.g., -9 = "no answer", -8 = "don't know"). When reading `.sav` files with `haven::read_sav()`, these are silently converted to `NA`, losing the information about *why* a value is missing.

`read_spss()` preserves this information using haven's tagged NA system: each missing value type gets a unique tag character (a-z, A-Z, 0-9) that can be inspected with `haven::na_tag()`. The values still behave as `NA` in all standard R operations (`mean()`, `sum()`, `is.na()`, etc.).

Use the companion functions to work with the tagged NAs:

- `na_frequencies()` - Frequency table of missing types
- `untag_na()` - Convert tagged NAs back to original SPSS codes
- `strip_tags()` - Convert tagged NAs to regular NAs (drop tags)

Value

A tibble with the SPSS data. When `tag_na = TRUE`:

- User-defined missing values are stored as tagged NAs
- `is.na()` returns `TRUE` for these values (standard R behavior)
- The original SPSS missing codes can be recovered via `na_frequencies()`, `untag_na()`, or `haven::na_tag()`
- Each tagged variable has an `"na_tag_map"` attribute mapping tag characters to original SPSS codes, and an `"spss_missing"` attribute with the original missing-value definition (used by `write_spss()` for an exact round trip)

A file that is not an SPSS `.sav` file (e.g. a Stata or Excel file) is reported with its apparent type and the matching reader.

See Also

`na_frequencies()`, `untag_na()`, `strip_tags()`, `haven::read_sav()`, `frequency()`, `read_por()`

Other data-import: `na_frequencies()`, `read_por()`, `read_sas()`, `read_stata()`, `read_xlsx()`, `read_xpt()`, `strip_tags()`, `untag_na()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip through a temporary .sav file (with your own data,
  # simply pass its path instead)
  tmp <- tempfile(fileext = ".sav")
  write_spss(survey_data, tmp)
  data <- read_spss(tmp)

  # Standard R operations work normally (NAs are excluded)
  mean(data$life_satisfaction, na.rm = TRUE)

  # frequency() shows each missing type separately
  data |> frequency(life_satisfaction)
```

```
  unlink(tmp)
}
```

`read_stata` *Read Stata Data with Tagged Missing Values*

Description

Reads a Stata `.dta` file and integrates with mariposa's tagged NA system. Handles two scenarios:

1. **Native extended missing values** (`.a` through `.z`): Automatically detected and annotated.
2. **Numeric missing codes** (e.g., -9, -42): When `tag_na` is provided, these regular values are converted to tagged NAs, giving the same result as `read_spss()` with `tag_na = TRUE`.

Usage

```
read_stata(path, encoding = NULL, tag_na = NULL, verbose = FALSE)
```

Arguments

`path`	Path to a Stata `.dta` file.
`encoding`	Character encoding for the file. If `NULL`, haven's default encoding detection is used. Generally only needed for Stata 13 files and earlier.
`tag_na`	Numeric vector of values to treat as missing (e.g., `c(-9, -8, -42)`). These values will be converted to tagged NAs across all numeric variables. Use this when Stata files contain SPSS-style missing codes stored as regular values. Default: `NULL` (only detect native Stata extended missing values).
`verbose`	If `TRUE`, prints a message summarizing how many variables contain tagged missing values.

Details

Native Extended Missing Values:

Stata supports 27 distinct missing value types: `.` (system missing) and `.a` through `.z` (extended missing values). The haven package preserves these as tagged NAs automatically. `read_stata()` adds the `na_tag_map` attribute so that mariposa's tagged NA functions work seamlessly.

Numeric Missing Codes (tag_na):

Many Stata files – especially those converted from SPSS – store missing value codes as regular numeric values (e.g., -9 = "No answer", -42 = "Data error"). The `tag_na` parameter converts these to tagged NAs, enabling proper handling in `frequency()`, `codebook()`, and other functions.

When `tag_na` is used, `untag_na()` can recover the original numeric codes.

Value

A tibble with the Stata data. Variables with missing value codes have:

- Tagged NAs for each missing type
- An "na_tag_map" attribute mapping tag characters to original codes
- is.na() returns TRUE for these values (standard R behavior)

See Also

na_frequencies(), strip_tags(), untag_na(), read_spss(), read_sas(), read_xpt(), haven::read_dta()

Other data-import: na_frequencies(), read_por(), read_sas(), read_spss(), read_xlsx(), read_xpt(), strip_tags(), untag_na()

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip through a temporary .dta file (with your own data,
  # simply pass its path instead)
  tmp <- tempfile(fileext = ".dta")
  write_stata(survey_data, tmp)
  data <- read_stata(tmp)

  # Read with SPSS-style missing codes tagged as distinct NA types
  data <- read_stata(tmp, tag_na = c(-9, -8))

  # frequency() shows each missing type separately
  data |> frequency(income)

  unlink(tmp)
}
```

---

read_xlsx *Read Excel Data with Label Reconstruction*

Description

Reads an Excel (.xlsx) file and returns a tibble. When the file was created by write_xlsx(), the accompanying "Labels" sheet is automatically detected and used to reconstruct variable labels, value labels, and tagged NA metadata – giving you the same labelled structure as the original data.

For plain Excel files (without a "Labels" sheet), this function works like a convenient wrapper around the openxlsx2 reader, returning a clean tibble.

Usage

```
read_xlsx(path, sheet = NULL, labels = TRUE, verbose = FALSE, ...)
```

Arguments

path	Path to the .xlsx file.
sheet	Sheet to read. NULL (default) auto-detects: reads the "Data" sheet if present, otherwise the first non-metadata sheet. Can be a sheet name (character) or index (integer).
labels	If TRUE (default), detects a "Labels" sheet and reconstructs haven_labelled columns, factor levels, variable labels, and tagged NA metadata. Set to FALSE to read raw data without label reconstruction.
verbose	If TRUE, prints a summary of reconstructed labels and tagged NA mappings.
...	Additional arguments (currently unused).

Details

Roundtrip Workflow:

read_xlsx() is designed to work seamlessly with write_xlsx():

```
write_xlsx(data, "survey.xlsx")   # Exports data + Labels sheet
data2 <- read_xlsx("survey.xlsx") # Reconstructs labels from Labels sheet
```

The reconstructed data will have the same variable labels, value labels, and tagged NA codes as the original. System NAs (empty cells in Excel) remain as regular NA.

File Format Detection:

read_xlsx() auto-detects the type of mariposa export:

- **data.frame export** ("Data" + "Labels" sheets): Reads the "Data" sheet and applies labels.
- **list export** (multiple data sheets + "Labels" with "Sheet" column): Reads the specified or first data sheet and applies matching labels.
- **codebook export** ("Overview" + "Codebook" sheets): Warns that this is a documentation file and reads it as a plain table.
- **plain Excel**: No Labels sheet detected; returns raw data.

When to Use This:

Use read_xlsx() when you:

- Want to read back data exported with write_xlsx()
- Need a quick way to import any .xlsx file as a tibble
- Want to share labelled survey data via Excel and read it back with all metadata intact

Value

A tibble. When a mariposa-formatted Labels sheet is detected:

- Numeric columns with value labels are returned as haven_labelled vectors
- Factor columns are reconstructed with their original levels
- Variable labels are attached via attr(x, "label")
- Tagged NAs are restored with na_tag_map and na_tag_format attributes

See Also

write_xlsx() for exporting data with labels, read_spss(), read_stata(), read_sas() for importing from statistical software

Other data-import: na_frequencies(), read_por(), read_sas(), read_spss(), read_stata(), read_xpt(), strip_tags(), untag_na()

Examples

```
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  # Roundtrip: export with labels, read back with reconstruction
  tmp <- tempfile(fileext = ".xlsx")
  write_xlsx(survey_data, tmp)

  data <- read_xlsx(tmp)
  attr(data$gender, "labels")   # Value labels restored
  attr(data$gender, "label")    # Variable label restored

  # Skip label reconstruction
  raw <- read_xlsx(tmp, labels = FALSE)

  unlink(tmp)
}
```

---

read_xpt *Read SAS Transport File with Tagged Missing Values*

Description

Reads a SAS transport file (.xpt) and integrates with mariposa's tagged NA system. Handles both native SAS special missing values and user-specified numeric missing codes (via tag_na). Transport files are a platform-independent SAS data format commonly used for FDA submissions and data exchange.

Usage

```
read_xpt(path, tag_na = NULL, verbose = FALSE)
```

Arguments

path	Path to a SAS transport file (.xpt).
tag_na	Numeric vector of values to treat as missing (e.g., c(-9, -8, -42)). These values will be converted to tagged NAs across all numeric variables. Default: NULL (only detect native SAS special missing values).
verbose	If TRUE, prints a message summarizing how many variables contain tagged missing values.

Details

SAS transport files support the same special missing values as `.sas7bdat` files (`.A-.Z`, `._`). This format is self-contained and does not require a separate catalog file for value labels.

When `tag_na` is used, `untag_na()` can recover the original numeric codes.

Value

A tibble with the SAS data. See `read_sas()` for details on tagged NA handling.

See Also

`read_sas()`, `na_frequencies()`, `strip_tags()`, `untag_na()`, `haven::read_xpt()`

Other data-import: `na_frequencies()`, `read_por()`, `read_sas()`, `read_spss()`, `read_stata()`, `read_xlsx()`, `strip_tags()`, `untag_na()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip through a temporary .xpt transport file
  tmp <- tempfile(fileext = ".xpt")
  write_xpt(survey_data, tmp, version = 8)  # names longer than 8 chars
  data <- read_xpt(tmp)

  # Read with numeric missing codes tagged as distinct NA types
  data <- read_xpt(tmp, tag_na = c(-9, -8))

  unlink(tmp)
}
```

---

rec *Recode Variables Using String Syntax*

Description

`rec()` recodes values of a variable using an intuitive string syntax. It consolidates recoding, reversing, dichotomizing, and missing value handling in a single function — the R equivalent of SPSS's `RECODE` command.

Usage

```
rec(
  data,
  ...,
  rules,
  as_factor = FALSE,
```

```
  suffix = NULL,
  var_label = NULL,
  val_labels = NULL,
  as.factor = NULL,
  var.label = NULL,
  val.labels = NULL
)
```

Arguments

`data`	A data frame or numeric vector. When a data frame is passed, use `...` to select variables.
`...`	Variables to recode (tidyselect). Only used when `data` is a data frame.
`rules`	A character string defining the recoding rules (see Details).
`as_factor`	If `TRUE`, return the result as a factor. Default: `FALSE`.
`suffix`	A character string appended to the column names for the recoded variables (e.g., `"_r"`). If `NULL` (default), the original columns are overwritten in-place.
`var_label`	A new variable label. If `NULL`, the existing label is kept with `" (recoded)"` appended (once: recoding a recoded variable does not stack the suffix).
`val_labels`	A named character vector of value labels for the new values (e.g., `c("1" = "Low", "2" = "Medium", "3" = "High")`). If `NULL`, labels are taken from inline `[Label]` syntax in `rules` (if present). Explicit `val_labels` always override inline labels.
`as.factor`, `var.label`, `val.labels`	Defunct dot-case argument names, removed in mariposa 0.6.9. Calling the function with any of them is an error; use the snake_case equivalents instead. (The formals are retained only so that the old names error clearly instead of being swallowed by `....`)

Details

Recoding Syntax:

Rules are specified as a semicolon-separated string of `"old=new"` pairs:

Syntax	Meaning	Example
`"old=new"`	Single value	`"1=0; 2=1"`
`"lo:hi=new"`	Range of values	`"1:3=1; 4:6=2"`
`"a,b=new"`	List of values/ranges	`"1,2=1; 3,4:5=2"`
`"old=new [Label]"`	Inline value label	`"1:2=1 [Low]; 3:5=2 [High]"`
`"else=new"`	Catch-all for unmatched	`"1=1; else=NA"`
`"copy"`	Keep original value	`"1:3=copy; else=NA"`
`"min"/"max"`	Dynamic boundaries	`"min:3=1; 4:max=2"`
`"rev"`	Reverse scale	`"rev"`
`"rev(lo, hi)"`	Reverse a lo-hi scale	`"rev(1, 5)"`
`"dicho"`	Median split	`"dicho"`
`"dicho(x)"`	Fixed cut-point	`"dicho(3)"`
`"mean"`	Mean split	`"mean"`

"quart"	Quartile split (4 groups)	"quart"
"NA=new"	Replace NA	"NA=0; else=copy"
"val=NA"	Set values to NA	"-9=NA; -8=NA"

Rules are evaluated in order — the first matching rule wins. Keywords (else, copy, NA, min, max, rev, dicho, mean, quart) are case-insensitive. A range must be written low to high ("1:5", not "5:1"). Inline labels may contain semicolons ("1:2=1 [low; poor]").

Valid values that match no rule become NA, with a warning listing them: add "else=copy" to keep them (what SPSS's in-place RECODE does) or "else=NA" to confirm.

Missing Values of Imported Data:

Missing values keep their type (the tagged NAs of read_spss(), e.g. "no answer" vs. "not applicable") unless an "NA=..." or "else=..." rule recodes them, like SPSS's RECODE keeps user-missing codes. na_frequencies(), frequency() and write_spss() therefore still see them on the result. A haven_labelled_spss vector (from haven::read_sav(user_na = TRUE)) is first converted to this tagged-NA form, so its user-missing codes are neither reversed nor recoded as valid values.

Decimal Values:

Both single values and ranges accept decimals (e.g. "3.6=2" or "2.5:3.5=1"). Single-value matching compares values as strings (rounded to 15 significant digits), so decimal codes match reliably even when stored with floating-point representation error.

Inline Value Labels:

You can attach value labels directly in the rules string using square brackets after the new value:

"1:2=1 [Low]; 3=2 [Medium]; 4:5=3 [High]"

This is equivalent to specifying val_labels = c("1" = "Low", "2" = "Medium", "3" = "High") but more compact and self-documenting. If both inline labels and val_labels are provided, val_labels takes precedence.

Special Modes:

"rev" reverses the scale by computing hi + lo - x. The scale range lo-hi is taken from the value labels of the valid codes (together with the observed values), so an item answered only with 2-5 on a labelled 1-5 scale becomes 4-1, not 5-2. Without value labels the observed minimum and maximum are used and a message says so; set the range explicitly with "rev(lo, hi)", e.g. rules = "rev(1, 5)". Value labels are mirrored accordingly; missing values keep their type.

"dicho" dichotomizes at the median: values $\leq$ median become 0, values $>$ median become 1.

"dicho(x)" dichotomizes at a fixed cut-point x: values $\leq x$ become 0, values $> x$ become 1.

"mean" dichotomizes at the arithmetic mean: values $\leq$ mean become 0, values $>$ mean become 1.

"quart" splits into four quartile groups using quantile(): values $\leq$ Q1 become 1, Q1–Q2 become 2, Q2–Q3 become 3, and $>$ Q3 become 4. Quartile boundaries are computed unweighted.

Value

If data is a vector, a recoded vector is returned. If data is a data frame, the modified data frame is returned (invisibly). A recoded variable is haven_labelled whenever it carries value

labels or the missing-value types of an imported variable, so the labels survive `write_spss()` and `write_stata()`. With `as_factor = TRUE` the factor levels are ordered by code and named by the result's value labels (unlabelled values keep their code as level name).

See Also

`to_dummy()` for creating dummy variables, `set_na()` for declaring missing values, `to_label()` for converting to factor labels

Other recode: `to_dummy()`

Examples

```
library(dplyr)
data(survey_data)

# Collapse a 5-point scale to 3 categories (inline labels)
data <- rec(survey_data, trust_government,
            rules = "1:2=1 [Low]; 3=2 [Medium]; 4:5=3 [High]")

# Reverse a scale (with suffix to keep original)
data <- rec(survey_data, trust_government, trust_media,
            rules = "rev(1, 5)", suffix = "_r")

# Dichotomize at the median
data <- rec(survey_data, age, rules = "dicho", suffix = "_d")

# Set missing value codes to NA
data <- rec(survey_data, starts_with("trust"),
            rules = "-9=NA; -8=NA; else=copy")

# Replace NA with 0
data <- rec(survey_data, trust_government,
            rules = "NA=0; else=copy")

# Quartile split
data <- rec(survey_data, age, rules = "quart", suffix = "_q")

# Use inside mutate()
survey_data <- survey_data %>%
  mutate(
    trust_gov_3 = rec(trust_government,
                      rules = "1:2=1 [Low]; 3=2 [Medium]; 4:5=3 [High]"),
    age_q = rec(age, rules = "quart")
  )
```

`reliability` *Check How Reliably Your Scale Measures a Concept*

Description

`reliability()` calculates Cronbach's Alpha, McDonald's Omega, and detailed item statistics to evaluate whether your survey items form a reliable scale. This is the R equivalent of SPSS's `RELIABILITY /MODEL=ALPHA /STATISTICS=DESCRIPTIVE CORR /SUMMARY=TOTAL`.

For example, if you have 3 items measuring "trust", reliability analysis tells you whether these items consistently measure the same concept.

Usage

```
reliability(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The items to analyze. Use bare column names separated by commas, or tidyselect helpers like `starts_with("trust")`.
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). Uses listwise deletion (only complete cases across all items).

Details

Understanding the Results:

Cronbach's Alpha tells you how internally consistent your scale is:

- Alpha > 0.90: Excellent reliability
- Alpha 0.80 - 0.90: Good reliability
- Alpha 0.70 - 0.80: Acceptable reliability
- Alpha 0.60 - 0.70: Questionable reliability
- Alpha < 0.60: Poor reliability - reconsider items

Item-Total Correlation shows how well each item fits the scale:

- Values > 0.40: Item fits well
- Values 0.20 - 0.40: Item may need review
- Values < 0.20: Consider removing the item

Alpha if Item Deleted shows what happens if you remove an item:

- If alpha increases when removing an item, that item hurts reliability
- If alpha decreases, the item contributes to the scale

McDonald's Omega:

McDonald's Omega (omega total) is a factor-model-based reliability coefficient. Where alpha assumes every item measures the construct equally well (tau-equivalence), omega fits a one-factor model and lets each item carry its own loading. The two agree when items are roughly tau-equivalent; omega is typically slightly higher (and the more accurate estimate) when loadings differ across items, which is the common case in survey scales (Hayes & Coutts, 2020).

`reliability()` reports two variants, mirroring the two alpha variants: `omega` from the covariance metric (analogous to raw alpha) and `omega_std` from the correlation metric (analogous to standardized alpha). **Omega if Item Deleted** refits the one-factor model without each item, mirroring Alpha if Item Deleted.

A one-factor model needs at least 3 items to be identified: with fewer than 3 items the omega fields are `NA` (alpha is still computed), and Omega if Item Deleted is `NA` whenever the reduced scale would fall below 3 items. Omega is also `NA`, with a warning, when the items' correlation matrix is singular or the one-factor solution is a Heywood case (an item's uniqueness at the lower bound, i.e. a loading of about 1: the "factor" is then that single item, not the scale).

Items with zero variance are removed from the scale with a warning, as SPSS RELIABILITY does (`$removed_items`).

Weighted analyses and SPSS comparison:

Cronbach's alpha and the item statistics reproduce SPSS v29 `RELIABILITY` output, weighted and unweighted. McDonald's omega is computed from a one-factor maximum-likelihood solution (the estimator of `efa` with ML extraction) and is not compared with SPSS output. The weighted omega uses the same weighted correlation and covariance matrices as the weighted alpha and equals the unweighted omega when all weights are 1; see `vignette("spss-compatibility")`.

When to Use This:

Run `reliability()` before creating scale scores with `row_means`:

1. Select your items
2. Check reliability
3. If acceptable (alpha > .70), create the index
4. If not, review items and consider removing problematic ones

Value

A reliability result object containing:

alpha Cronbach's Alpha (unstandardized)

alpha_standardized Cronbach's Alpha based on standardized items

omega McDonald's Omega (raw/total, one-factor ML model; NA for fewer than 3 items)

omega_std Standardized Omega (correlation metric)

n_items Number of items in the scale

item_statistics Mean, SD, and N for each item

item_total Corrected Item-Total Correlation, Alpha if Item Deleted, and Omega if Item Deleted

inter_item_cor Inter-item correlation matrix

n Sample size (listwise)

variable_labels Named character vector with the items' variable labels (`NA` where an item has none); `summary()` lists them next to the item names

removed_items Items removed for zero variance, or `NULL`

negative_items When alpha is negative: the items with a negative corrected item-total correlation (candidates for reverse-coding)

Use `summary()` for the full SPSS-style output with toggleable sections.

References

McDonald, R. P. (1999). *Test Theory: A Unified Treatment*. Mahwah, NJ: Lawrence Erlbaum.

Hayes, A. F., & Coutts, J. J. (2020). Use omega rather than Cronbach's alpha for estimating reliability. But... *Communication Methods and Measures*, 14(1), 1-24.

See Also

`row_means` for creating mean indices after checking reliability.

`pearson_cor` for bivariate correlations.

`summary.reliability` for detailed output with toggleable sections.

Other scale: `efa()`, `pomps()`, `row_count()`, `row_means()`, `row_sums()`

Examples

```
library(dplyr)
data(survey_data)

# Check reliability of trust items
reliability(survey_data, trust_government, trust_media, trust_science)

# With survey weights
reliability(survey_data, trust_government, trust_media, trust_science,
            weights = sampling_weight)

# Using tidyselect helpers
reliability(survey_data, starts_with("trust"))

# Grouped by region
survey_data %>%
  group_by(region) %>%
  reliability(trust_government, trust_media, trust_science)

# --- Three-layer output ---
result <- reliability(survey_data, trust_government, trust_media, trust_science)
result              # compact one-line overview
summary(result)     # full detailed output with all sections
summary(result, inter_item_correlations = FALSE)  # hide correlations
```

---

residuals.linear_regression

Residuals of a linear_regression model

Description

Dispatches to `stats::residuals()` for the fitted `lm`; grouped and pairwise results raise an informative error.

Usage

```
## S3 method for class 'linear_regression'
residuals(object, ...)
```

Arguments

object	A linear_regression result (ungrouped, listwise).
...	Passed to the lm method.

Value

A numeric vector.

---

row_count *Count Occurrences of a Value Across Columns*

Description

Counts how often a specific value appears in each row across the selected variables. Useful for data quality checks (e.g., "How many items did a respondent answer with -9?") or for creating count-based indices.

Usage

```
row_count(data, ..., count, na.rm = TRUE)
```

Arguments

data	Your survey data (a data frame or tibble).
...	The variables to check. Supports tidyselect.
count	The value(s) to count: one value (count = 5) or a set (count = c(4, 5) counts cells equal to 4 or 5, like SPSS's COUNT n = v1 TO v5 (4, 5)). NA counts missing values (SPSS keyword MISSING). Missing codes of imported data (e.g. -9 read by read_spss() as a tagged NA) are counted when listed, as SPSS's COUNT counts user-missing values.
na.rm	If TRUE (default), NA values are ignored. If FALSE, any row containing a missing value that is not counted returns NA.

Value

An integer vector with one value per row — the count of how often count appears.

See Also

row_sums() for row-wise sums, row_means() for row-wise means

Other scale: efa(), pomps(), reliability(), row_means(), row_sums()

Examples

```
library(dplyr)
data(survey_data)

# How many items did each respondent answer with the highest value (5)?
survey_data <- survey_data %>%
  mutate(n_top = row_count(., trust_government, trust_media,
                           trust_science, count = 5))

# Top-2 box (4 or 5) and number of missing answers per respondent
survey_data <- survey_data %>%
  mutate(n_top2 = row_count(., trust_government, trust_media,
                            trust_science, count = c(4, 5)),
         n_miss = row_count(., trust_government, trust_media,
                            trust_science, count = NA))
```

`row_means` *Compute Row Means Across Items*

Description

Calculates the mean across multiple variables for each row. This is the standard way to create scale scores from survey items — the R equivalent of SPSS's `COMPUTE score = MEAN(var1, var2, var3)`.

Use `min_valid` to require a minimum number of non-missing items, mirroring SPSS's `MEAN.n()` syntax: `min_valid = 2` corresponds to `MEAN.2(a, b, c)`.

Usage

```
row_means(data, ..., min_valid = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble). Inside `mutate()`, the recommended form is `pick()`: `mutate(score = row_means(pick(item1, item2, item3)))`. It also works in a grouped `mutate()`; `.` (the magrittr placeholder) is the whole data set and therefore only works without groups.
`...`	The variables to average. Use bare column names separated by commas, or tidyselect helpers like `starts_with("trust")`. If no variables are specified, all numeric columns in `data` are used (useful with `pick()`; non-numeric columns are ignored with a warning).
`min_valid`	Minimum number of non-missing values required to compute a mean (a whole number). If a row has fewer valid values, `NA` is returned. Default is `NULL` (compute mean if at least 1 value is valid).
`na.rm`	Remove missing values before calculating? Default: `TRUE`.

Details

How It Works:

For each row, `row_means()` computes the arithmetic mean of the selected variables. Missing values are ignored by default, so a respondent who answered 2 out of 3 items still gets a score.

The min_valid Parameter:

In practice, you often want to require a minimum number of valid responses. If someone only answered 1 out of 5 items, a mean based on a single item may not be reliable. Set `min_valid` to control this:

- `min_valid` = `NULL` (default): Compute mean with any number of valid values (at least 1)
- `min_valid` = `2`: Require at least 2 valid values
- `min_valid` = `3`: Require at least 3 valid values

When to Use This:

Use `row_means()` after checking reliability with `reliability`:

1. Run `reliability()` to check if items form a reliable scale
2. If Cronbach's Alpha is acceptable (typically > .70), create the index
3. Use the index in further analyses (t-tests, correlations, regression)

Value

A numeric vector with one value per row — the mean across the selected variables. Use inside `dplyr::mutate()` to add it as a new column.

See Also

`row_sums()` for row-wise sums, `row_count()` for counting specific values, `pomps()` for rescaling to 0-100

Other scale: `efa()`, `pomps()`, `reliability()`, `row_count()`, `row_sums()`

Examples

```
library(dplyr)
data(survey_data)

# Create a trust scale from 3 items (recommended: pick())
survey_data <- survey_data %>%
  mutate(m_trust = row_means(pick(trust_government, trust_media,
                                  trust_science)))

# tidyselect helpers inside pick(); also works after group_by()
survey_data <- survey_data %>%
  group_by(region) %>%
  mutate(m_trust = row_means(pick(starts_with("trust")))) %>%
  ungroup()

# Alternative without groups: the data placeholder `.`
survey_data <- survey_data %>%
```

```
  mutate(m_trust = row_means(., trust_government, trust_media, trust_science))

# Require at least 2 valid items (like SPSS MEAN.2)
survey_data <- survey_data %>%
  mutate(m_trust = row_means(., trust_government, trust_media,
                             trust_science, min_valid = 2))
```

`row_sums` *Compute Row Sums Across Items*

Description

Calculates the sum across multiple variables for each row. This is the R equivalent of SPSS's `COMPUTE total = SUM(var1, var2, var3)`.

Use `min_valid` to require a minimum number of non-missing items, mirroring SPSS's `SUM.n()` syntax.

Usage

```
row_sums(data, ..., min_valid = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble). Inside `mutate()`, the recommended form is `pick()`: `mutate(score = row_means(pick(item1, item2, item3)))`. It also works in a grouped `mutate()`; `.` (the magrittr placeholder) is the whole data set and therefore only works without groups.
`...`	The variables to average. Use bare column names separated by commas, or tidyselect helpers like `starts_with("trust")`. If no variables are specified, all numeric columns in `data` are used (useful with `pick()`; non-numeric columns are ignored with a warning).
`min_valid`	Minimum number of non-missing values required to compute a mean (a whole number). If a row has fewer valid values, `NA` is returned. Default is `NULL` (compute mean if at least 1 value is valid).
`na.rm`	Remove missing values before calculating? Default: `TRUE`.

Details

When to Use row_sums() vs row_means():

- `row_means()`: For Likert-type scales where you want an average score (preserves the original scale range)
- `row_sums()`: For count-based scores (e.g., number of symptoms endorsed) or when you need a total score

Value

A numeric vector with one value per row — the sum across the selected variables.

See Also

`row_means()` for row-wise means, `row_count()` for counting specific values

Other scale: `efa()`, `pomps()`, `reliability()`, `row_count()`, `row_means()`

Examples

```
library(dplyr)
data(survey_data)

# Total score across items
survey_data <- survey_data %>%
  mutate(total = row_sums(., trust_government, trust_media, trust_science))

# With min_valid (like SPSS SUM.3)
survey_data <- survey_data %>%
  mutate(total = row_sums(., trust_government, trust_media,
                          trust_science, min_valid = 3))
```

---

scheffe_test *Compare All Groups More Conservatively After ANOVA*

Description

`scheffe_test()` tells you which groups differ after ANOVA, using the most conservative approach. It's like Tukey's test but even more careful about avoiding false positives.

Think of it as:

- The most cautious post-hoc test available
- A way to compare groups when sample sizes are very unequal
- Insurance against finding differences that aren't real

Usage

```
scheffe_test(x, conf.level = 0.95, ...)

## Default S3 method:
scheffe_test(x, conf.level = 0.95, ...)
```

Arguments

x	ANOVA results from oneway_anova()
conf.level	Confidence level of the simultaneous intervals (Default: 0.95). As in SPSS it corresponds to the post-hoc ALPHA (1 - alpha) and does not follow the conf.level of oneway_anova(), which (like SPSS CILEVEL) only sets the intervals of the group means.
...	Additional arguments (currently unused)

Details

Understanding the Results:

Adjusted P-values: Extra conservative to prevent false positives

- p < 0.05: Groups are significantly different (you can be very confident)
- p >= 0.05: No significant difference between these groups
- Scheffe adjustments are stricter than other methods

Confidence Intervals: Wider than Tukey's

- Do not include 0: Groups differ significantly
- Include 0: No significant difference
- Wider intervals reflect extra caution

When to Use Scheffe Test:

Use Scheffe test when:

- Your ANOVA shows significant differences (p < 0.05)
- Group sizes are very unequal
- You want to be extra cautious about false positives
- You might test complex comparisons (not just pairs)
- Sample sizes are small

Scheffe vs. Tukey:

Scheffe Test:

- Most conservative (hardest to find differences)
- Best for unequal group sizes
- Protects against all possible comparisons
- Wider confidence intervals

Tukey Test:

- Less conservative (easier to find differences)
- Best for equal group sizes
- Protects only pairwise comparisons
- Narrower confidence intervals

Reading the Output:

Example: "Group A - Group B: Diff = 3.2, p = 0.082"

- Group A's average is 3.2 units higher than Group B's

- This difference is NOT significant with Scheffe ($p > 0.05$)
- It might be significant with less conservative tests

Tips for Success:

- Scheffe may not find differences even when ANOVA does
- This is normal - it's being extra careful
- Consider Tukey if group sizes are similar
- Report which post-hoc test you used and why
- Focus on confidence intervals, not just p-values

Value

Pairwise comparison results showing:

- Which group pairs are significantly different
- Size of the difference between each pair
- Adjusted p-values (extra conservative)
- Confidence intervals for each difference

References

Scheffe, H. (1953). A method for judging all contrasts in the analysis of variance. Biometrika, 40(1-2), 87-110.

Scheffe, H. (1959). The Analysis of Variance. New York: Wiley.

See Also

oneway_anova for performing ANOVA tests.

tukey_test for Tukey HSD post-hoc tests.

levene_test for testing homogeneity of variances.

Other posthoc: dunn_test(), levene_test(), pairwise_wilcoxon(), tukey_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Perform ANOVA followed by Scheffe post-hoc test
anova_result <- survey_data %>%
  oneway_anova(life_satisfaction, group = education)

# Scheffe post-hoc comparisons
anova_result %>% scheffe_test()

# Multiple variables
anova_result_multi <- survey_data %>%
  oneway_anova(life_satisfaction, income, group = education)
```

```
anova_result_multi %>% scheffe_test()

# Weighted analysis
anova_weighted <- survey_data %>%
  oneway_anova(life_satisfaction, group = education, weights = sampling_weight)

anova_weighted %>% scheffe_test()

# Grouped analysis
anova_grouped <- survey_data %>%
  group_by(region) %>%
  oneway_anova(life_satisfaction, group = education)

anova_grouped %>% scheffe_test()

# Custom confidence level (99%)
anova_result %>% scheffe_test(conf.level = 0.99)
```

---

set_na *Declare Values as Missing*

Description

Replaces specific numeric values with NA (or tagged NAs) across one or more variables. This is essential for data cleaning workflows where missing value codes (e.g., -9, -8, 99) are stored as regular values and need to be declared as missing after import.

Usage

```
set_na(data, ..., tag = TRUE, verbose = FALSE)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Values to set as missing. Can be: • **Unnamed numeric values**: Applied to all numeric columns (e.g., `set_na(data, -9, -8)`) • **Named pairs**: Applied to specific variables (e.g., `set_na(data, income = c(-9, -8), age = -1)`) Factors are left unchanged (with a warning when named explicitly): `set_na()` works on numeric codes.
tag	If TRUE (default), uses tagged NAs to preserve distinct missing types. The resulting tagged NAs integrate with `na_frequencies()`, `frequency()`, and `codebook()`. If FALSE, replaces with regular NA.
verbose	If TRUE, prints a summary of conversions.

Details

Tagged vs. Regular NA:

When `tag = TRUE` (default), each missing value code gets a unique tag character, so you can distinguish between "No answer" (-9) and "Not applicable" (-8) in downstream analysis. This is the same system used by `read_spss()` with `tag_na = TRUE`.

When `tag = FALSE`, all specified values become regular `NA` and the distinction between different missing types is lost.

Interaction with Existing Labels:

If a value being set to missing has an existing value label, that label is preserved as a tagged NA label (when `tag = TRUE`), making it visible in `frequency()` and `codebook()` output.

Value

The modified data (invisibly for data frames, visibly for vectors).

See Also

`na_frequencies()` for inspecting missing types, `strip_tags()` for converting tagged NAs to regular NA, `untag_na()` for recovering original codes

Other labels: `copy_labels()`, `drop_labels()`, `find_var()`, `to_character()`, `to_label()`, `to_labelled()`, `to_numeric()`, `unlabel()`, `val_labels()`, `var_label()`

Examples

```
# Use regular NA instead of tagged NA (no packages required)
data <- set_na(survey_data, -9, -8, tag = FALSE)

# Tagged NAs preserve the distinction between missing types
# (requires the haven package)
if (requireNamespace("haven", quietly = TRUE)) {
  # Set -9 and -8 as missing across all numeric variables
  data <- set_na(survey_data, -9, -8)

  # Set missing for specific variables only
  data <- set_na(survey_data,
    income = c(-9, -8, -42),
    life_satisfaction = c(-9, -11)
  )

  # Check the result
  na_frequencies(data$income)
}
```

spearman_rho *Spearman's Rank Correlation Analysis*

Description

Calculates Spearman's rank correlation coefficients (rho) between variables with support for grouped data and multiple variable pairs. Provides significance testing and SPSS-compatible output formatting. Survey weights only select cases (see `weights`); the coefficient itself is unweighted.

Spearman's rho is a non-parametric measure of rank correlation that assesses monotonic relationships between variables. It is particularly suitable for ordinal data or when the assumptions of Pearson correlation are not met.

Usage

```
spearman_rho(
  data,
  ...,
  weights = NULL,
  alternative = c("two.sided", "less", "greater"),
  use = c("pairwise", "listwise"),
  na.rm = NULL
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The variables you want to correlate. List two for a single correlation or more for a correlation matrix. You can use helpers like `starts_with("trust")`. Numeric variables or ordered factors (ranked by their level order).
`weights`	Optional survey weights. They are used only to select cases (rows with a weight <= 0 or NA are left out); the rank correlation itself is computed unweighted on the remaining sample. For design-based weighted Spearman use `survey::svyolr()` or `wCorr::weightedCorr(method = "Spearman")`. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`alternative`	Direction of the test: • `"two.sided"` (default): Two-tailed test • `"less"`: One-tailed test (negative correlation) • `"greater"`: One-tailed test (positive correlation)
`use`	How to handle missing values: • `"pairwise"` (default): Pairwise deletion - each correlation uses all available cases • `"listwise"`: Listwise deletion - only complete cases across all variables
`na.rm`	Deprecated. Use `use` instead.

Details

Understanding the Results:

Spearman's rho measures the strength and direction of a monotonic relationship between two variables. Unlike Pearson correlation, it does not require a linear relationship – it just needs one variable to consistently increase (or decrease) as the other increases. Think of it as ranking your data first, then checking if the ranks tend to go together.

The rho value ranges from -1 to +1:

- **Strong positive** (0.7 to 1.0): High ranks in one variable go with high ranks in the other
- **Moderate positive** (0.3 to 0.7): Moderate tendency for ranks to increase together
- **Weak positive** (0 to 0.3): Slight tendency for ranks to increase together
- **No correlation** (near 0): No relationship between the variables
- **Negative values**: As one variable's rank increases, the other's tends to decrease

The output also provides:

- **p-value**: Probability of seeing this correlation by chance
- **n**: Number of observation pairs used
- **significance stars**: Visual indicator (*** very strong, ** strong, * moderate evidence)

When to Use This:

Choose Spearman's rho when:

- Your relationship is monotonic but not necessarily linear
- Your data has outliers (they have less impact on ranks)
- Your variables are ordinal (ordered categories)
- You are not sure if your data meets Pearson correlation assumptions
- You want to detect any monotonic trend, not just linear ones

Spearman vs. Kendall:

- **Spearman's rho** is usually larger in magnitude than Kendall's tau
- **Spearman's rho** is better for detecting linear relationships in ranks
- **Kendall's tau** is more robust and has better statistical properties
- **Spearman's rho** is more commonly reported in research

Value

Correlation results showing rank-based relationships between variables, including the rho coefficient, p-value, t-statistic, and sample size for each pair. For multiple variables, correlation, significance, and sample size matrices are also provided. Use `summary()` for the full SPSS-style output with toggleable sections.

References

Spearman, C. (1904). The proof and measurement of association between two things. *American Journal of Psychology*, 15(1), 72–101.

Lehmann, E. L. (1975). *Nonparametrics: Statistical Methods Based on Ranks*. Holden-Day.

See Also

cor with method = "spearman" for the base R implementation.

kendall_tau for Kendall's rank correlation.

pearson_cor for Pearson correlation analysis.

summary.spearman_rho for detailed output with toggleable sections.

Other correlation: kendall_tau(), partial_cor(), pearson_cor()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic correlation between two variables
survey_data %>%
  spearman_rho(life_satisfaction, political_orientation)

# Correlation matrix for multiple variables
survey_data %>%
  spearman_rho(life_satisfaction, political_orientation, trust_media)

# Weights only select cases (weight <= 0 or NA left out); rho itself is
# unweighted
survey_data %>%
  spearman_rho(age, income, weights = sampling_weight)

# Grouped correlations
survey_data %>%
  group_by(region) %>%
  spearman_rho(age, income, life_satisfaction)

# Using tidyselect helpers
survey_data %>%
  spearman_rho(starts_with("trust"), weights = sampling_weight)

# Listwise deletion for missing data
survey_data %>%
  spearman_rho(age, income, use = "listwise")

# One-tailed test
survey_data %>%
  spearman_rho(age, income, alternative = "greater")

# --- Three-layer output ---
result <- spearman_rho(survey_data, age, income, life_satisfaction)
result              # compact one-line overview
summary(result)     # full correlation, p-value, and N matrices
summary(result, pvalue_matrix = FALSE)  # hide p-values
```

std *Standardize Variables (Z-Scores)*

Description

Standardizes variables by centering on the mean and dividing by a measure of spread. Supports multiple standardization methods including robust alternatives.

When used on a grouped data frame (via `dplyr::group_by()`), standardization is performed within each group.

Usage

```
std(data, ..., method = "sd", weights = NULL, suffix = NULL, na.rm = TRUE)
```

Arguments

`data`	A data frame or numeric vector.
`...`	Variables to standardize (tidyselect). Only used when `data` is a data frame.
`method`	Standardization method: `"sd"` Standard z-score: `(x - mean) / sd` (default) `"2sd"` Gelman's 2-SD method: `(x - mean) / (2 * sd)`. Useful in regression with binary predictors. `"mad"` Robust: `(x - median) / mad`. Resistant to outliers. `"gmd"` Gini's Mean Difference: `(x - mean) / gmd`. A robust alternative.
`weights`	Optional survey weights. When provided, weighted mean and weighted SD are used for standardization. Only supported for methods `"sd"` and `"2sd"`. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`suffix`	A character string appended to column names (e.g., `"_z"`). If `NULL` (default), the original columns are overwritten.
`na.rm`	Remove missing values before computing mean and SD? Default: `TRUE`.

Details

Standardization Methods:

- **sd (default)**: Standard z-transformation. Mean = 0, SD = 1.
- **2sd**: Divides by 2 standard deviations (Gelman, 2008). This makes standardized continuous predictors comparable to binary predictors in regression.
- **mad**: Uses the Median Absolute Deviation instead of SD. Robust against outliers.
- **gmd**: Uses Gini's Mean Difference — a robust spread measure based on all pairwise absolute differences.

Weighted Standardization:

When `weights` is provided, the weighted mean and weighted standard deviation (using SPSS frequency weight formula) are used. This is only supported for methods `"sd"` and `"2sd"`. The robust methods `"mad"` and `"gmd"` do not support weights.

Group-By Standardization:

When `data` is grouped (via `group_by()`), standardization is performed separately within each group. This is useful for within-group comparisons. Weights are also subsetted per group.

Missing Values:

The result is a plain numeric variable (value labels are dropped, the variable label is kept with " (standardized)" appended). Missing values of imported data, including their SPSS missing types, become plain `NA` – as SPSS's `DESCRIPTIVES /SAVE` gives system-missing z-scores.

Value

If `data` is a vector, a standardized numeric vector. If `data` is a data frame, the modified data frame (invisibly).

See Also

`center()` for mean-centering without scaling, `pomps()` for rescaling to 0-100

Other transform: `center()`

Examples

```
library(dplyr)
data(survey_data)

# Standard z-scores
data <- std(survey_data, age, income, suffix = "_z")

# Gelman 2-SD standardization (for regression)
data <- std(survey_data, income, age, method = "2sd",
            suffix = "_z")

# Robust standardization
data <- std(survey_data, income, method = "mad", suffix = "_z")

# Weighted standardization
data <- std(survey_data, income, age,
            weights = sampling_weight, suffix = "_z")

# Group-wise standardization
data <- survey_data %>%
  group_by(region) %>%
  std(income, suffix = "_z")
```

strip_tags *Strip Tags from Tagged NAs*

Description

Converts all tagged NAs to regular (untagged) NA values, effectively removing the missing value type information. Works with data from any format: read_spss(), read_por(), read_stata(), read_sas(), or read_xpt().

Usage

```
strip_tags(x, ...)
```

Arguments

x	A numeric vector with tagged NAs, or a data frame.
...	For a data frame: the columns to convert (tidyselect). If empty, every numeric column is converted. Ignored for vectors.

Value

The input with all tagged NAs replaced by regular NA. Value labels for missing types are removed; labels for valid values and the variable label are preserved. A haven_labelled input stays haven_labelled, so the labels survive write_spss() and write_stata(). A data frame is returned with the selected columns converted.

See Also

read_spss(), read_stata(), read_sas(), read_xpt(), na_frequencies(), untag_na()

Other data-import: na_frequencies(), read_por(), read_sas(), read_spss(), read_stata(), read_xlsx(), read_xpt(), untag_na()

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  x <- set_na(c(1, 2, -9, 3, -8), -9, -8)
  # Remove tag information: all missings become plain NA
  strip_tags(x)
}
```

summary.ancova *Summary method for ANCOVA results*

Description

Creates a summary object that produces detailed output when printed, including the full ANOVA table, parameter estimates, estimated marginal means, and Levene's test.

Usage

```
## S3 method for class 'ancova'
summary(
  object,
  between_subjects = TRUE,
  parameter_estimates = TRUE,
  marginal_means = TRUE,
  levene_test = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	An ancova result object.
between_subjects	Logical. Show the ANOVA table? (Default: TRUE)
parameter_estimates	Logical. Show parameter estimates? (Default: TRUE)
marginal_means	Logical. Show estimated marginal means? (Default: TRUE)
levene_test	Logical. Show Levene's test? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.ancova object.

See Also

ancova for the main analysis function.

Examples

```
result <- ancova(survey_data, dv = life_satisfaction, between = gender, covariate = age)
summary(result)
summary(result, marginal_means = FALSE)
```

summary.binomial_test *Summary method for binomial test results*

Description

Creates a summary object that produces detailed output when printed, including the category table with counts and observed proportions, and the test statistics table with test proportion, p-value, and confidence interval.

Usage

```
## S3 method for class 'binomial_test'
summary(object, categories = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A binomial_test result object.
categories	Logical. Show the category table? (Default: TRUE)
results	Logical. Show test statistics table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.binomial_test object.

See Also

binomial_test for the main analysis function.

Examples

```
result <- binomial_test(survey_data, gender, p = 0.50)
summary(result)
summary(result, categories = FALSE)
```

summary.chisq_gof *Summary method for chi-square goodness-of-fit test results*

Description

Creates a summary object that produces detailed output when printed, including the frequency table with observed, expected, and residual counts, and the test statistics table.

Usage

```
## S3 method for class 'chisq_gof'
summary(object, frequency_table = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A chisq_gof result object.
frequency_table	Logical. Show the frequency table with observed, expected, and residual counts? (Default: TRUE)
results	Logical. Show the test statistics table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.chisq_gof object.

See Also

chisq_gof for the main analysis function.

Examples

```
result <- chisq_gof(survey_data, gender)
summary(result)
summary(result, frequency_table = FALSE)
```

`summary.chi_square` *Summary method for chi-squared test results*

Description

Creates a summary object that produces detailed output when printed, including observed and expected frequency tables, test results, and effect size measures.

Usage

```
## S3 method for class 'chi_square'
summary(
  object,
  observed = TRUE,
  expected = TRUE,
  results = TRUE,
  effect_sizes = TRUE,
  digits = 3,
  ...
)
```

Arguments

`object`	A `chi_square` result object.
`observed`	Logical. Show observed frequency table? (Default: TRUE)
`expected`	Logical. Show expected frequency table? (Default: TRUE)
`results`	Logical. Show chi-squared test results table? (Default: TRUE)
`effect_sizes`	Logical. Show effect size measures? (Default: TRUE)
`digits`	Number of decimal places for formatting (Default: 3).
`...`	Additional arguments (not used).

Value

A `summary.chi_square` object.

See Also

`chi_square` for the main analysis function.

Examples

```
result <- chi_square(survey_data, gender, education)
summary(result)
summary(result, expected = FALSE)
```

summary.codebook *Detailed Console Summary of the Codebook*

Description

Creates a detailed summary object for console display, showing per-variable information with toggleable sections. Use this when you want detailed text output without opening the HTML Viewer.

Usage

```
## S3 method for class 'codebook'
summary(
  object,
  overview = TRUE,
  variable_details = TRUE,
  value_labels = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A codebook object from codebook()
overview	Show dataset-level overview? (Default: TRUE)
variable_details	Show per-variable detail blocks? (Default: TRUE)
value_labels	Show value labels within variable details? (Default: TRUE)
digits	Number of decimal places for numeric output (Default: 3)
...	Additional arguments (ignored)

Value

A summary.codebook object (printed by print.summary.codebook())

Examples

```
data(survey_data)
cb <- codebook(survey_data)
summary(cb)
summary(cb, value_labels = FALSE)
```

`summary.crosstab` *Summary method for crosstab results*

Description

Creates a summary object that produces detailed output when printed, including the full cross-tabulation table with cell counts, marginal totals, and the requested percentage breakdowns.

Usage

```
## S3 method for class 'crosstab'
summary(
  object,
  crosstab_table = TRUE,
  percentages = TRUE,
  residuals = FALSE,
  digits = object$digits %||% 1,
  ...
)
```

Arguments

`object`	A `crosstab` result object.
`crosstab_table`	Logical. Show the cross-tabulation table? (Default: TRUE)
`percentages`	Logical. Show the percentage sub-rows inside the table (as requested via the `percentages` argument of `crosstab`)? (Default: TRUE)
`residuals`	Logical. Show the adjusted standardized residual as a sub-row in each cell (SPSS `CROSSTABS /CELLS=ASRESID`)? After a significant `chi_square` test, cells with an absolute adjusted residual above roughly 2 are the ones deviating from independence. (Default: FALSE, matching SPSS's opt-in cell display)
`digits`	Number of decimal places for percentages (Default: the `digits` given to `crosstab`, i.e. 1 unless set).
`...`	Additional arguments (not used).

Value

A `summary.crosstab` object.

See Also

`crosstab` for the main analysis function.

Examples

```
result <- crosstab(survey_data, gender, region)
summary(result)
summary(result, percentages = FALSE)
summary(result, residuals = TRUE)   # which cells drive the association?
```

`summary.describe` *Summary method for describe results*

Description

Creates a summary object that produces the descriptive statistics table when printed. Since `describe()` output is already a summary table by nature, the printed content matches `print()`; the summary layer adds the `statistics` section toggle and the `digits` formatting option for consistency with the other analysis functions.

Usage

```
## S3 method for class 'describe'
summary(object, statistics = TRUE, digits = 3, ...)
```

Arguments

`object`	A `describe` result object.
`statistics`	Logical. Show the descriptive statistics table? (Default: TRUE)
`digits`	Number of decimal places for formatting (Default: 3).
`...`	Additional arguments (not used).

Value

A `summary.describe` object.

See Also

`describe` for the main analysis function.

Examples

```
result <- describe(survey_data, age, income)
summary(result)
summary(result, digits = 2)
```

summary.dunn_test *Summary method for Dunn post-hoc test results*

Description

Creates a summary object that produces detailed output when printed, including the pairwise comparison tables with Z-statistics, unadjusted and adjusted p-values, and interpretation.

Usage

```
## S3 method for class 'dunn_test'
summary(object, comparisons = TRUE, interpretation = TRUE, digits = 3, ...)
```

Arguments

object	A dunn_test result object.
comparisons	Logical. Show the pairwise comparison tables? (Default: TRUE)
interpretation	Logical. Show the interpretation section? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.dunn_test object.

See Also

dunn_test for the main analysis function.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction,
                         group = education) |> dunn_test()
summary(result)
summary(result, interpretation = FALSE)
```

summary.efa *Summarize an exploratory factor analysis*

Description

Creates a detailed summary of an EFA result. All sections are shown by default; set individual toggles to FALSE to suppress specific sections.

Usage

```
## S3 method for class 'efa'
summary(
  object,
  kmo_bartlett = TRUE,
  communalities = TRUE,
  variance_explained = TRUE,
  unrotated_matrix = TRUE,
  rotated_matrix = TRUE,
  pattern_matrix = TRUE,
  structure_matrix = TRUE,
  factor_correlations = TRUE,
  descriptives = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	An efa result object
kmo_bartlett	Show KMO and Bartlett's test? (Default: TRUE)
communalities	Show communalities table? (Default: TRUE)
variance_explained	Show total variance explained? (Default: TRUE)
unrotated_matrix	Show unrotated component/factor matrix? (Default: TRUE)
rotated_matrix	Show rotated matrix (varimax)? (Default: TRUE)
pattern_matrix	Show pattern matrix (oblimin/promax)? (Default: TRUE)
structure_matrix	Show structure matrix (oblimin/promax)? (Default: TRUE)
factor_correlations	Show factor correlation matrix (oblimin/promax)? (Default: TRUE)
descriptives	Show the per-item descriptive statistics (mean, SD, analysis N, missing N; SPSS /PRINT UNIVARIATE)? (Default: TRUE)
digits	Number of decimal places (Default: 3)
...	Additional arguments (ignored)

Value

A `summary.efa` object (list with `$show` toggles)

See Also

`efa` for the main analysis function.

Examples

```
result <- efa(survey_data, political_orientation, environmental_concern,
              life_satisfaction, trust_government, trust_media, trust_science)
summary(result)
summary(result, communalities = FALSE)
```

---

summary.factorial_anova

Summary method for factorial ANOVA results

Description

Creates a summary object that produces detailed output when printed, including the full ANOVA table, descriptive statistics, and Levene's test.

Usage

```
## S3 method for class 'factorial_anova'
summary(
  object,
  between_subjects = TRUE,
  descriptives = TRUE,
  levene_test = TRUE,
  digits = 3,
  ...
)
```

Arguments

`object`	A `factorial_anova` result object.
`between_subjects`	Logical. Show the ANOVA table? (Default: TRUE)
`descriptives`	Logical. Show descriptive statistics? (Default: TRUE)
`levene_test`	Logical. Show Levene's test? (Default: TRUE)
`digits`	Number of decimal places for formatting (Default: 3).
`...`	Additional arguments (not used).

Value

A summary.factorial_anova object.

See Also

factorial_anova for the main analysis function.

Examples

```
result <- factorial_anova(survey_data, dv = life_satisfaction, between = c(gender, education))
summary(result)
summary(result, levene_test = FALSE)
```

---

summary.fisher_test *Summary method for Fisher's exact test results*

Description

Creates a summary object that produces detailed output when printed, including the contingency table of observed frequencies and the test results table with method, exact p-value, sample size, and (for 2x2 tables) the odds ratio with its 95\

Usage

```
## S3 method for class 'fisher_test'
summary(object, contingency_table = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A fisher_test result object.
contingency_table	Logical. Show the contingency table? (Default: TRUE)
results	Logical. Show the test results table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.fisher_test object.

See Also

fisher_test for the main analysis function.

Examples

```
result <- fisher_test(survey_data, row = gender, col = region)
summary(result)
summary(result, contingency_table = FALSE)
```

`summary.frequency` *Summary method for frequency results*

Description

Creates a summary object that produces detailed output when printed, including the per-variable summary statistics line (total N, valid N, mean, SD, skewness) and the full ASCII frequency tables with counts, raw/valid/cumulative percentages, and missing value breakdowns.

Usage

```
## S3 method for class 'frequency'
summary(object, frequency_table = TRUE, summary_stats = TRUE, digits = 2, ...)
```

Arguments

`object`	A `frequency` result object.
`frequency_table`	Logical. Show the frequency tables? (Default: TRUE)
`summary_stats`	Logical. Show the per-variable summary statistics line? (Default: TRUE)
`digits`	Number of decimal places for percentages and summary statistics (Default: 2); the statistics keep at least two decimals.
`...`	Additional arguments (not used).

Value

A `summary.frequency` object.

See Also

`frequency` for the main analysis function.

Examples

```
result <- frequency(survey_data, gender)
summary(result)
summary(result, summary_stats = FALSE)
```

summary.friedman_test *Summary method for Friedman test results*

Description

Creates a summary object that produces detailed output when printed, including the mean rank table per measurement and the test statistics table with chi-square, p-value, and Kendall's W.

Usage

```
## S3 method for class 'friedman_test'
summary(object, ranks = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A friedman_test result object.
ranks	Logical. Show the mean rank table? (Default: TRUE)
results	Logical. Show test statistics table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.friedman_test object.

See Also

friedman_test for the main analysis function.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science)
summary(result)
summary(result, ranks = FALSE)
```

summary.kendall_tau *Summary method for Kendall's tau correlation results*

Description

Creates a summary object that produces detailed output when printed, including tau matrices, p-value matrices, and sample size matrices.

Usage

```
## S3 method for class 'kendall_tau'
summary(
  object,
  correlation_matrix = TRUE,
  pvalue_matrix = TRUE,
  n_matrix = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A kendall_tau result object.
correlation_matrix	Logical. Show the tau coefficient matrix? (Default: TRUE)
pvalue_matrix	Logical. Show the p-value matrix? (Default: TRUE)
n_matrix	Logical. Show the sample size matrix? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.kendall_tau object.

See Also

kendall_tau for the main analysis function.

Examples

```
result <- kendall_tau(survey_data[1:300, ], trust_government, trust_media)
summary(result)
summary(result, pvalue_matrix = FALSE)
```

summary.kruskal_wallis *Summary method for Kruskal-Wallis test results*

Description

Creates a summary object that produces detailed output when printed, including rank statistics per group and the test statistics table with H, degrees of freedom, p-value, and epsilon-squared.

Usage

```
## S3 method for class 'kruskal_wallis'
summary(object, ranks = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A kruskal_wallis result object.
ranks	Logical. Show rank statistics per group? (Default: TRUE)
results	Logical. Show test statistics table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.kruskal_wallis object.

See Also

kruskal_wallis for the main analysis function.

Examples

```
result <- kruskal_wallis(survey_data, life_satisfaction, group = education)
summary(result)
summary(result, ranks = FALSE)
```

summary.levene_test *Summary method for Levene test results*

Description

Creates a summary object that produces detailed output when printed, including the per-variable results tables, the interpretation of the homogeneity-of-variance test, and the follow-up recommendation.

Usage

```
## S3 method for class 'levene_test'
summary(
  object,
  results = TRUE,
  interpretation = TRUE,
  recommendation = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A levene_test result object.
results	Logical. Show the test results tables? (Default: TRUE)
interpretation	Logical. Show the interpretation section? (Default: TRUE)
recommendation	Logical. Show the recommendation section (when available)? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.levene_test object.

See Also

levene_test for the main analysis function.

Examples

```
result <- levene_test(survey_data, life_satisfaction, group = education)
summary(result)
summary(result, interpretation = FALSE)
```

summary.linear_regression *Summary method for linear regression results*

Description

Creates a summary object that produces detailed output when printed, including model summary, ANOVA table, and coefficient table.

Usage

```
## S3 method for class 'linear_regression'
summary(
  object,
  model_summary = TRUE,
  anova_table = TRUE,
  coefficients = TRUE,
  conf_int = TRUE,
  collinearity = TRUE,
  descriptives = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A linear_regression result object.
model_summary	Logical. Show model summary (R, R-squared)? (Default: TRUE)
anova_table	Logical. Show ANOVA table? (Default: TRUE)
coefficients	Logical. Show coefficients table? (Default: TRUE)
conf_int	Logical. Show the confidence-interval columns for B in the coefficients table (SPSS /STATISTICS CI)? The interval level is the conf.level passed to linear_regression. (Default: TRUE)
collinearity	Logical. Show collinearity diagnostics (Tolerance, VIF per model term)? (Default: TRUE)
descriptives	Logical. Show the Descriptive Statistics table (Mean, SD, N for the dependent and predictor variables)? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.linear_regression object.

See Also

linear_regression for the main analysis function.

Examples

```
result <- linear_regression(survey_data, life_satisfaction ~ age + trust_government)
summary(result)
summary(result, descriptives = FALSE)
summary(result, conf_int = FALSE)   # hide the CI columns
```

---

summary.logistic_regression

Summary method for logistic regression results

Description

Creates a summary object that produces detailed output when printed, including omnibus test, model summary, Hosmer-Lemeshow test, classification table, and coefficient table.

Usage

```
## S3 method for class 'logistic_regression'
summary(
  object,
  omnibus_test = TRUE,
  model_summary = TRUE,
  hosmer_lemeshow = TRUE,
  classification = TRUE,
  coefficients = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A logistic_regression result object.
omnibus_test	Logical. Show omnibus test of model coefficients? (Default: TRUE)
model_summary	Logical. Show model summary (pseudo R-squared)? (Default: TRUE)
hosmer_lemeshow	Logical. Show Hosmer-Lemeshow test? (Default: TRUE)
classification	Logical. Show classification table? (Default: TRUE)
coefficients	Logical. Show coefficients table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.logistic_regression object.

See Also

logistic_regression for the main analysis function.

Examples

```
survey_data$high_satisfaction <- ifelse(survey_data$life_satisfaction >= 4, 1, 0)
result <- logistic_regression(survey_data, high_satisfaction ~ age + income)
summary(result)
summary(result, classification = FALSE)
```

---

summary.mann_whitney *Summary method for Mann-Whitney test results*

Description

Creates a summary object that produces detailed output when printed, including rank statistics per group, test results table with U, W, Z statistics, and effect size interpretation.

Usage

```
## S3 method for class 'mann_whitney'
summary(
  object,
  ranks = TRUE,
  results = TRUE,
  effect_sizes = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A mann_whitney result object.
ranks	Logical. Show rank statistics per group? (Default: TRUE)
results	Logical. Show test results table? (Default: TRUE)
effect_sizes	Logical. Show effect size output and interpretation? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.mann_whitney object.

See Also

mann_whitney for the main analysis function.

Examples

```
result <- mann_whitney(survey_data, life_satisfaction, group = gender)
summary(result)
summary(result, effect_sizes = FALSE)
```

---

summary.marginal_effects

Summary method for average marginal effects

Description

Creates a summary object that produces the detailed AME table (estimate, delta-method SE, z, p, confidence interval) when printed.

Usage

```
## S3 method for class 'marginal_effects'
summary(object, effects = TRUE, digits = 3, ...)
```

Arguments

object	A marginal_effects result object.
effects	Logical. Show the AME table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.marginal_effects object.

See Also

marginal_effects for the main analysis function.

Examples

```
survey_data$high_satisfaction <- as.integer(survey_data$life_satisfaction >= 4)
model <- logistic_regression(survey_data, high_satisfaction ~ age + income)
summary(marginal_effects(model))
```

summary.mcnemar_test *Summary method for McNemar test results*

Description

Creates a summary object that produces detailed output when printed, including the 2x2 contingency table, the test results table with asymptotic and exact p-values, and the discordant pair counts.

Usage

```
## S3 method for class 'mcnemar_test'
summary(
  object,
  contingency_table = TRUE,
  results = TRUE,
  discordant_pairs = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A mcnemar_test result object.
contingency_table	Logical. Show the 2x2 contingency table? (Default: TRUE)
results	Logical. Show the test results table? (Default: TRUE)
discordant_pairs	Logical. Show the discordant pair counts? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.mcnemar_test object.

See Also

mcnemar_test for the main analysis function.

Examples

```
test_data <- transform(survey_data,
  trust_gov_high = as.integer(trust_government >= 4),
  trust_media_high = as.integer(trust_media >= 4))
result <- mcnemar_test(test_data, var1 = trust_gov_high,
                       var2 = trust_media_high)
summary(result)
```

```
summary(result, contingency_table = FALSE)
```

`summary.multiple_response` *Summary method for multiple response results*

Description

Creates a summary object that produces the detailed SPSS-style MULT RESPONSE output when printed: the frequencies table (percent of responses and of cases) and, when by was given, the crosstab with case-based column percentages.

Usage

```
## S3 method for class 'multiple_response'
summary(object, frequencies = TRUE, crosstab = TRUE, digits = 1, ...)
```

Arguments

`object`	A `multiple_response` result object.
`frequencies`	Logical. Show the frequencies table? (Default: TRUE)
`crosstab`	Logical. Show the by-crosstab (when by was given)? (Default: TRUE)
`digits`	Number of decimal places for percentages (Default: 1).
`...`	Additional arguments (not used).

Value

A `summary.multiple_response` object.

See Also

`multiple_response` for the main analysis function.

Examples

```
d <- survey_data
d$gov <- as.integer(d$trust_government >= 4)
d$media <- as.integer(d$trust_media >= 4)
summary(multiple_response(d, gov, media))
```

summary.normality_test *Summary method for normality test results*

Description

Creates a summary object that produces the detailed SPSS-style "Tests of Normality" table when printed.

Usage

```
## S3 method for class 'normality_test'
summary(object, tests = TRUE, digits = 3, ...)
```

Arguments

object	A normality_test result object.
tests	Logical. Show the tests-of-normality table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.normality_test object.

See Also

normality_test for the main analysis function.

Examples

```
result <- normality_test(survey_data, age, income)
summary(result)
```

summary.oneway_anova *Summary method for one-way ANOVA results*

Description

Creates a summary object that produces detailed output when printed, including group descriptives, ANOVA table with Welch test, and effect sizes.

Usage

```
## S3 method for class 'oneway_anova'
summary(
  object,
  descriptives = TRUE,
  anova_table = TRUE,
  effect_sizes = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A oneway_anova result object.
descriptives	Logical. Show group descriptive statistics? (Default: TRUE)
anova_table	Logical. Show ANOVA results table and Welch test? (Default: TRUE)
effect_sizes	Logical. Show effect size measures? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.oneway_anova object.

See Also

oneway_anova for the main analysis function.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction, group = education)
summary(result)
summary(result, effect_sizes = FALSE)
```

---

summary.pairwise_wilcoxon

Summary method for pairwise Wilcoxon post-hoc test results

Description

Creates a summary object that produces detailed output when printed, including the pairwise comparison tables with Z-statistics, unadjusted and adjusted p-values, and interpretation.

Usage

```
## S3 method for class 'pairwise_wilcoxon'
summary(object, comparisons = TRUE, interpretation = TRUE, digits = 3, ...)
```

Arguments

object	A pairwise_wilcoxon result object.
comparisons	Logical. Show the pairwise comparison tables? (Default: TRUE)
interpretation	Logical. Show the interpretation section? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.pairwise_wilcoxon object.

See Also

pairwise_wilcoxon for the main analysis function.

Examples

```
result <- friedman_test(survey_data, trust_government, trust_media,
                        trust_science) |> pairwise_wilcoxon()
summary(result)
summary(result, interpretation = FALSE)
```

---

summary.partial_cor *Summary method for partial correlation results*

Description

Creates a summary object that produces detailed output when printed: the pairwise partial-correlation table (with zero-order comparison, df, t, and p) and — for three or more variables — the partial-correlation matrix.

Usage

```
## S3 method for class 'partial_cor'
summary(object, pairwise = TRUE, matrix = TRUE, digits = 3, ...)
```

Arguments

object	A partial_cor result object.
pairwise	Logical. Show the pairwise results table? (Default: TRUE)
matrix	Logical. Show the partial-correlation matrix (three or more analysis variables)? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.partial_cor object.

See Also

partial_cor for the main analysis function.

Examples

```
result <- partial_cor(survey_data, trust_government, trust_media,
                      trust_science, controls = age)
summary(result)
summary(result, matrix = FALSE)
```

---

summary.pearson_cor *Summary method for Pearson correlation results*

Description

Creates a summary object that produces detailed output when printed, including correlation matrices, p-value matrices, and sample size matrices.

Usage

```
## S3 method for class 'pearson_cor'
summary(
  object,
  correlation_matrix = TRUE,
  pvalue_matrix = TRUE,
  n_matrix = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A pearson_cor result object.
correlation_matrix	Logical. Show the correlation coefficient matrix? (Default: TRUE)
pvalue_matrix	Logical. Show the p-value matrix? (Default: TRUE)
n_matrix	Logical. Show the sample size matrix? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.pearson_cor object.

See Also

pearson_cor for the main analysis function.

Examples

```
result <- pearson_cor(survey_data, trust_government, trust_media)
summary(result)
summary(result, pvalue_matrix = FALSE)
```

---

summary.reliability *Summarize a reliability analysis*

Description

Creates a detailed summary of a reliability analysis result. All sections are shown by default; set individual toggles to FALSE to suppress specific sections.

Usage

```
## S3 method for class 'reliability'
summary(
  object,
  reliability_statistics = TRUE,
  item_statistics = TRUE,
  inter_item_correlations = TRUE,
  item_total_statistics = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A reliability result object
reliability_statistics	Show Cronbach's Alpha and McDonald's Omega statistics? (Default: TRUE)
item_statistics	Show per-item means and SDs? (Default: TRUE)
inter_item_correlations	Show inter-item correlation matrix? (Default: TRUE)
item_total_statistics	Show item-total statistics? (Default: TRUE)
digits	Number of decimal places (Default: 3)
...	Additional arguments (ignored)

Value

A summary.reliability object (list with $show toggles)

See Also

reliability for the main analysis function.

Examples

```
result <- reliability(survey_data, trust_government, trust_media, trust_science)
summary(result)
summary(result, inter_item_correlations = FALSE)
```

---

summary.scheffe_test *Summary method for Scheffe test results*

Description

Creates a summary object that produces detailed output when printed, including the pairwise comparison tables with mean differences, confidence intervals, Scheffe-adjusted p-values, and interpretation.

Usage

```
## S3 method for class 'scheffe_test'
summary(
  object,
  parameters = TRUE,
  comparisons = TRUE,
  interpretation = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A scheffe_test result object.
parameters	Logical. Show test parameters (confidence level, method notes)? (Default: TRUE)
comparisons	Logical. Show the pairwise comparison tables? (Default: TRUE)
interpretation	Logical. Show the interpretation section? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.scheffe_test object.

See Also

scheffe_test for the main analysis function.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> scheffe_test()
summary(result)
summary(result, interpretation = FALSE)
```

---

summary.spearman_rho *Summary method for Spearman correlation results*

Description

Creates a summary object that produces detailed output when printed, including rho matrices, p-value matrices, and sample size matrices.

Usage

```
## S3 method for class 'spearman_rho'
summary(
  object,
  correlation_matrix = TRUE,
  pvalue_matrix = TRUE,
  n_matrix = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A spearman_rho result object.
correlation_matrix	Logical. Show the rho coefficient matrix? (Default: TRUE)
pvalue_matrix	Logical. Show the p-value matrix? (Default: TRUE)
n_matrix	Logical. Show the sample size matrix? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.spearman_rho object.

See Also

spearman_rho for the main analysis function.

Examples

```
result <- spearman_rho(survey_data, trust_government, trust_media)
summary(result)
summary(result, pvalue_matrix = FALSE)
```

---

summary.tukey_test *Summary method for Tukey HSD test results*

Description

Creates a summary object that produces detailed output when printed, including the pairwise comparison tables with mean differences, confidence intervals, Tukey-adjusted p-values, and interpretation.

Usage

```
## S3 method for class 'tukey_test'
summary(
  object,
  parameters = TRUE,
  comparisons = TRUE,
  interpretation = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A tukey_test result object.
parameters	Logical. Show test parameters (confidence level, method notes)? (Default: TRUE)
comparisons	Logical. Show the pairwise comparison tables? (Default: TRUE)
interpretation	Logical. Show the interpretation section? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.tukey_test object.

See Also

tukey_test for the main analysis function.

Examples

```
result <- oneway_anova(survey_data, life_satisfaction,
                       group = education) |> tukey_test()
summary(result)
summary(result, interpretation = FALSE)
```

---

summary.t_test *Summary method for t-test results*

Description

Creates a summary object that produces detailed output when printed, including group descriptives, test results with both variance assumptions, effect sizes, and confidence intervals.

Usage

```
## S3 method for class 't_test'
summary(
  object,
  descriptives = TRUE,
  results = TRUE,
  effect_sizes = TRUE,
  digits = 3,
  ...
)
```

Arguments

object	A t_test result object.
descriptives	Logical. Show group descriptive statistics? (Default: TRUE)
results	Logical. Show test results table? (Default: TRUE)
effect_sizes	Logical. Show effect size measures? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A summary.t_test object.

See Also

t_test for the main analysis function.

Examples

```
result <- t_test(survey_data, life_satisfaction, group = gender)
summary(result)
summary(result, effect_sizes = FALSE)
```

---

summary.wilcoxon_test *Summary method for Wilcoxon signed-rank test results*

Description

Creates a summary object that produces detailed output when printed, including the rank table (negative/positive ranks, ties) and the test statistics table with Z, p-value, and effect size r.

Usage

```
## S3 method for class 'wilcoxon_test'
summary(object, ranks = TRUE, results = TRUE, digits = 3, ...)
```

Arguments

object	A wilcoxon_test result object.
ranks	Logical. Show the rank table? (Default: TRUE)
results	Logical. Show test statistics table? (Default: TRUE)
digits	Number of decimal places for formatting (Default: 3).
...	Additional arguments (not used).

Value

A `summary.wilcoxon_test` object.

See Also

`wilcoxon_test` for the main analysis function.

Examples

```
result <- wilcoxon_test(survey_data, x = trust_government, y = trust_media)
summary(result)
summary(result, ranks = FALSE)
```

---

survey_data *Social Survey Data (Synthetic)*

Description

A synthetic dataset modeled after large-scale social surveys with demographics, attitudes, and survey design features. Created for testing and demonstration purposes without licensing concerns.

Usage

```
survey_data
```

Format

A data frame with 2,500 rows and 16 variables:

id Unique identifier (1-2500)

age Age in years (18-95)

gender Gender (Male, Female)

region Region (East, West)

education Education level (Basic Secondary, Intermediate Secondary, Academic Secondary, University)

income Monthly household income in EUR (800-15000)

employment Employment status (Student, Employed, Unemployed, Retired, Other)

political_orientation Political orientation (1=very left to 5=very right)

environmental_concern Environmental concern (1=not concerned to 5=very concerned)

life_satisfaction Life satisfaction (1=very dissatisfied to 5=very satisfied)

trust_government Trust in government (1=no trust to 5=complete trust)

trust_media Trust in media (1=no trust to 5=complete trust)

trust_science Trust in science (1=no trust to 5=complete trust)

sampling_weight Post-stratification sampling weight (0.7-1.4)

stratum Stratification variable combining region and age group

interview_mode Interview mode (Face-to-face, Telephone, Online)

Details

This dataset contains realistic patterns of correlation between variables:

- Education correlates with income and political attitudes
- Regional differences reflect East/West patterns
- Age effects on employment and attitudes
- Realistic missing data patterns (3-12% depending on sensitivity)
- Survey weights for post-stratification adjustment

The data includes proper variable labels and follows social survey conventions for coding and structure. Generated with set.seed(2024) for reproducibility.

Source

Generated synthetically using realistic demographic and attitudinal patterns. No real survey data was used.

See Also

describe for descriptive statistics of numeric variables.

frequency for categorical frequency tables.

t_test for group comparisons.

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic descriptive statistics
survey_data %>% describe(age, income, weights = sampling_weight)

# Frequency analysis
survey_data %>% frequency(education, region, weights = sampling_weight)

# Group comparisons
survey_data %>%
  group_by(region) %>%
  t_test(life_satisfaction, group = gender, weights = sampling_weight)
```

to_character *Convert Labelled Variables to Character*

Description

Converts haven_labelled variables to character vectors using value labels as text. Same as to_label() but returns character instead of factor.

Usage

```
to_character(
  data,
  ...,
  drop_na = TRUE,
  add_non_labelled = FALSE,
  drop.na = NULL,
  add.non.labelled = NULL
)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Optional: unquoted variable names (tidyselect supported). If empty, converts every haven_labelled column whose values are all value-labelled (metric variables are left unchanged with a message, as in to_label()).
drop_na	If TRUE (default), tagged NAs become regular NA.
add_non_labelled	If TRUE, unlabelled values are included as their numeric string representation. Default: FALSE.
drop.na, add.non.labelled	Defunct dot-case argument names, removed in mariposa 0.6.9. Calling the function with any of them is an error; use the snake_case equivalents instead. (The formals are retained only so that the old names error clearly instead of being swallowed by ....)

Details

This is a convenience wrapper around to_label() that calls as.character() on the result. Use this when you need character output (e.g., for string operations) rather than factor output.

The variable label ("label" attribute) is preserved on the output.

Value

The input with labelled variables converted to character. For single vector input, returns a character vector.

See Also

to_label() for factor output, val_labels()

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_label(), to_labelled(), to_numeric(), unlabel(), val_labels(), var_label()

Examples

```
# Convert a single variable to character
to_character(survey_data$gender)

# Convert specific columns in a data frame
data <- to_character(survey_data, gender, region)
```

---

to_dummy *Create Dummy Variables (One-Hot Encoding)*

Description

Creates 0/1 dummy variables from categorical variables. Column names are derived from value labels when available, making results readable for SPSS-style data.

Unlike model.matrix(), to_dummy() uses value labels for column naming and handles haven_labelled vectors correctly.

Usage

```
to_dummy(data, ..., suffix = "val", ref = NULL, append = TRUE)
```

Arguments

data	A data frame or vector. When a data frame is passed, use ... to select variables.
...	Variables to dummy-code (tidyselect). Only used when data is a data frame.
suffix	How to name dummy columns: "val" (default) uses the raw value (e.g., gender_1), "label" uses the value label (e.g., gender_Male).
ref	A value to use as reference category (omitted from output). If NULL (default), all categories get a dummy variable. Set to a specific value for n-1 coding (e.g., for regression). For a factor, give a level name (ref = "Male") or a number: the original code of a to_label() factor, otherwise the level position (ref = 1 = first level). A ref that matches no category is an error.
append	If TRUE (default), the dummy columns are appended to the original data frame. If FALSE, only the dummy columns are returned. Ignored when data is a vector.

Details

Column Naming:

With `suffix = "val"`: `{varname}_{value}` (e.g., `gender_1`, `gender_2`). With `suffix = "label"`: `{varname}_{label}` where labels are cleaned (umlauts transliterated, e.g. "männlich" -> `maennlich`; spaces replaced with _; other special characters removed). Values without a usable label (e.g. "..") use their value; labels shared by several values get the value appended, so every category keeps its own column.

Reference Category:

For regression, you typically need n-1 dummy variables. Set `ref` to the value of the reference category to omit it.

Value

If `append = TRUE` (default), the original data frame with dummy columns appended. If `append = FALSE`, a tibble with only the dummy columns. For vector input, always a tibble of dummy columns.

See Also

`rec()` for general recoding, `to_label()` for converting to factor

Other recode: `rec()`

Examples

```
library(dplyr)
data(survey_data)

# Create dummies and append to data (default)
data <- to_dummy(survey_data, gender)

# Use labels for column names
data <- to_dummy(survey_data, gender, suffix = "label")

# n-1 dummies with reference category
data <- to_dummy(survey_data, gender, ref = 1)

# Multiple variables
data <- to_dummy(survey_data, gender, education, suffix = "label")

# Return only the dummy columns (without original data)
dummies <- to_dummy(survey_data, gender, append = FALSE)
```

to_label *Convert Labelled Variables to Factors*

Description

Converts haven_labelled variables to factors, using value labels as factor levels. This is the primary function for making labelled survey data ready for plotting, cross-tabulation, and regression analysis.

Usage

```
to_label(
  data,
  ...,
  ordered = FALSE,
  drop_na = TRUE,
  drop_unused = FALSE,
  add_non_labelled = FALSE,
  drop.na = NULL,
  drop.unused = NULL,
  add.non.labelled = NULL
)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Optional: unquoted variable names (tidyselect supported). If empty, converts every haven_labelled column whose values are all value-labelled; metric variables (e.g. age with labels only for its missing codes), weights and partially labelled variables are left unchanged with a message. Selected variables are always converted.
ordered	If TRUE, creates an ordered factor. Default: FALSE.
drop_na	If TRUE (default), tagged NAs are converted to regular NA (excluded from factor levels). If FALSE, tagged NAs are kept as factor levels with their label text.
drop_unused	If TRUE, removes factor levels with zero observations. Default: FALSE.
add_non_labelled	If TRUE, values without labels are included as factor levels using their numeric value as the level name. Default: FALSE (unlabelled values become NA, with a warning naming the variables).
drop.na, drop.unused, add.non.labelled	Defunct dot-case argument names, removed in mariposa 0.6.9. Calling the function with any of them is an error; use the snake_case equivalents instead. (The formals are retained only so that the old names error clearly instead of being swallowed by ....)

Details

For each labelled variable, the numeric codes are replaced by their associated value labels. The resulting factor levels are ordered by the original numeric values (not alphabetically). Distinct codes that share a label text (e.g. ".." for the unlabelled points of a scale) stay distinct levels with their code appended: `".. (2)"`, `".. (3)"`.

The original code of every level is kept in the factor's `"codes"` attribute (a named numeric vector, names = levels), so `to_numeric()` and `to_labelled()` can restore the original values (e.g. 6, 42, 90) instead of renumbering the levels 1, 2, 3. The attribute survives dplyr verbs (`filter()`, `mutate()`, `arrange()`, ...) but, like the variable label, is dropped by base subsetting (`x[i]`) and no longer applies once the levels are renamed or extended.

When to Use This:

Use `to_label()` when you:

- Want human-readable labels in plots (ggplot2)
- Need factor variables for regression models
- Want to create frequency tables with label text

Value

The input with labelled variables converted to factors. For single vector input, returns a factor.

See Also

`to_character()` for character output, `to_labelled()` for the reverse operation, `val_labels()` for viewing labels

Other labels: `copy_labels()`, `drop_labels()`, `find_var()`, `set_na()`, `to_character()`, `to_labelled()`, `to_numeric()`, `unlabel()`, `val_labels()`, `var_label()`

Examples

```
# Convert a single variable
to_label(survey_data$gender)

# Convert specific columns in a data frame
data <- to_label(survey_data, gender, region)

# Convert all labelled columns
data <- to_label(survey_data)

# Keep values without labels as factor levels
to_label(survey_data$life_satisfaction, add_non_labelled = TRUE)
```

to_labelled *Convert Variables to Labelled Format*

Description

Converts factors, character vectors, or plain numeric vectors to haven_labelled class, optionally assigning value labels and a variable label. This is the reverse of to_label().

Usage

```
to_labelled(data, ..., labels = NULL, label = NULL)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Optional: unquoted variable names (tidyselect supported). If empty on a data frame, converts all factor columns.
labels	Optional: a named numeric vector of value labels to assign. If NULL and the input is a factor, factor levels are used as labels.
label	Optional: a variable label string.

Details

Factor Conversion:

When converting a factor created by to_label(), the original codes stored in its "codes" attribute are restored (the round trip to_labelled(to_label(x)) returns the original values and labels). For other factors, or when labels is supplied, the integer codes (1, 2, 3, ...) become the numeric values and the factor levels become the value labels.

Character Conversion:

Character vectors are converted to numeric (sequential integers) with the unique character values as labels.

Value

The input with variables converted to haven_labelled. For single vector input, returns a haven_labelled vector.

See Also

to_label() for the reverse operation, val_labels() for setting labels on existing labelled vectors

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_character(), to_label(), to_numeric(), unlabel(), val_labels(), var_label()

Examples

```
# Factor -> haven_labelled
x <- factor(c("Male", "Female", "Male"))
to_labelled(x)

# Numeric with custom labels
to_labelled(c(1, 2, 3),
  labels = c("Low" = 1, "Medium" = 2, "High" = 3),
  label = "Satisfaction level"
)

# All factors in a data frame
data <- to_labelled(survey_data)
```

---

to_numeric *Convert Factors or Labelled Variables to Numeric*

Description

Converts factor levels or labelled values to numeric. For factors, uses the underlying integer codes (or the numeric value of levels if they are numeric strings). For labelled variables, extracts the underlying numeric values.

Usage

```
to_numeric(
  data,
  ...,
  use_labels = TRUE,
  start_at = NULL,
  keep_labels = FALSE,
  use.labels = NULL,
  start.at = NULL,
  keep.labels = NULL
)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Optional: unquoted variable names (tidyselect supported). If empty, converts all factor columns.
use_labels	If TRUE (default), restores the original codes of a factor created by to_label(), otherwise uses the numeric value of factor levels (e.g., level "3" becomes 3). If FALSE, uses sequential integers (1, 2, 3, ...).
start_at	If not NULL, the lowest numeric value in the output starts at this number. Default: NULL (use original values).

keep_labels
: If TRUE, the result is a haven_labelled vector: the former factor levels (or the existing value labels and missing-value types of a labelled input) are kept, so they survive write_spss(). Default: FALSE (plain numeric; missing-value types become NA).

use.labels, start.at, keep.labels
: Defunct dot-case argument names, removed in mariposa 0.6.9. Calling the function with any of them is an error; use the snake_case equivalents instead. (The formals are retained only so that the old names error clearly instead of being swallowed by ....)

Details

This function handles four input types:

1. **Factors from** to_label(): Restores the original codes stored in the "codes" attribute, so a to_label() / to_numeric() round trip returns the original values (6, 42, 90, ...) instead of 1, 2, 3.
2. **Numeric factors** (levels like "1", "2", "3"): Extracts the numeric values from the level names.
3. **Other text factors** (levels like "Male", "Female"): Converts to sequential integers (1, 2, 3, ... in level order). Use use_labels = FALSE to force this behavior for any factor.
4. **haven_labelled**: Extracts the underlying numeric vector, stripping the labelled class. Missing-value types of imported data (tagged NAs) become plain NA.

Value

The input with variables converted to numeric. For single vector input, returns a numeric vector.

See Also

to_label() for the reverse (numeric -> factor), to_labelled() for creating labelled vectors

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_character(), to_label(), to_labelled(), unlabel(), val_labels(), var_label()

Examples

```
# Numeric factor levels -> numeric
x <- factor(c("1", "3", "5", "3"))
to_numeric(x)
# [1] 1 3 5 3

# Sequential integers
to_numeric(x, use_labels = FALSE)
# [1] 1 2 3 2

# Haven labelled -> plain numeric
to_numeric(survey_data$life_satisfaction)
```

`tukey_test` *Find Which Specific Groups Differ After ANOVA*

Description

`tukey_test()` tells you exactly which groups are different from each other after ANOVA finds overall differences. It's like a follow-up investigation that pinpoints where the differences lie.

Think of it as:

- ANOVA says "there are differences somewhere"
- Tukey test says "specifically, Group A differs from Group C"
- A way to make all possible comparisons while controlling error rates

Usage

```
tukey_test(x, conf.level = 0.95, ...)

## Default S3 method:
tukey_test(x, conf.level = 0.95, ...)
```

Arguments

`x`	ANOVA results from `oneway_anova()`
`conf.level`	Confidence level of the simultaneous intervals (Default: 0.95). As in SPSS it corresponds to the post-hoc `ALPHA` (1 - alpha) and does not follow the `conf.level` of `oneway_anova()`, which (like SPSS `CILEVEL`) only sets the intervals of the group means.
`...`	Additional arguments (currently unused)

Details

Understanding the Results:

Adjusted P-values: Control for multiple comparisons

- $p < 0.05$: Groups are significantly different
- $p >= 0.05$: No significant difference between these groups
- When you make many comparisons, chance alone could produce false positives
- Tukey adjustment protects against this by being more conservative

Mean Differences:

- Positive: First group has higher average than second
- Negative: Second group has higher average than first
- Zero in confidence interval: No significant difference

When to Use Tukey Test:

Use Tukey test when:

- Your ANOVA shows significant differences ($p < 0.05$)
- You want to know which specific groups differ
- You need to compare all possible pairs
- Group sizes are roughly equal
- Variances are roughly equal across groups

Tukey vs. Scheffe:

Tukey Test:

- Less conservative (easier to find differences)
- Best for equal group sizes
- Protects only pairwise comparisons
- Narrower confidence intervals

Scheffe Test:

- Most conservative (hardest to find differences)
- Best for unequal group sizes
- Protects against all possible comparisons
- Wider confidence intervals

Reading the Output:

Example: "Group A - Group B: Diff = 3.2, p = 0.012"

- Group A's average is 3.2 units higher than Group B's
- This difference is statistically significant ($p < 0.05$)
- You can be confident these groups truly differ

Tips for Success:

- Only run post-hoc tests if ANOVA is significant
- Focus on comparisons that make theoretical sense
- Consider practical significance, not just statistical
- Report both the difference and its confidence interval
- Remember: non-significant doesn't mean "exactly equal"

Value

Pairwise comparison results showing:

- Which group pairs are significantly different
- Size of the difference between each pair
- Adjusted p-values (controlling for multiple comparisons)
- Confidence intervals for each difference

References

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. Biometrics, 5(2), 99-114.

Kramer, C. Y. (1956). Extension of multiple range tests to group means with unequal numbers of replications. Biometrics, 12(3), 307-310.

See Also

oneway_anova for performing ANOVA tests.

TukeyHSD for the base R Tukey HSD function.

levene_test for testing homogeneity of variances.

Other posthoc: dunn_test(), levene_test(), pairwise_wilcoxon(), scheffe_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Perform ANOVA followed by Tukey post-hoc test
anova_result <- survey_data %>%
  oneway_anova(life_satisfaction, group = education)

# Tukey post-hoc comparisons
anova_result %>% tukey_test()

# With weights
anova_weighted <- survey_data %>%
  oneway_anova(life_satisfaction, group = education, weights = sampling_weight)

anova_weighted %>% tukey_test()

# Multiple variables
anova_multi <- survey_data %>%
  oneway_anova(trust_government, trust_science, group = education)

anova_multi %>% tukey_test()

# Grouped analysis
anova_grouped <- survey_data %>%
  group_by(region) %>%
  oneway_anova(life_satisfaction, group = education)

anova_grouped %>% tukey_test()
```

---

t_test *Test If Two Groups Differ*

Description

t_test() helps you determine if two groups have different average values. For example, do men and women have different satisfaction scores? Does Region A spend more than Region B? Is this year better than last year?

The test tells you:

- **Whether the difference is statistically significant**
- **How big the difference is** (effect sizes)
- **The likely range of the true difference** (confidence interval)

Usage

```
t_test(
  data,
  ...,
  group = NULL,
  weights = NULL,
  var.equal = FALSE,
  mu = 0,
  alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables to test. List multiple variables or use tidyselect helpers like `starts_with("trust")`.
`group`	The variable that defines your two groups (e.g., gender, region). Must have exactly two categories. Leave empty for one-sample tests.
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`var.equal`	Should we assume equal variances? (Default: FALSE) • `FALSE`: Safer, works even if groups vary differently (Welch's test) • `TRUE`: Traditional approach, assumes equal spread (Student's test)
`mu`	Comparison value (Default: 0). For two-sample tests, usually 0. For one-sample tests, your benchmark value.
`alternative`	Direction of the test: • `"two.sided"` (default): Any difference • `"greater"`: First group is higher • `"less"`: First group is lower
`conf.level`	Confidence level for intervals (Default: 0.95 = 95%)

Details

Understanding the Results:

P-value: Is the difference statistically significant?

- $p < 0.001$: Very strong evidence of a difference
- $p < 0.01$: Strong evidence of a difference

- p < 0.05: Moderate evidence of a difference
- p >= 0.05: No significant difference found

Effect Sizes (How big is the difference?):

- **Cohen's d**: The standard measure
  - |d| < 0.2: Negligible difference
  - |d| = 0.2-0.5: Small difference
  - |d| = 0.5-0.8: Medium difference
  - |d| > 0.8: Large difference
- **Hedges' g**: Corrected for small samples
- **Glass' Delta**: Uses control group SD only

Confidence Interval: Range where the true difference likely falls

- If it includes 0, groups may not differ
- Width indicates precision (narrower = more precise)

When to Use This:

Use t-test when:

- You have exactly two groups to compare
- Your outcome variable is numeric (continuous)
- Groups are independent (different people in each)
- Data is roughly normally distributed (or n >= 30 per group)

Don't use when:

- You have more than two groups (use ANOVA)
- Data is severely skewed with small samples (use Mann-Whitney)
- Groups are paired or matched (use wilcoxon_test())
- Variables are categorical (use chi-square)

Reading the Output:

The results show both variance assumptions:

- **Welch's test** (var.equal = FALSE): Safer, doesn't assume equal variances
- **Student's test** (var.equal = TRUE): Traditional, assumes equal variances

A result with t = 2.45, p = 0.015, d = 0.62 means:

- Groups are 2.45 standard errors apart (t-statistic)
- 1.5% chance this is due to random variation (p-value)
- Medium-sized practical difference (Cohen's d)

Tips for Success:

- Always check sample sizes (aim for 30+ per group)
- Consider both statistical significance AND effect size
- Use weights for population-level conclusions
- Plot your data first to check assumptions
- Report confidence intervals along with p-values

Value

Test results showing whether groups differ, including:

- t-statistic and p-value for statistical significance
- Mean difference and confidence interval (one-sample tests: mean minus `mu` and the interval of that difference, as in the SPSS One-Sample Test)
- Effect sizes (Cohen's d, Hedges' g, Glass' Delta)
- Group statistics (mean, SD, standard error of the mean, sample size)
- A `note` column that explains why a variable (or group) was not computed, e.g. no variance or only one group with valid data Use `summary()` for the full SPSS-style output with toggleable sections.

References

Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences (2nd ed.). Lawrence Erlbaum Associates.

Hedges, L. V. (1981). Distribution theory for Glass's estimator of effect size and related estimators. Journal of Educational Statistics, 6(2), 107-128.

Glass, G. V. (1976). Primary, secondary, and meta-analysis of research. Educational Researcher, 5(10), 3-8.

Welch, B. L. (1947). The generalization of "Student's" problem when several different population variances are involved. Biometrika, 34(1-2), 28-35.

See Also

`t.test` for the base R t-test function.

`print.t_test` for printing results.

`group_by` for grouped analyses.

`summary.t_test` for detailed output with toggleable sections.

Other hypothesis_tests: `ancova()`, `binomial_test()`, `chi_square()`, `chisq_gof()`, `factorial_anova()`, `fisher_test()`, `friedman_test()`, `kruskal_wallis()`, `mann_whitney()`, `mcnemar_test()`, `oneway_anova()`, `wilcoxon_test()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic two-sample test
survey_data %>%
  t_test(life_satisfaction, group = gender)

# With survey weights
survey_data %>%
  t_test(life_satisfaction, group = gender, weights = sampling_weight)
```

```
# Multiple variables
survey_data %>%
  t_test(age, income, life_satisfaction, group = region, weights = sampling_weight)

# One-sample test against a benchmark
survey_data %>%
  t_test(life_satisfaction, mu = 5, weights = sampling_weight)

# Grouped analysis
survey_data %>%
  group_by(region) %>%
  t_test(life_satisfaction, group = gender, weights = sampling_weight)

# Equal variance assumption
survey_data %>%
  t_test(life_satisfaction, group = gender, var.equal = TRUE)

# One-sided test
survey_data %>%
  t_test(income, group = gender, alternative = "greater")

# Using tidyselect helpers
survey_data %>%
  t_test(starts_with("trust"), group = gender, weights = sampling_weight)

# Store results for further analysis
result <- survey_data %>%
  t_test(life_satisfaction, group = gender, weights = sampling_weight)
print(result)

# --- Three-layer output ---
result <- t_test(survey_data, life_satisfaction, group = gender)
result              # compact one-line summary
summary(result)     # full detailed output with all sections
summary(result, effect_sizes = FALSE)  # hide effect sizes
```

`unlabel` *Remove All Label Metadata*

Description

Strips all label-related attributes from variables, converting `haven_labelled` vectors to their base R types (numeric or character). This is useful when you need plain data without any labelling for functions or packages that do not support labelled data.

Usage

```
unlabel(data, ...)
```

Arguments

data	A data frame, tibble, or a single vector.
...	Optional: unquoted variable names (tidyselect supported). If empty, applies to all columns.

Details

The following attributes are removed:

- "label" (variable label)
- "labels" (value labels)
- "na_tag_map" (tagged NA mapping)
- "na_tag_format" (tagged NA format)
- "na_values", "na_range" (SPSS missing value specs)
- "format.spss", "format.stata", "format.sas" (format info)

Tagged NAs are converted to regular NA.

Value

The data with all labelling removed. haven_labelled numeric vectors become plain double, factors remain factors (but lose their "label" attribute).

See Also

strip_tags() for removing only tagged NAs, to_label() for converting to factors (preserving label info)

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_character(), to_label(), to_labelled(), to_numeric(), val_labels(), var_label()

Examples

```
# Remove all labels from the entire dataset
data_plain <- unlabel(survey_data)

# Remove labels from specific variables only
data <- unlabel(survey_data, gender, life_satisfaction)
```

untag_na *Convert Tagged NAs Back to Original Codes*

Description

Replaces tagged NAs with their original missing value codes. Works with data imported via `read_spss()` (with `tag_na = TRUE`) or any reader that used the `tag_na` parameter (`read_stata()`, `read_sas()`, `read_xpt()`). For native Stata/SAS tagged NAs (e.g., `.a`, `.A`) that have no numeric codes to recover, use `strip_tags()` instead.

Usage

```
untag_na(x, ...)
```

Arguments

`x`	A numeric vector with tagged NAs, or a data frame.
`...`	For a data frame: the columns to convert (tidyselect). If empty, every numeric column is converted. Ignored for vectors.

Value

The input with tagged NAs that have numeric codes replaced by their original values (e.g., -9, -8, -42). System NAs (untagged) remain `NA`. Value labels and the variable label are kept (labels of missing types are attached to their codes again), so the result is `haven_labelled` when the input was. For native Stata/SAS tagged NAs (no numeric codes), falls back to `strip_tags()` behavior with a warning. A data frame is returned with the selected columns converted.

See Also

`read_spss()`, `read_stata()`, `read_sas()`, `na_frequencies()`, `strip_tags()`

Other data-import: `na_frequencies()`, `read_por()`, `read_sas()`, `read_spss()`, `read_stata()`, `read_xlsx()`, `read_xpt()`, `strip_tags()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Tag -9/-8 as missing, then recover the original codes
  x <- set_na(c(1, 2, -9, 3, -8), -9, -8)
  untag_na(x)   # -9 and -8 are back
}
```

update.linear_regression

Update and re-fit a linear_regression model

Description

Re-runs linear_regression with a modified formula or arguments, e.g. update(model, . ~ . + income); step() works through it. Needs the data by name: a model fitted inside a %>% pipe cannot be updated.

Usage

```
## S3 method for class 'linear_regression'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

object	A linear_regression result (also grouped or pairwise).
formula.	Changes to the formula (see stats::update()).
...	Further arguments of linear_regression() to change.
evaluate	If FALSE, return the updated call.

Value

A new linear_regression result (or the call).

update.logistic_regression

Update and re-fit a logistic_regression model

Description

Re-runs logistic_regression with a modified formula or arguments, e.g. update(model, . ~ . + income); step() works through it. Needs the data by name: a model fitted inside a %>% pipe cannot be updated.

Usage

```
## S3 method for class 'logistic_regression'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

object	A logistic_regression result (also grouped).
formula.	Changes to the formula (see stats::update()).
...	Further arguments of logistic_regression() to change.
evaluate	If FALSE, return the updated call.

Value

A new logistic_regression result (or the call).

---

val_labels *Get or Set Value Labels*

Description

Retrieves or assigns value labels (the "labels" attribute) on labelled vectors or data frame columns. Value labels map numeric codes to descriptive text (e.g., 1 = "Male", 2 = "Female").

The function operates in two modes:

- **GET mode**: Called with bare variable names, returns existing value labels.
- **SET mode**: Called with name = c(...) pairs, assigns value labels to variables.

Usage

```
val_labels(data, ..., .add = FALSE, drop_na = TRUE)
```

Arguments

data	A data frame, tibble, or a single vector.
...	In GET mode: unquoted variable names (tidyselect supported). In SET mode: named pairs where the name is a variable and the value is a named vector of labels (e.g., c("Male" = 1, "Female" = 2)). Use NULL to remove all value labels from a variable.
.add	If TRUE, adds labels to any existing ones instead of replacing them. Default: FALSE.
drop_na	If TRUE (default), tagged NA labels are excluded from GET results.

Details

Value labels are stored as the "labels" attribute in haven's format: a named numeric vector where names are the label text and values are the numeric codes. This is the standard used by read_spss(), read_stata(), and read_sas().

Adding vs. Replacing Labels:

By default, SET mode replaces all existing value labels. Use .add = TRUE to keep existing labels and only add new ones. If a value already has a label, the new label overwrites it.

Value labels belong to numeric codes: setting them on a factor or a character column is an error (convert it with to_labelled() first).

Value

- **GET mode (vector input)**: A named numeric vector of value labels, or NULL.
- **GET mode (data frame, single variable)**: A named numeric vector of value labels.
- **GET mode (data frame, multiple variables)**: A named list of label vectors.
- **SET mode**: The modified data frame (invisibly).

See Also

var_label() for variable labels, to_label() for converting labelled vectors to factors, drop_labels() for removing unused labels

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_character(), to_label(), to_labelled(), to_numeric(), unlabel(), var_label()

Examples

```
# SET: assign value labels to a numeric rating (1-5)
data <- val_labels(survey_data,
  life_satisfaction = c("Very dissatisfied" = 1, "Very satisfied" = 5)
)

# GET: retrieve value labels from a vector
val_labels(data$life_satisfaction)

# GET: from specific variables (returns list)
val_labels(data, life_satisfaction, trust_government)

# ADD: add labels without removing existing ones
data <- val_labels(data,
  life_satisfaction = c("Neither" = 3),
  .add = TRUE
)

# REMOVE: set to NULL
data <- val_labels(data, life_satisfaction = NULL)
```

---

var_label *Get or Set Variable Labels*

Description

Retrieves or assigns variable labels (the "label" attribute) on vectors or data frame columns. Variable labels describe what a variable measures (e.g., "Age of respondent") and are commonly used in survey data imported from SPSS, Stata, or SAS.

The function operates in two modes:

- **GET mode**: Called with bare variable names or no ... arguments, returns existing labels.
- **SET mode**: Called with name = "label" pairs, assigns labels to variables and returns the modified data.

Usage

```
var_label(data, ...)

var_label(x) <- value
```

Arguments

data	A data frame, tibble, or a single vector.
...	In GET mode: unquoted variable names (tidyselect supported). If empty, returns labels for all variables. In SET mode: named pairs where the name is a variable and the value is the label string. Use NULL to remove a label.
x	For the replacement form: a vector or a data frame.
value	For the replacement form: a single character string (or NULL to remove the label) for a vector; for a data frame a named list or named character vector, names = columns (a NULL entry removes that label). Columns not named keep their labels.

Details

Variable labels are stored as the "label" attribute on each column, which is the standard used by the haven package for SPSS/Stata/SAS imports. These labels are preserved by mariposa's codebook(), frequency(), and describe() functions.

When to Use This:

Use var_label() when you:

- Want to inspect what variables measure in imported survey data
- Need to add labels to manually created data before using codebook() or write_spss()
- Want to update or correct labels after import

Value

- **GET mode (vector input)**: A single character string (the label), or NULL if no label exists.
- **GET mode (data frame input)**: A named character vector of labels. Variables without labels return NA.
- **SET mode**: The modified data frame (invisibly), with labels assigned.

See Also

val_labels() for value labels, codebook() for viewing all metadata, copy_labels() for restoring labels lost by base-R operations

Other labels: copy_labels(), drop_labels(), find_var(), set_na(), to_character(), to_label(), to_labelled(), to_numeric(), unlabel(), val_labels()

Examples

```
# GET: retrieve all variable labels
var_label(survey_data)

# GET: specific variables
var_label(survey_data, age, gender)

# GET: from a single vector
var_label(survey_data$age)

# SET: assign labels
data <- var_label(survey_data,
  age = "Age of respondent",
  gender = "Gender identity"
)

# REMOVE: set to NULL
data <- var_label(data, age = NULL)

# Replacement form
x <- c(1, 2, 3)
var_label(x) <- "Score"
d <- survey_data
var_label(d) <- list(age = "Age of respondent", gender = "Gender identity")
```

---

vcov.linear_regression

Variance-covariance matrix of a linear_regression model

Description

Unweighted models: `stats::vcov()` of the `lm`. Weighted models: the covariance matrix under SPSS frequency weights (residual variance with `sum(w) - rank` df), whose square-rooted diagonal equals the Std.Error column of `summary()`.

Usage

```
## S3 method for class 'linear_regression'
vcov(object, ...)
```

Arguments

`object`	A `linear_regression` result (ungrouped, listwise).
`...`	Passed to `stats::vcov()`.

Value

A square numeric matrix.

`wilcoxon_test` *Compare Two Related Measurements Without Assuming Normality*

Description

`wilcoxon_test()` compares two paired measurements from the same subjects when your data isn't normally distributed. It's the non-parametric alternative to the paired t-test.

Think of it as:

- A way to test whether scores changed between two time points
- Comparing ratings of two items from the same respondents
- A robust paired comparison that works with any data shape

The test tells you:

- Whether scores are significantly different between the two measurements
- How many subjects increased, decreased, or stayed the same
- The strength of the effect (effect size r)

Usage

```
wilcoxon_test(data, x, y, weights = NULL, conf.level = 0.95)
```

Arguments

`data`	Your survey data (a data frame or tibble) in wide format, with one row per subject and the two measurements in separate columns
`x`	The first measurement variable (e.g., pre-test, trust in government)
`y`	The second measurement variable (e.g., post-test, trust in media). The difference is computed as `y - x`
`weights`	Optional survey weights for population-representative results. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`conf.level`	Not used. Rank tests report no confidence interval (SPSS `NPAR TESTS` neither); kept for backward compatibility.

Details

Understanding the Results:

P-value: If $p < 0.05$, the two measurements are significantly different

- $p < 0.001$: Very strong evidence of a difference
- $p < 0.01$: Strong evidence of a difference
- $p < 0.05$: Moderate evidence of a difference
- $p > 0.05$: No significant difference found

Effect Size r (How strong is the difference?):

- < 0.1: Negligible effect
- 0.1 - 0.3: Small effect
- 0.3 - 0.5: Medium effect
- 0.5 or higher: Large effect

Rank Categories:

- Negative Ranks: subjects where y < x (decreased)
- Positive Ranks: subjects where y > x (increased)
- Ties: subjects where y = x (no change)

Z: as in SPSS, computed from the smaller of the two rank sums, so it is never positive; `z_based_on` ("negative ranks" or "positive ranks") names that sum, like SPSS's footnote. "Based on negative ranks" means that increases (y > x) dominate.

When to Use This:

Use Wilcoxon signed-rank test when:

- Comparing two related measurements from the same subjects
- Your data is not normally distributed
- You have ordinal data (ratings, rankings)
- Sample size is small
- You want a robust alternative to the paired t-test

Relationship to Other Tests:

- For normally distributed paired data: Use paired t-test instead
- For independent groups: Use `mann_whitney()` instead

Weighted analyses:

Weights are treated as frequency weights. With all weights equal to 1 the weighted test reproduces the unweighted one exactly. Weighted results are not compared with SPSS output; see `vignette("spss-compatibility")`.

- For 3+ related measurements: Use `friedman_test()` instead

Value

Test results showing whether the two measurements differ, including:

- Z statistic (standardized test statistic, normal approximation)
- P-value (is there a significant difference?)
- Effect size r (how strong is the difference?)
- Rank statistics (negative ranks, positive ranks, ties)

References

Wilcoxon, F. (1945). Individual comparisons by ranking methods. Biometrics Bulletin, 1(6), 80-83.

Fritz, C. O., Morris, P. E., & Richler, J. J. (2012). Effect size estimates: current use, calculations, and interpretation. Journal of Experimental Psychology: General, 141(1), 2.

See Also

wilcox.test for the base R Wilcoxon test.

mann_whitney for comparing two independent groups.

Other hypothesis_tests: ancova(), binomial_test(), chi_square(), chisq_gof(), factorial_anova(), fisher_test(), friedman_test(), kruskal_wallis(), mann_whitney(), mcnemar_test(), oneway_anova(), t_test()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Compare trust in government vs trust in media
survey_data %>%
  wilcoxon_test(x = trust_government, y = trust_media)

# Weighted analysis
survey_data %>%
  wilcoxon_test(x = trust_government, y = trust_media,
                weights = sampling_weight)

# Grouped analysis (separate test per region)
survey_data %>%
  group_by(region) %>%
  wilcoxon_test(x = trust_government, y = trust_media)
```

---

write_spss *Export Data to SPSS Format*

Description

Writes a data frame to an SPSS .sav file, preserving variable labels, value labels, and user-defined missing values. When exporting data that was imported with read_spss() (with tag_na = TRUE), the tagged NAs are automatically converted back to SPSS user-defined missing values, enabling full roundtrip fidelity.

Usage

```
write_spss(data, path, compress = c("byte", "none", "zsav"))
```

Arguments

data	A data frame to export. Columns of class haven_labelled will have their labels and missing value metadata written to the .sav file.
path	Path to the output file. Must end in .sav or .zsav.
compress	Compression type. One of "byte" (default, byte-level compression), "none" (no compression), or "zsav" (zlib compression, requires SPSS v21+).

Details

Tagged NA Roundtripping:

Data imported via `read_spss()` stores SPSS user-defined missing values as tagged NAs with an `na_tag_map` attribute mapping tag characters to original codes (e.g., -9, -8). `write_spss()` reverses this process: tagged NAs are converted back to their original numeric codes, and the SPSS user-defined missing value specification is reconstructed so that the exported `.sav` file has the same missing value definitions as the original: `read_spss()` remembers the original definition (e.g. `LOWEST THRU -1`), which is written back as long as it still covers every missing code and no valid value. Otherwise up to 3 codes are written as discrete values; more codes use SPSS's "range plus one discrete value" form (the tightest that contains no valid value), announced in one message. If no such range exists, the export stops with an error instead of declaring valid values missing.

Cross-Format Export:

When exporting data originally imported from Stata or SAS (with native extended missing values like `.a-.z` or `.A-.Z`), these cannot be represented as SPSS user-defined missing values. In this case, they are written as system missing (regular `NA`) with a warning.

Derived Variables:

Arithmetic on an imported variable (e.g. `x + 1` inside `mutate()`) keeps its tagged NAs but loses the code map, so the original codes are unknown. Such values are written as system missing with a warning naming the variables. `rec()` keeps the map; `std()`, `center()`, `pomps()` and `to_numeric()` return plain `NA` by design. Numeric columns that carry a bare `"labels"` attribute (without the `haven_labelled` class) are exported with their value labels.

When to Use This:

Use `write_spss()` when you:

- Need to export processed survey data back to SPSS format
- Want to preserve user-defined missing value definitions
- Need roundtrip fidelity: `read_spss()` -> processing -> `write_spss()`

Value

Invisibly returns the file path.

See Also

`read_spss()` for importing SPSS files, `write_xlsx()` for Excel export, `write_stata()` for Stata export, `write_xpt()` for SAS transport export, `untag_na()`, `strip_tags()`

Other data-export: `write_stata()`, `write_xlsx()`, `write_xpt()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip: write to a temporary .sav, read back
  tmp <- tempfile(fileext = ".sav")
  write_spss(survey_data, tmp)
  data <- read_spss(tmp)
```

```
  # Export with zlib compression (smaller file, requires SPSS v21+)
  tmp_z <- tempfile(fileext = ".zsav")
  write_spss(survey_data, tmp_z, compress = "zsav")

  unlink(c(tmp, tmp_z))
}
```

`write_stata` *Export Data to Stata Format*

Description

Writes a data frame to a Stata `.dta` file, preserving variable labels, value labels, and missing value types. Tagged NAs from any source format (SPSS, Stata, SAS) are written as Stata extended missing values (`.a` through `.z`).

Usage

```
write_stata(data, path, version = 14)
```

Arguments

`data`	A data frame to export. Columns of class `haven_labelled` will have their labels written to the `.dta` file.
`path`	Path to the output file. Must end in `.dta`.
`version`	Stata file version to use. Supported values: 8-15. Default is 14 (Stata 14/15, supports Unicode). Use 13 for compatibility with older Stata versions.

Details

Tagged NA Handling:

Stata natively supports 27 extended missing values: `.a` through `.z` and `.` (system missing). When exporting data with tagged NAs:

- **From Stata** (`read_stata()`): Native extended missing values are preserved in a full roundtrip.
- **From SPSS** (`read_spss()`): Tagged NAs are written as Stata extended missing values (`.a`, `.b`, etc.). The original SPSS numeric codes (e.g., -9, -8) are not preserved in the Stata file, but the distinct missing value types and their labels are.
- **From SAS** (`read_sas()`, `read_xpt()`): SAS special missing values are mapped to their Stata equivalents.

Limitations:

Stata supports at most 27 distinct missing types per variable. SPSS data with more than 27 tagged NA types in a single variable may lose some distinctions (though this is extremely rare in practice).

Value

Invisibly returns the file path.

See Also

`read_stata()` for importing Stata files, `write_spss()` for SPSS export, `write_xlsx()` for Excel export, `write_xpt()` for SAS transport export

Other data-export: `write_spss()`, `write_xlsx()`, `write_xpt()`

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip: write to a temporary .dta, read back
  tmp <- tempfile(fileext = ".dta")
  write_stata(survey_data, tmp)
  data <- read_stata(tmp)

  # Stata 13 compatibility
  tmp13 <- tempfile(fileext = ".dta")
  write_stata(survey_data, tmp13, version = 13)

  unlink(c(tmp, tmp13))
}
```

---

write_xlsx *Export Data to Excel with Label Support*

Description

Writes data to an Excel (`.xlsx`) file with support for variable labels, value labels, and tagged NA metadata. When exporting labelled survey data, a "Labels" reference sheet is automatically included so that reviewers can look up what each numeric code means – even without R access.

Four input types are supported:

1. **Data frame**: Exports the data plus a "Labels" reference sheet with variable labels, value labels, and missing value codes.
2. **Codebook object**: Exports the codebook as it appears in the HTML viewer – with values, value labels, and frequencies stacked inside each cell, tagged NAs separated by a line, and an overview sheet with dataset metadata.
3. **Analysis results**: `frequency()` and `crosstab()` tables in their printed layout; every other result (e.g. `describe()`, `t_test()`, `oneway_anova()`, `reliability()`, `linear_regression()`) as its result tables.
4. **Named list**: Each list element becomes a separate sheet. Elements can be data frames, `codebook()` results, or any analysis result – types can be mixed freely. For data frame elements, a combined "Labels" sheet is appended.

Usage

```
write_xlsx(x, file, ...)

## S3 method for class 'frequency'
write_xlsx(x, file, overwrite = TRUE, ...)

## Default S3 method:
write_xlsx(x, file, overwrite = TRUE, ...)
```

Arguments

x	Object to export: a data frame, a codebook() result, an analysis result, or a named list of these.
file	Path to the output .xlsx file. Must end in .xlsx.
...	Additional arguments passed to methods: labels (data.frame and list methods) Include a "Labels" reference sheet? Default: TRUE frequencies (codebook method) Include per-variable frequency sheets? Default: FALSE
overwrite	Overwrite existing file? Default: TRUE.

Details

Data Frame Export:

The "Labels" sheet provides a complete lookup table for all variable and value labels in your data, structured in long format. The "Type" column distinguishes between regular value labels ("valid") and tagged missing value labels ("missing"), making it easy to filter in Excel.

Data values are written as their underlying numeric or character codes (not as label text), preserving the original coding scheme.

Codebook Export:

The codebook export reproduces the HTML codebook layout: each variable occupies one row, with values, value labels, and frequencies stacked within their cells using line breaks. Tagged NAs are shown below a separator line, matching the visual style of codebook() in the RStudio Viewer.

When to Use This:

Use write_xlsx() when you:

- Need to share survey data with non-R users who need label context
- Want to export a codebook for manual review or documentation
- Need to combine multiple tables (data, codebook, frequencies) in one Excel file
- Want to hand analysis results (descriptives, crosstabs, test results) to colleagues as Excel tables

Value

Invisibly returns the file path.

Methods (by class)

- `write_xlsx(frequency)`: Export frequency tables to Excel

  Exports one or more frequency tables to a single Excel sheet, mirroring the console print output. Each variable gets a header row, stats summary, column headers, data rows, and total rows. Multiple variables are separated by 3 blank rows.

- `write_xlsx(default)`: Export any other mariposa analysis result (e.g. `describe()`, `crosstab()`, `t_test()`, `reliability()`, `linear_regression()`) to one sheet. `crosstab()` results are written in the SPSS table layout (counts and the requested percentages per row category, with totals); all other results as the table as.data.frame() returns, followed by the secondary tables SPSS shows next to it (group descriptives, mean ranks, item statistics, model summary, ...). Grouped results keep the grouping variables as leading columns (crosstabs get one block per group).

See Also

`codebook()` for generating codebook objects, `frequency()` for frequency tables, as.data.frame() for analysis results as data frames, `read_spss()`, `read_por()`, `read_stata()`, `read_sas()`, `read_xpt()`, `read_xlsx()` for importing labelled data

Other data-export: `write_spss()`, `write_stata()`, `write_xpt()`

Examples

```
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  # Export data with automatic label reference sheet
  tmp <- tempfile(fileext = ".xlsx")
  write_xlsx(survey_data, tmp)

  # Export a codebook (reproduces HTML layout in Excel)
  tmp_cb <- tempfile(fileext = ".xlsx")
  codebook(survey_data) |> write_xlsx(tmp_cb)

  # Export frequency tables
  tmp_freq <- tempfile(fileext = ".xlsx")
  frequency(survey_data, gender) |> write_xlsx(tmp_freq)

  # Multi-sheet export (mixed types: data frames, frequencies, codebooks)
  tmp_multi <- tempfile(fileext = ".xlsx")
  write_xlsx(
    list(
      "Frequencies" = frequency(survey_data, gender, life_satisfaction),
      "Codebook"    = codebook(survey_data),
      "Data"        = survey_data
    ),
    tmp_multi
  )

  unlink(c(tmp, tmp_cb, tmp_freq, tmp_multi))
}
```

```
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  # Single variable
  tmp <- tempfile(fileext = ".xlsx")
  frequency(survey_data, gender) |> write_xlsx(tmp)

  # Multiple variables on one sheet
  tmp2 <- tempfile(fileext = ".xlsx")
  frequency(survey_data, gender, life_satisfaction, region) |>
    write_xlsx(tmp2)

  unlink(c(tmp, tmp2))
}


if (requireNamespace("openxlsx2", quietly = TRUE)) {
  # Analysis results: one sheet each, or several in one workbook
  tmp_res <- tempfile(fileext = ".xlsx")
  t_test(survey_data, age, income, group = gender) |> write_xlsx(tmp_res)

  tmp_list <- tempfile(fileext = ".xlsx")
  write_xlsx(
    list(
      "Descriptives" = describe(survey_data, age, income),
      "Crosstab"     = crosstab(survey_data, gender, region),
      "ANOVA"        = oneway_anova(survey_data, age, group = education)
    ),
    tmp_list
  )

  unlink(c(tmp_res, tmp_list))
}
```

`write_xpt` *Export Data to SAS Transport Format*

Description

Writes a data frame to a SAS transport file (`.xpt`), preserving variable labels and missing value types. Tagged NAs from any source format (SPSS, Stata, SAS) are written as SAS special missing values (`.A` through `.Z`, `._`).

Usage

```
write_xpt(data, path, version = 5, name = NULL)
```

Arguments

data	A data frame to export.
path	Path to the output file. Must end in .xpt.
version	SAS transport file version. Either 5 (default, SAS Transport v5, most compatible, variable names of up to 8 characters) or 8 (SAS Transport v8, names of up to 32 characters). Longer names are truncated with a warning; if truncation would create duplicate names, the export stops.
name	Member name for the dataset within the transport file. If NULL, derived from the file name. Maximum 8 characters for version 5.

Details

Tagged NA Handling:

SAS supports 28 special missing values: . (system missing), .A through .Z, and ._. When exporting data with tagged NAs:

- **From SAS** (read_sas(), read_xpt()): Native special missing values are preserved in a full roundtrip.
- **From SPSS** (read_spss()): Tagged NAs are written as SAS special missing values. The original SPSS numeric codes (e.g., -9, -8) are not preserved, but the distinct missing value types are.
- **From Stata** (read_stata()): Stata extended missing values are mapped to their SAS equivalents.

Limitations:

SAS transport files do **not** store value labels. If your data has value labels, consider using write_spss() or write_xlsx() instead, which preserve full label metadata. Variable labels are preserved.

Value

Invisibly returns the file path.

See Also

read_xpt() and read_sas() for importing SAS files, write_spss() for SPSS export, write_stata() for Stata export, write_xlsx() for Excel export

Other data-export: write_spss(), write_stata(), write_xlsx()

Examples

```
if (requireNamespace("haven", quietly = TRUE)) {
  # Roundtrip: write to a temporary .xpt transport file, read back
  tmp <- tempfile(fileext = ".xpt")
  write_xpt(survey_data, tmp, version = 8)  # names longer than 8 chars
  data <- read_xpt(tmp)

  unlink(tmp)
```

```
}
```

`w_iqr` *Measure Population-Representative Spread (IQR)*

Description

`w_iqr()` calculates the interquartile range using survey weights. The IQR is the distance between the 25th and 75th percentiles – it tells you the range that contains the middle 50% of your population. Unlike the standard deviation, the IQR is not affected by outliers, making it a robust measure of spread.

Usage

```
w_iqr(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("income")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample IQR. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

Understanding the Results:

- **Weighted IQR**: The range that covers the middle 50% of the weighted population. A larger IQR means more spread in the central part of the data.
- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

The IQR is especially useful when your data is skewed. For example, with income data, the IQR gives a better sense of "typical spread" than the SD because extreme incomes do not distort it.

When to Use This:

Use `w_iqr()` when:

- Your data has outliers or is skewed (e.g., income, response times)
- You want a robust measure of spread that is not influenced by extremes

- You need to describe the spread of the middle 50% of your population
- You need SPSS-compatible weighted IQR values

Formula:

$IQR_w = Q_{3,w} - Q_{1,w}$

where $Q_{1,w}$ and $Q_{3,w}$ are the weighted 25th and 75th percentiles, calculated using cumulative weights (see w_quantile).

Value

Population-weighted IQR(s) with sample size information, including the weighted IQR, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

IQR for the base R IQR function.

w_quantile for arbitrary weighted percentiles.

w_sd for weighted standard deviation (another spread measure).

w_range for the full weighted range.

describe for comprehensive descriptive statistics including IQR.

Method:

Unweighted values use quantile Type 6 (SPSS HAVERAGE). Weighted values apply the HAVERAGE position to the cumulative weights with linear interpolation. Both reproduce SPSS FREQUENCIES (with and without WEIGHT BY); see vignette("spss-compatibility").

Other weighted_statistics: w_kurtosis(), w_mean(), w_median(), w_modus(), w_quantile(), w_range(), w_sd(), w_se(), w_skew(), w_var()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted IQR
survey_data %>% w_iqr(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_iqr(age, income, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_iqr(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(iqr_age = w_iqr(age, weights = sampling_weight))
```

```
# Unweighted (for comparison)
survey_data %>% w_iqr(age)
```

`w_kurtosis` *Measure Population-Representative Kurtosis*

Description

`w_kurtosis()` measures how heavy the tails of your data's distribution are, using survey weights for population-representative results. Kurtosis tells you whether your data has unusually many extreme values (outliers) compared to a normal distribution:

- **Positive (excess) kurtosis**: Heavier tails than normal (more outliers)
- **Negative (excess) kurtosis**: Lighter tails than normal (fewer outliers)
- **Near zero**: Similar tail behavior to a normal distribution

Usage

```
w_kurtosis(data, ..., weights = NULL, na.rm = TRUE, excess = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("trust")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample kurtosis. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).
`excess`	Show excess kurtosis? (Default: TRUE, matching SPSS). Excess kurtosis subtracts 3 so that a normal distribution has kurtosis of 0, making interpretation easier.

Details

Understanding the Results:

- **Weighted Kurtosis** (excess, default):
  - Near 0: Tail behavior similar to a normal distribution
  - Positive (> 0): Heavier tails, more extreme values than normal
  - Negative (< 0): Lighter tails, fewer extreme values than normal
  - Values beyond 2 or below -2 indicate notable departure from normality

- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

Kurtosis is often checked together with skewness to assess normality. Both should be close to zero for normally distributed data.

When to Use This:

Use `w_kurtosis()` when:

- You need to assess whether a variable follows a normal distribution
- You want to check for unusually many outliers
- You are deciding whether parametric tests are appropriate
- You need SPSS-compatible weighted kurtosis values

Formula:

Uses the SPSS Type 2 (sample-corrected) excess kurtosis formula. First, the population excess kurtosis (g_2) is calculated:

$g_2 = \frac{m_4}{m_2^2} - 3$

where $m_2 = \sum w_i(x_i - \bar{x}_w)^2/V_1$ and $m_4 = \sum w_i(x_i - \bar{x}_w)^4/V_1$ with $V_1 = \sum w_i$.

Then, the bias-corrected (Type 2) kurtosis is:

$G_2 = \frac{(n+1)\cdot g_2+6}{(n-2)(n-3)} \cdot (n-1)$

where $n = V_1$ for weighted data.

Value

Population-weighted kurtosis value(s) with sample size information, including the weighted kurtosis, effective sample size (effective N), and the number of valid observations used.

References

Joanes, D. N., & Gill, C. A. (1998). Comparing measures of sample skewness and kurtosis. The Statistician, 47(1), 183-189.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`w_skew` for weighted skewness (distribution asymmetry).

`describe` for comprehensive descriptive statistics including kurtosis.

Other weighted_statistics: `w_iqr()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)
```

```
# Basic weighted kurtosis (excess kurtosis, default)
survey_data %>% w_kurtosis(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_kurtosis(age, income, life_satisfaction, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_kurtosis(age, weights = sampling_weight)

# Raw kurtosis (not excess)
survey_data %>% w_kurtosis(age, weights = sampling_weight, excess = FALSE)

# In summarise context
survey_data %>% summarise(kurt_age = w_kurtosis(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_kurtosis(age)
```

`w_mean` *Calculate Population-Representative Averages*

Description

`w_mean()` calculates averages that accurately represent your population by using survey weights. This ensures that groups who were over- or under-sampled contribute appropriately to the final average.

Usage

```
w_mean(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to average. You can list multiple variables or use helpers like `starts_with("income")`
`weights`	Survey weights to make the average representative of your population. Without weights, you get the simple sample average. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

When to Use This:

Use `w_mean()` when your survey uses sampling weights and you need population-representative averages. Weights correct for:

- Oversampling of certain groups (weights < 1)
- Undersampling of other groups (weights > 1)
- Non-response patterns
- Complex survey designs

Understanding the Results:

- **Weighted Mean**: The population-representative average
- **Effective N**: How many independent observations your weighted data represents
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

Formula:

$\bar{x}_w = \frac{\sum w_i \cdot x_i}{\sum w_i}$

The effective sample size is: $n_{eff} = (\sum w_i)^2 / \sum w_i^2$

Value

Population-weighted average(s) with sample size information

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`weighted.mean` for the base R weighted mean function.

`w_sd` for weighted standard deviation.

`w_median` for weighted median.

`describe` for comprehensive descriptive statistics.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted usage
survey_data %>% w_mean(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_mean(age, income, life_satisfaction, weights = sampling_weight)
```

```
# Grouped data
survey_data %>% group_by(region) %>% w_mean(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(mean_age = w_mean(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_mean(age)
```

`w_median` *Find the Population-Representative Middle Value*

Description

`w_median()` finds the median (middle value) of your data using survey weights. The weighted median is the value where half the population falls below and half falls above. Unlike the mean, the median is not pulled by extreme values, making it a robust measure of the "typical" value in your population.

Usage

```
w_median(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("income")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample median. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

Understanding the Results:

- **Weighted Median**: The value that splits the weighted population in half. 50% of the population (by weight) falls below this value, 50% above.
- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

Comparing the weighted median to the weighted mean is informative:

- If they are similar, the distribution is roughly symmetric.
- If the mean is much larger than the median, the distribution is right-skewed (a few very high values pull the mean up, e.g., income).

When to Use This:

Use `w_median()` when:

- Your data has outliers or is skewed (e.g., income, housing prices)
- You want a robust "typical value" not influenced by extremes
- You need the weighted 50th percentile
- You need SPSS-compatible weighted median values

Formula:

The weighted median is the weighted 50th percentile, computed with the SPSS HAVERAGE (quantile Type 6) position h = 0.5 * (W + 1) on the cumulative weights, with linear interpolation between bracketing observations. It is identical to `w_quantile(x, probs = 0.5)` by construction. Unweighted and weighted values reproduce SPSS FREQUENCIES (with and without `WEIGHT BY`).

Value

Population-weighted median(s) with sample size information, including the weighted median, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`median` for the base R median function.

`w_mean` for weighted means.

`w_quantile` for arbitrary weighted percentiles.

`describe` for comprehensive descriptive statistics including the median.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted median
survey_data %>% w_median(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_median(age, income, weights = sampling_weight)
```

```
# Grouped data
survey_data %>% group_by(region) %>% w_median(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(med_age = w_median(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_median(age)
```

---

w_modus *Find the Most Common Value in Your Population*

Description

w_modus() finds the mode (most frequently occurring value) of your data using survey weights for population-representative results. The mode tells you which category or value is the most common in your population. This is especially useful for categorical variables (e.g., the most common education level, the most frequent employment status).

Unlike mean and median, the mode works with both numeric and categorical data.

Usage

```
w_modus(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The variables you want to analyze. Works best with categorical or discrete numeric variables. You can list multiple variables or use helpers like starts_with("trust")
weights	Survey weights to make results representative of your population. Without weights, the mode is simply the most frequent value in your sample. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
na.rm	Remove missing values before calculating? (Default: TRUE). With FALSE, the result for a variable that contains missing values is NA (as in base R).

Details

Understanding the Results:

- **Weighted Mode**: The value that occurs most frequently in the weighted population. For weighted data, the mode is the value whose observations have the largest total weight.
- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under WEIGHT BY; Kish's effective N is shown by summary().

If multiple values share the highest (weighted) frequency, the smallest value (for factors: the first level) is returned, as in SPSS; the print flags such results ("Multiple modes exist") and `$results$n_modes` holds the number of tied values.

When to Use This:

Use `w_modus()` when:

- You want to find the most common response (e.g., most popular education level)
- You are working with categorical or ordinal data
- You want the "typical" value for discrete data where mean is not meaningful
- You need SPSS-compatible weighted mode values

Formula:

The weighted mode is the value x_k that maximizes the total weight:

$\text{Mode}_w = \arg\max_{x_k} \sum_{i:x_i=x_k} w_i$

In other words, sum the weights for each unique value and pick the value with the largest total weight.

Value

Population-weighted mode(s) with sample size information, including the most common value (by weighted frequency), effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`w_median` for the weighted middle value.

`w_mean` for weighted means.

`frequency` for complete frequency tables of categorical variables.

`describe` for comprehensive descriptive statistics including the mode.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted mode (most frequent value)
survey_data %>% w_modus(gender, weights = sampling_weight)

# Multiple variables (works best with categorical/discrete data)
survey_data %>% w_modus(gender, region, weights = sampling_weight)

# Grouped data
```

```
survey_data %>% group_by(region) %>% w_modus(gender, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(mode_gender = w_modus(gender, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_modus(gender)
```

`w_quantile` *Calculate Population-Representative Percentiles*

Description

`w_quantile()` calculates percentiles (quantiles) using survey weights for population-representative results. Percentiles divide your data into equal portions – for example, the 25th percentile is the value below which 25% of your population falls. This is essential for understanding the distribution of variables like income, age, or satisfaction scores.

Usage

```
w_quantile(
  data,
  ...,
  weights = NULL,
  probs = c(0, 0.25, 0.5, 0.75, 1),
  na.rm = TRUE
)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("income")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample quantiles. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`probs`	Which percentiles to calculate, as proportions between 0 and 1. Default: `c(0, 0.25, 0.5, 0.75, 1)` for the minimum, 25th percentile, median, 75th percentile, and maximum. Use `c(0.1, 0.5, 0.9)` for deciles, or `c(0.25, 0.5, 0.75)` for quartiles only.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

Understanding the Results:

- **Percentile values**: The data values at each requested percentile in the weighted population. For example, if the weighted 25th percentile of income is 35,000, then 25% of the population earns less than that.
- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

Common percentiles and their meaning:

- **0% (minimum)**: The smallest observed value
- **25% (Q1)**: One quarter of the population falls below this value
- **50% (median)**: Half the population falls below this value
- **75% (Q3)**: Three quarters of the population falls below this value
- **100% (maximum)**: The largest observed value

When to Use This:

Use `w_quantile()` when:

- You want to know at what values the population splits into groups
- You need to construct population-representative income brackets or age groups
- You want to identify the median or quartiles with proper weighting
- You need SPSS-compatible weighted percentile values

Formula:

Weighted quantiles are calculated using cumulative weights. Observations are sorted by value, weights are accumulated, and the requested percentile is found by linear interpolation at the point where the cumulative weight proportion reaches the target probability.

Value

Population-weighted quantile(s) with sample size information, including the weighted percentile values, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`quantile` for the base R quantile function.

`w_median` for the weighted median (50th percentile).

`w_iqr` for the weighted interquartile range (Q3 - Q1).

`describe` for comprehensive descriptive statistics including quantiles.

Method:

Unweighted values use quantile Type 6 (SPSS HAVERAGE). Weighted values apply the HAVERAGE position to the cumulative weights with linear interpolation. Both reproduce SPSS FREQUENCIES (with and without `WEIGHT BY`); see `vignette("spss-compatibility")`.

Other weighted_statistics: w_iqr(), w_kurtosis(), w_mean(), w_median(), w_modus(), w_range(), w_sd(), w_se(), w_skew(), w_var()

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted quantiles (0%, 25%, 50%, 75%, 100%)
survey_data %>% w_quantile(age, weights = sampling_weight)

# Custom quantiles
survey_data %>% w_quantile(income, weights = sampling_weight, probs = c(0.1, 0.5, 0.9))

# Multiple variables
survey_data %>% w_quantile(age, income, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_quantile(age, weights = sampling_weight)

# Unweighted (for comparison)
survey_data %>% w_quantile(age)
```

---

w_range *Find the Range of Your Data*

Description

w_range() calculates the range (maximum minus minimum) of your data. The range gives you the total spread from the smallest to the largest observed value. It provides the w_* interface for consistency with other weighted statistics, though the range itself is not affected by weights (the minimum and maximum values remain the same regardless of weighting).

Usage

```
w_range(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The numeric variables you want to analyze. You can list multiple variables or use helpers like starts_with("income")
weights	Survey weights are accepted for interface consistency, but do not affect the range calculation. The range depends only on the observed minimum and maximum values.
na.rm	Remove missing values before calculating? (Default: TRUE). With FALSE, the result for a variable that contains missing values is NA (as in base R).

Details

Understanding the Results:

- **Range**: The difference between the largest and smallest values. A large range indicates that at least some values are far apart.
- **Effective N**: Reported when weights are provided, for consistency with other weighted statistics.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

Note: The range is sensitive to outliers. A single extreme value can dramatically increase the range. Consider using `w_iqr` for a more robust measure of spread.

When to Use This:

Use `w_range()` when:

- You want a quick overview of the total spread of your data
- You need to check for data entry errors (impossible values)
- You want to compare the total spread across groups

Formula:

$Range = \max(x) - \min(x)$

Note: This statistic is weight-invariant. The minimum and maximum observed values do not change when weights are applied.

Value

The range (max - min) with sample size information, including the effective sample size (effective N) when weights are provided, and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`range` for the base R range function.

`w_iqr` for the weighted interquartile range (more robust).

`w_sd` for weighted standard deviation (another spread measure).

`describe` for comprehensive descriptive statistics including range.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_sd()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)
```

```
# Basic range
survey_data %>% w_range(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_range(age, income, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_range(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(range_age = w_range(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_range(age)
```

---

w_sd *Calculate Population-Representative Standard Deviations*

Description

w_sd() calculates standard deviations that accurately represent your population by using survey weights. The standard deviation tells you how spread out your data is around the average – a larger SD means more variation in responses, while a smaller SD means responses cluster tightly around the mean.

Without weights, you describe spread in your sample only. With weights, you estimate how spread out values are in the entire population.

Usage

```
w_sd(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The numeric variables you want to analyze. You can list multiple variables or use helpers like starts_with("trust")
weights	Survey weights to make results representative of your population. Without weights, you get the simple sample standard deviation. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
na.rm	Remove missing values before calculating? (Default: TRUE). With FALSE, the result for a variable that contains missing values is NA (as in base R).

Details

Understanding the Results:

- **Weighted SD**: The population-representative standard deviation. Roughly 68% of your population falls within one SD of the weighted mean.
- **Effective N**: How many independent observations your weighted data represents. Always less than or equal to the actual sample size.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

A large difference between weighted and unweighted SD suggests that the variability in your sample does not accurately reflect the population.

When to Use This:

Use `w_sd()` when:

- You need to report how spread out a variable is in the population
- You want to compare variability across groups with proper weighting
- Your survey used complex sampling (oversampling, stratification)
- You need SPSS-compatible weighted standard deviations

Formula:

The weighted standard deviation uses the SPSS frequency weights formula:

$s_w = \sqrt{\frac{\sum w_i(x_i - \bar{x}_w)^2}{V_1 - 1}}$

where $V_1 = \sum w_i$ is the sum of all weights and $\bar{x}_w = \sum w_i x_i / V_1$ is the weighted mean.

The effective sample size is: $n_{eff} = (\sum w_i)^2 / \sum w_i^2$

Value

Population-weighted standard deviation(s) with sample size information, including the weighted SD, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`sd` for the base R standard deviation function.

`w_var` for weighted variance (the square of weighted SD).

`w_mean` for weighted means.

`describe` for comprehensive descriptive statistics including SD.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_se()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted standard deviation
survey_data %>% w_sd(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_sd(age, income, life_satisfaction, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_sd(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(sd_age = w_sd(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_sd(age)
```

---

w_se *Calculate Population-Representative Standard Errors*

Description

`w_se()` calculates the standard error of the mean using survey weights. The standard error tells you how precisely you have estimated the population mean – a smaller SE means your estimate is more precise. This is essential for constructing confidence intervals and assessing the reliability of your weighted mean estimates.

Usage

```
w_se(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("trust")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample standard error. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

Understanding the Results:

- **Weighted SE**: The precision of your weighted mean estimate. Smaller values mean more precise estimates. You can build a 95% confidence interval as: weighted mean +/- 1.96 * weighted SE.
- **Effective N**: How many independent observations your weighted data represents. Weights that vary a lot reduce effective N, increasing the SE.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

When to Use This:

Use `w_se()` when:

- You need to report precision of mean estimates
- You want to construct confidence intervals for weighted means
- You need to compare precision across subgroups
- You need SPSS-compatible weighted standard errors

Formula:

The weighted standard error is calculated as:

$SE_w = \frac{s_w}{\sqrt{V_1}}$

where s_w is the weighted standard deviation (see `w_sd`) and $V_1 = \sum w_i$ is the sum of all weights.

For the unweighted case: $SE = s/\sqrt{n}$

Value

Population-weighted standard error(s) with sample size information, including the weighted SE, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`w_sd` for weighted standard deviation.

`w_mean` for weighted means.

`describe` for comprehensive descriptive statistics including SE.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_skew()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)
```

```
# Basic weighted standard error
survey_data %>% w_se(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_se(age, income, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_se(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(se_age = w_se(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_se(age)
```

---

w_skew *Measure Population-Representative Skewness*

Description

w_skew() measures how asymmetric your data's distribution is, using survey weights for population-representative results. Skewness tells you whether values tend to trail off more to the left or right:

- **Positive skew**: A long tail to the right (e.g., income – many low, few very high)
- **Negative skew**: A long tail to the left (e.g., exam scores – many high, few very low)
- **Near zero**: Roughly symmetric (e.g., height in a population)

Usage

```
w_skew(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

data	Your survey data (a data frame or tibble)
...	The numeric variables you want to analyze. You can list multiple variables or use helpers like starts_with("trust")
weights	Survey weights to make results representative of your population. Without weights, you get the simple sample skewness. Give a column name (unquoted or as a string), an expression such as sampling_weight * 2, or a numeric vector with one weight per row.
na.rm	Remove missing values before calculating? (Default: TRUE). With FALSE, the result for a variable that contains missing values is NA (as in base R).

Details

Understanding the Results:

- **Weighted Skewness**: The population-representative skewness value.
  - Near 0: Distribution is roughly symmetric
  - Between -0.5 and 0.5: Approximately symmetric
  - Between -1 and -0.5 or 0.5 and 1: Moderately skewed
  - Beyond -1 or 1: Highly skewed
- **Effective N**: How many independent observations your weighted data represents.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

High skewness suggests you might want to use the median instead of the mean as a measure of center, and non-parametric tests instead of t-tests.

When to Use This:

Use `w_skew()` when:

- You need to check if a variable is normally distributed
- You want to decide between parametric and non-parametric tests
- You need to assess whether the mean is a good summary measure
- You need SPSS-compatible weighted skewness values

Formula:

Uses the SPSS Type 2 (sample-corrected) skewness formula. First, the population skewness (g_1) is calculated:

$g_1 = \frac{m_3}{m_2^{3/2}}$

where $m_2 = \sum w_i(x_i - \bar{x}_w)^2/V_1$ and $m_3 = \sum w_i(x_i - \bar{x}_w)^3/V_1$ with $V_1 = \sum w_i$.

Then, the bias-corrected (Type 2) skewness is:

$G_1 = g_1 \cdot \frac{\sqrt{n(n-1)}}{n-2}$

where $n = V_1$ for weighted data.

Value

Population-weighted skewness value(s) with sample size information, including the weighted skewness, effective sample size (effective N), and the number of valid observations used.

References

Joanes, D. N., & Gill, C. A. (1998). Comparing measures of sample skewness and kurtosis. The Statistician, 47(1), 183-189.

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`w_kurtosis` for weighted kurtosis (tail heaviness).

`describe` for comprehensive descriptive statistics including skewness.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_var()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted skewness
survey_data %>% w_skew(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_skew(age, income, life_satisfaction, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_skew(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(skew_age = w_skew(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_skew(age)
```

---

w_var *Calculate Population-Representative Variance*

Description

`w_var()` calculates variance that accurately represents your population by using survey weights. Variance measures how far values spread out from the average – it is the square of the standard deviation. A larger variance means more dispersion in your population's responses.

Without weights, you describe the spread in your sample only. With weights, you estimate how spread out values are in the entire population.

Usage

```
w_var(data, ..., weights = NULL, na.rm = TRUE)
```

Arguments

`data`	Your survey data (a data frame or tibble)
`...`	The numeric variables you want to analyze. You can list multiple variables or use helpers like `starts_with("trust")`
`weights`	Survey weights to make results representative of your population. Without weights, you get the simple sample variance. Give a column name (unquoted or as a string), an expression such as `sampling_weight * 2`, or a numeric vector with one weight per row.
`na.rm`	Remove missing values before calculating? (Default: TRUE). With `FALSE`, the result for a variable that contains missing values is `NA` (as in base R).

Details

Understanding the Results:

- **Weighted Variance**: The population-representative variance. Because variance is in squared units (e.g., years-squared for age), it is often easier to interpret the standard deviation (`w_sd`) instead.
- **Effective N**: How many independent observations your weighted data represents. Always less than or equal to the actual sample size.
- **N / Missing**: Valid and missing cases. With weights, both are sums of weights (displayed rounded), as SPSS reports them under `WEIGHT BY`; Kish's effective N is shown by `summary()`.

When to Use This:

Use `w_var()` when:

- You need variance for further calculations (e.g., pooled variance, F-tests)
- You want to compare variability across groups with proper weighting
- Your analysis formulas require variance rather than SD
- You need SPSS-compatible weighted variance values

For reporting purposes, `w_sd` is usually preferred because the standard deviation is in the same units as the original variable.

Formula:

The weighted variance uses the SPSS frequency weights formula:

$s_w^2 = \frac{\sum w_i (x_i - \bar{x}_w)^2}{V_1 - 1}$

where $V_1 = \sum w_i$ is the sum of all weights and $\bar{x}_w = \sum w_i x_i / V_1$ is the weighted mean.

Note: $s_w^2 = (s_w)^2$, i.e., the weighted variance is the square of the weighted standard deviation.

Value

Population-weighted variance(s) with sample size information, including the weighted variance, effective sample size (effective N), and the number of valid observations used.

References

IBM Corp. (2023). IBM SPSS Statistics 29 Algorithms. IBM Corporation.

See Also

`var` for the base R variance function.

`w_sd` for weighted standard deviation (the square root of variance).

`w_mean` for weighted means.

`describe` for comprehensive descriptive statistics including variance.

Other weighted_statistics: `w_iqr()`, `w_kurtosis()`, `w_mean()`, `w_median()`, `w_modus()`, `w_quantile()`, `w_range()`, `w_sd()`, `w_se()`, `w_skew()`

Examples

```
# Load required packages and data
library(dplyr)
data(survey_data)

# Basic weighted variance
survey_data %>% w_var(age, weights = sampling_weight)

# Multiple variables
survey_data %>% w_var(age, income, weights = sampling_weight)

# Grouped data
survey_data %>% group_by(region) %>% w_var(age, weights = sampling_weight)

# In summarise context
survey_data %>% summarise(var_age = w_var(age, weights = sampling_weight))

# Unweighted (for comparison)
survey_data %>% w_var(age)
```

Index

∗ **correlation**
  kendall_tau, 58
  partial_cor, 99
  pearson_cor, 101
  spearman_rho, 184
∗ **data-export**
  write_spss, 248
  write_stata, 250
  write_xlsx, 251
  write_xpt, 254
∗ **data-import**
  na_frequencies, 89
  read_por, 159
  read_sas, 160
  read_spss, 162
  read_stata, 164
  read_xlsx, 165
  read_xpt, 167
  strip_tags, 189
  untag_na, 240
∗ **datasets**
  longitudinal_data, 76
  longitudinal_data_wide, 78
  survey_data, 222
∗ **descriptive**
  crosstab, 30
  describe, 33
  frequency, 52
  multiple_response, 87
  normality_test, 91
∗ **effect_sizes**
  phi, 105
∗ **hypothesis_tests**
  ancova, 6
  binomial_test, 14
  chi_square, 21
  chisq_gof, 19
  factorial_anova, 44
  fisher_test, 49
  friedman_test, 55
  kruskal_wallis, 61
  mann_whitney, 79
  mcnemar_test, 85
  oneway_anova, 93
  t_test, 234
  wilcoxon_test, 246
∗ **labels**
  copy_labels, 29
  drop_labels, 36
  find_var, 47
  set_na, 182
  to_character, 224
  to_label, 227
  to_labelled, 229
  to_numeric, 230
  unlabel, 238
  val_labels, 242
  var_label, 243
∗ **posthoc**
  dunn_test, 37
  levene_test, 64
  pairwise_wilcoxon, 96
  scheffe_test, 179
  tukey_test, 232
∗ **recode**
  rec, 168
  to_dummy, 225
∗ **regression**
  linear_regression, 67
  logistic_regression, 71
  marginal_effects, 83
∗ **scale**
  efa, 40
  pomps, 106
  reliability, 171
  row_count, 175
  row_means, 176
  row_sums, 178

∗ **transform**
center, 17
std, 187
∗ **weighted_statistics**
w_iqr, 256
w_kurtosis, 258
w_mean, 260
w_median, 262
w_modus, 264
w_quantile, 266
w_range, 268
w_sd, 270
w_se, 272
w_skew, 274
w_var, 276

ancova, 6, *46*, *109*, *130*, *190*
ancova(), *16*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *237*, *248*
anova.linear_regression, 9
anova.logistic_regression, 10
aov, *95*
as.data.frame(), *253*
as.data.frame.ancova (as.data.frame.mariposa), 10
as.data.frame.binomial_test (as.data.frame.mariposa), 10
as.data.frame.chi_square (as.data.frame.mariposa), 10
as.data.frame.chisq_gof (as.data.frame.mariposa), 10
as.data.frame.codebook (as.data.frame.mariposa), 10
as.data.frame.crosstab (as.data.frame.mariposa), 10
as.data.frame.describe (as.data.frame.mariposa), 10
as.data.frame.dunn_test (as.data.frame.mariposa), 10
as.data.frame.efa (as.data.frame.mariposa), 10
as.data.frame.factorial_anova (as.data.frame.mariposa), 10
as.data.frame.fisher_test (as.data.frame.mariposa), 10
as.data.frame.frequency (as.data.frame.mariposa), 10
as.data.frame.friedman_test (as.data.frame.mariposa), 10
as.data.frame.kendall_tau (as.data.frame.mariposa), 10
as.data.frame.kruskal_wallis (as.data.frame.mariposa), 10
as.data.frame.levene_test (as.data.frame.mariposa), 10
as.data.frame.linear_regression (as.data.frame.mariposa), 10
as.data.frame.logistic_regression (as.data.frame.mariposa), 10
as.data.frame.mann_whitney (as.data.frame.mariposa), 10
as.data.frame.marginal_effects (as.data.frame.mariposa), 10
as.data.frame.mariposa, 10
as.data.frame.mcnemar_test (as.data.frame.mariposa), 10
as.data.frame.multiple_response (as.data.frame.mariposa), 10
as.data.frame.normality_test (as.data.frame.mariposa), 10
as.data.frame.oneway_anova (as.data.frame.mariposa), 10
as.data.frame.pairwise_wilcoxon (as.data.frame.mariposa), 10
as.data.frame.partial_cor (as.data.frame.mariposa), 10
as.data.frame.pearson_cor (as.data.frame.mariposa), 10
as.data.frame.reliability (as.data.frame.mariposa), 10
as.data.frame.scheffe_test (as.data.frame.mariposa), 10
as.data.frame.spearman_rho (as.data.frame.mariposa), 10
as.data.frame.t_test (as.data.frame.mariposa), 10
as.data.frame.tukey_test (as.data.frame.mariposa), 10
as.data.frame.w_iqr (as.data.frame.mariposa), 10
as.data.frame.w_kurtosis (as.data.frame.mariposa), 10
as.data.frame.w_mean (as.data.frame.mariposa), 10
as.data.frame.w_median (as.data.frame.mariposa), 10
as.data.frame.w_modus

(as.data.frame.mariposa), 10
as.data.frame.w_quantile
(as.data.frame.mariposa), 10
as.data.frame.w_range
(as.data.frame.mariposa), 10
as.data.frame.w_sd
(as.data.frame.mariposa), 10
as.data.frame.w_se
(as.data.frame.mariposa), 10
as.data.frame.w_skew
(as.data.frame.mariposa), 10
as.data.frame.w_var
(as.data.frame.mariposa), 10
as.data.frame.wilcoxon_test
(as.data.frame.mariposa), 10

base::as.data.frame(), *13*
binom.test, *16*
binomial_test, 14, *20, 21*, *51*, *130*, *191*
binomial_test(), *8*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *237*, *248*

center, 17
center(), *188*, *249*
chi_square, *16*, *20, 21*, 21, *31, 32*, *49*, *51*, *55*, *74*, *86*, *105, 106*, *111, 112*, *132*, *193*, *195*
chi_square(), *8*, *14*, *16*, *21*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *237*, *248*
chisq.test, *23*
chisq_gof, 19, *131*, *192*
chisq_gof(), *8*, *16*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *237*, *248*
codebook, 24
codebook(), *36*, *112*, *161*, *164*, *182, 183*, *194*, *244*, *251–253*
coef.linear_regression, 27
coef.logistic_regression (logistic_regression-accessors), 75
confint.linear_regression, 27
confint.logistic_regression, 28, *74*
copy_labels, 29
copy_labels(), *36*, *49*, *183*, *225*, *228, 229*, *231*, *239*, *243, 244*
cor, *60*, *104*, *186*
cor.test, *104*
cramers_v (phi), 105
crosstab, *23*, 30, *55*, *89*, *112*, *133*, *195*
crosstab(), *13, 14*, *35*, *55*, *89*, *92*, *251*, *253*

describe, 33, *55*, *70*, *92*, *134*, *196*, *223*, *257*, *259*, *261*, *263*, *265*, *267*, *269*, *271*, *273*, *275*, *277*
describe(), *13*, *26*, *32*, *55*, *89*, *92*, *244*, *251*, *253*
df.residual.linear_regression, 35
dplyr::arrange(), *29*
dplyr::bind_rows(), *10*
dplyr::filter(), *29*
dplyr::group_by(), *14*
dplyr::select(), *29*
drop_labels, 36
drop_labels(), *29*, *49*, *183*, *225*, *228, 229*, *231*, *239*, *243, 244*
dunn_test, 37, *98, 99*, *113*, *135*, *197*
dunn_test(), *67*, *99*, *181*, *234*

efa, 40, *114*, *136*, *173*, *199*
efa(), *14*, *107*, *174, 175*, *177*, *179*

factorial_anova, *8*, 44, *115*, *137*, *200*
factorial_anova(), *8*, *16*, *21*, *23*, *51*, *57*, *63*, *82*, *86*, *96*, *237*, *248*
find_var, 47
find_var(), *29*, *36*, *183*, *225*, *228, 229*, *231*, *239*, *243, 244*
fisher_test, *20*, 49, *138*, *200*
fisher_test(), *8*, *16*, *21*, *23*, *46*, *57*, *63*, *82*, *86*, *96*, *237*, *248*
fitted.linear_regression, 52
fitted.logistic_regression (logistic_regression-accessors), 75
fre (frequency), 52
frequency, *23*, *32*, *35*, 52, *89*, *138*, *201*, *223*, *265*
frequency(), *14*, *26*, *32*, *35, 36*, *89, 90*, *92*, *161*, *163, 164*, *182, 183*, *244*, *251*, *253*
friedman.test, *57*
friedman_test, 55, *98, 99*, *139*, *202*
friedman_test(), *8*, *16*, *21*, *23*, *46*, *51*, *63*, *82*, *86*, *96*, *237*, *248*

goodman_gamma (phi), 105
group_by, *237*

haven::na_tag(), *163*

haven::read_dta(), *165*
haven::read_por(), *160*
haven::read_sas(), *162*
haven::read_sav(), *163*
haven::read_xpt(), *168*
haven::tagged_na(), *160*, *162*

interactive(), *25*
IQR, *257*

kendall_tau, 58, *104*, *118*, *140*, *186*, *203*
kendall_tau(), *101*, *104*, *186*
kruskal.test, *63*
kruskal_wallis, *38*, *39*, *57*, 61, *92*, *98*, *141*, *204*
kruskal_wallis(), *8*, *16*, *21*, *23*, *46*, *51*, *57*, *82*, *86*, *96*, *237*, *248*

levene_test, *46*, 64, *91*, *92*, *95*, *141*, *181*, *205*, *234*
levene_test(), *39*, *99*, *181*, *234*
linear_regression, *8*, 67, *73*, *74*, *84*, *100*, *101*, *120*, *142*, *206*, *207*, *241*
linear_regression(), *13*, *74*, *84*, *251*, *253*
logistic_regression, *28*, *69*, *70*, 71, *83*, *84*, *120*, *143*, *208*, *241*
logistic_regression(), *70*, *84*
logistic_regression-accessors, 75
longitudinal_data, 76, *78*
longitudinal_data_wide, *77*, 78

mann_whitney, *63*, 79, *92*, *121*, *144*, *209*, *247*, *248*
mann_whitney(), *8*, *16*, *21*, *23*, *46*, *51*, *57*, *63*, *86*, *96*, *237*, *248*
marginal_effects, 83, *122*, *144*, *209*
marginal_effects(), *70*, *74*
mcnemar_test, *51*, 85, *145*, *210*
mcnemar_test(), *8*, *16*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *96*, *237*, *248*
median, *263*
merge(), *29*
multiple_response, 87, *123*, *146*, *211*
multiple_response(), *32*, *35*, *55*, *92*

na_frequencies, 89, *170*
na_frequencies(), *160*, *162*, *163*, *165*, *167*, *168*, *182*, *183*, *189*, *240*
nobs.linear_regression, 90
nobs.logistic_regression (logistic_regression-accessors), 75
normality_test, 91, *124*, *146*, *147*, *212*
normality_test(), *13*, *32*, *35*, *55*, *89*

oneway.test, *95*
oneway_anova, *8*, *35*, *46*, *63*, *67*, *92*, 93, *125*, *148*, *181*, *213*, *234*
oneway_anova(), *8*, *16*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *237*, *248*, *251*

pairwise_wilcoxon, 96, *126*, *148*, *214*
pairwise_wilcoxon(), *39*, *67*, *181*, *234*
partial_cor, 99, *126*, *149*, *215*
partial_cor(), *60*, *104*, *186*
pearson_cor, *60*, *70*, *92*, *100*, *101*, 101, *127*, *150*, *174*, *186*, *216*
pearson_cor(), *60*, *101*, *186*
phi, 105
pomps, 106
pomps(), *43*, *174*, *175*, *177*, *179*, *188*, *249*
predict.linear_regression, 107
predict.logistic_regression, 108
print, *113*, *126*, *128*, *156*
print.ancova, 109
print.binomial_test, 109
print.chi_square, 111
print.chisq_gof, 110
print.codebook, 111
print.crosstab, 112
print.describe, 113
print.dunn_test, 113
print.efa, 114
print.factorial_anova, 115
print.fisher_test, 115
print.frequency, 116
print.friedman_test, 117
print.kendall_tau, 117
print.kruskal_wallis, 118
print.levene_test, 119
print.linear_regression, 119
print.logistic_regression, 120
print.mann_whitney, 121
print.marginal_effects, 122
print.mcnemar_test, 122
print.multiple_response, 123
print.normality_test, 124
print.oneway_anova, 125

print.pairwise_wilcoxon, 125
print.partial_cor, 126
print.pearson_cor, 127
print.reliability, 127
print.scheffe_test, 128
print.spearman_rho, 129
print.summary.ancova, 129
print.summary.binomial_test, 130
print.summary.chi_square, 132
print.summary.chisq_gof, 131
print.summary.codebook, 132
print.summary.codebook(), *194*
print.summary.crosstab, 133
print.summary.describe, 134
print.summary.dunn_test, 135
print.summary.efa, 136
print.summary.factorial_anova, 136
print.summary.fisher_test, 137
print.summary.frequency, 138
print.summary.friedman_test, 139
print.summary.kendall_tau, 139
print.summary.kruskal_wallis, 140
print.summary.levene_test, 141
print.summary.linear_regression, 142
print.summary.logistic_regression, 142
print.summary.mann_whitney, 143
print.summary.marginal_effects, 144
print.summary.mcnemar_test, 145
print.summary.multiple_response, 146
print.summary.normality_test, 146
print.summary.oneway_anova, 147
print.summary.pairwise_wilcoxon, 148
print.summary.partial_cor, 149
print.summary.pearson_cor, 150
print.summary.reliability, 150
print.summary.scheffe_test, 151
print.summary.spearman_rho, 152
print.summary.t_test, 154
print.summary.tukey_test, 153
print.summary.w_quantile, 155
print.summary.w_statistic, 156
print.summary.wilcoxon_test, 154
print.t_test, 157, *237*
print.tukey_test, 156
print.w_modus, 158
print.wilcoxon_test, 158
profile.logistic_regression, 159

quantile, *267*

range, *269*
read_por, 159
read_por(), *26*, *90*, *162*, *163*, *165*, *167*, *168*, *189*, *240*, *253*
read_sas, 160
read_sas(), *25*, *26*, *54*, *89*, *90*, *160*, *163*, *165*, *167*, *168*, *189*, *240*, *242*, *253*, *255*
read_spss, *48*, 162, *170*, *175*
read_spss(), *25*, *26*, *54*, *89*, *90*, *159*, *160*, *162*, *164*, *165*, *167*, *168*, *183*, *189*, *240*, *242*, *248*, *249*, *253*
read_stata, 164
read_stata(), *25*, *26*, *54*, *89*, *90*, *160*, *162*, *163*, *167*, *168*, *189*, *240*, *242*, *251*, *253*
read_xlsx, 165
read_xlsx(), *90*, *160*, *162*, *163*, *165*, *168*, *189*, *240*, *253*
read_xpt, 167
read_xpt(), *25*, *26*, *54*, *89*, *90*, *160*, *162*, *163*, *165*, *167*, *189*, *240*, *253*, *255*
rec, 168
rec(), *226*, *249*
reliability, *43*, *128*, *151*, 171, *177*, *217*
reliability(), *14*, *43*, *107*, *175*, *177*, *179*, *251*, *253*
residuals.linear_regression, 174
residuals.logistic_regression *(*logistic_regression-accessors*)*, *75*
row_count, 175
row_count(), *43*, *107*, *174*, *177*, *179*
row_means, *43*, *107*, *173*, *174*, 176
row_means(), *43*, *107*, *174*, *175*, *179*
row_sums, 178
row_sums(), *43*, *107*, *174*, *175*, *177*

scheffe_test, *38*, *39*, *128*, *152*, 179, *218*
scheffe_test(), *39*, *67*, *99*, *234*
sd, *271*
set_na, 182
set_na(), *29*, *36*, *49*, *171*, *225*, *228*, *229*, *231*, *239*, *243*, *244*
spearman_rho, *60*, *92*, *104*, *129*, *152*, 184, *219*
spearman_rho(), *60*, *101*, *104*
std, 187
std(), *18*, *249*
strip_tags, 189

strip_tags(), *90*, *160*, *162, 163*, *165*, *167, 168*, *183*, *239, 240*, *249*
summary, *35*
summary.ancova, *8*, *129, 130*, 190
summary.binomial_test, *130*, 191
summary.chi_square, *23*, *132*, 193
summary.chisq_gof, *131*, 192
summary.codebook, 194
summary.codebook(), *133*
summary.crosstab, *133*, 195
summary.describe, *134*, 196
summary.dunn_test, *135*, 197
summary.efa, *43*, *136*, 198
summary.factorial_anova, *46*, *136, 137*, 199
summary.fisher_test, *137, 138*, 200
summary.frequency, *138*, 201
summary.friedman_test, *139*, 202
summary.kendall_tau, *60*, *139, 140*, 203
summary.kruskal_wallis, *140, 141*, 204
summary.levene_test, *141*, 205
summary.linear_regression, *70*, *142*, 206
summary.logistic_regression, *74*, *142, 143*, 207
summary.mann_whitney, *82*, *143, 144*, 208
summary.marginal_effects, *84*, *144*, 209
summary.mcnemar_test, *145*, 210
summary.multiple_response, *89*, *146*, 211
summary.normality_test, *92*, *147*, 212
summary.oneway_anova, *95*, *147, 148*, 212
summary.pairwise_wilcoxon, *148*, 213
summary.partial_cor, *101*, *149*, 214
summary.pearson_cor, *104*, *150*, 215
summary.reliability, *150, 151*, *174*, 216
summary.scheffe_test, *151, 152*, 217
summary.spearman_rho, *152*, *186*, 218
summary.t_test, *154*, 220, *237*
summary.tukey_test, *153*, 219
summary.wilcoxon_test, *154, 155*, 221
survey_data, *77*, 222
svyranktest, *82*

t.test, *237*
t_test, *35*, *67*, *82*, *92*, *154*, *157*, *221*, *223*, 234
t_test(), *8*, *13, 14*, *16*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *248*, *251*, *253*
table, *32*, *55*
tibble::as_tibble(), *10*
to_character, 224
to_character(), *29*, *36*, *49*, *183*, *228, 229*, *231*, *239*, *243, 244*
to_dummy, 225
to_dummy(), *171*
to_label, *69*, *72*, *225*, 227
to_label(), *29*, *36*, *49*, *171*, *183*, *224–226*, *229–231*, *239*, *243, 244*
to_labelled, 229
to_labelled(), *29*, *36*, *49*, *183*, *225*, *228*, *231*, *239*, *242–244*
to_numeric, 230
to_numeric(), *29*, *36*, *49*, *183*, *225*, *228, 229*, *239*, *243, 244*, *249*
tukey_test, *38, 39*, *46*, *95*, *153*, *156*, *181*, *220*, 232
tukey_test(), *39*, *67*, *99*, *181*
TukeyHSD, *234*

unlabel, 238
unlabel(), *29*, *36*, *49*, *183*, *225*, *228, 229*, *231*, *243, 244*
untag_na, 240
untag_na(), *90*, *160–165*, *167, 168*, *183*, *189*, *249*
update.linear_regression, 241
update.logistic_regression, 241
utils::write.csv(), *10*

val_labels, 242
val_labels(), *29*, *36*, *49*, *183*, *225*, *228, 229*, *231*, *239*, *244*
var, *277*
var.test, *67*
var_label, 243
var_label(), *29*, *36*, *49*, *183*, *225*, *228, 229*, *231*, *239*, *243*
var_label<- *(*var_label*)*, 243
vcov.linear_regression, 245
vcov.logistic_regression *(*logistic_regression-accessors*)*, *75*

w_iqr, 256, *267*, *269*
w_iqr(), *259*, *261*, *263*, *265*, *268, 269*, *271*, *273*, *275*, *277*
w_kurtosis, 258, *275*
w_kurtosis(), *257*, *261*, *263*, *265*, *268, 269*, *271*, *273*, *275*, *277*
w_mean, *35*, 260, *263*, *265*, *271*, *273*, *277*

w_mean(), *257*, *259*, *263*, *265*, *268, 269*, *271*, *273*, *275*, *277*
w_median, *35*, *261*, 262, *265*, *267*
w_median(), *257*, *259*, *261*, *265*, *268, 269*, *271*, *273*, *275*, *277*
w_modus, 264
w_modus(), *257*, *259*, *261*, *263*, *268, 269*, *271*, *273*, *275*, *277*
w_quantile, *257*, *263*, 266
w_quantile(), *257*, *259*, *261*, *263*, *265*, *269*, *271*, *273*, *275*, *277*
w_range, *257*, 268
w_range(), *257*, *259*, *261*, *263*, *265*, *268*, *271*, *273*, *275*, *277*
w_sd, *35*, *257*, *261*, *269*, 270, *273*, *277*
w_sd(), *257*, *259*, *261*, *263*, *265*, *268, 269*, *273*, *275*, *277*
w_se, 272
w_se(), *257*, *259*, *261*, *263*, *265*, *268, 269*, *271*, *275*, *277*
w_skew, *259*, 274
w_skew(), *257*, *259*, *261*, *263*, *265*, *268, 269*, *271*, *273*, *277*
w_var, *271*, 276
w_var(), *257*, *259*, *261*, *263*, *265*, *268, 269*, *271*, *273*, *275*
weighted.mean, *261*
wilcox.test, *82*, *248*
wilcoxon_test, *57*, *86*, *98, 99*, *155*, *222*, 246
wilcoxon_test(), *8*, *16*, *21*, *23*, *46*, *51*, *57*, *63*, *82*, *86*, *96*, *237*
write_spss, *170, 171*, 248
write_spss(), *163*, *189*, *231*, *244*, *251*, *253*, *255*
write_stata, *171*, 250
write_stata(), *189*, *249*, *253*, *255*
write_xlsx, 251
write_xlsx(), *10*, *14*, *26*, *165–167*, *249*, *251*, *255*
write_xpt, 254
write_xpt(), *249*, *251*, *253*