Package: factorH (via r-universe)

May 9, 2026

Type Package

Title Multifactor Nonparametric Rank-Based ANOVA with Post Hoc Tests

Version 0.6.0

Description Multifactor nonparametric analysis of variance based on ranks. Builds on the Kruskal-Wallis H test and its 2x2 Scheirer-Ray-Hare extension to handle any factorial designs. Provides effect sizes, Dunn-Bonferroni pairwise-comparison matrices, and simple-effects analyses. Tailored for psychology and the social sciences, with beginner-friendly R syntax and outputs that can be dropped into journal reports. Includes helpers to export tab-separated results and compact tables of descriptive statistics (to APA-style reports).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1)

Imports rcompanion, FSA, car, stats, utils

Suggests MASS, ARTool, testthat (>= 3.0.0), knitr, rmarkdown, haven

Config/testthat/edition 3

VignetteBuilder knitr

Contact tomasz.rak@upjp2.edu.pl

LazyData true

NeedsCompilation no

Author Tomasz Rak [aut, cre], Szymon Wrzesniowski [aut]

Maintainer Tomasz Rak <tomasz.rak@upjp2.edu.pl>

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

Date/Publication 2026-04-09 11:29:36 UTC

RemoteUrl https://github.com/cran/factorH

RemoteRef HEAD

RemoteSha 243d973e1aaebc89c1846304e23f2043275ab838

Contents

`as_jamovi_srh_full` *Convert srh.kway.full() output to a Jamovi-ready structure*

Description

Normalizes the output of `srh.kway.full()` into a stable list structure that can be mapped by a Jamovi backend into:

- one ANOVA table,
- one descriptives/summary table,
- dynamic post hoc tables for all effects,
- dynamic simple-effects tables,
- optional plan diagnostics,
- compact meta/info block.

Usage

```
as_jamovi_srh_full(
  x,
  show_diagnostics = TRUE,
  show_intercept = FALSE,
  keep_empty = FALSE,
  posthoc_cells_view = c("long", "matrix"),
  plan_diagnostics = NULL
)
```

Arguments

x	A result returned by `srh.kway.full()`.
show_diagnostics	Logical; should diagnostics be included when available? Default `TRUE`.
show_intercept	Logical; should the `(Intercept)` row from ANOVA be kept when present? Default `FALSE`.
keep_empty	Logical; if `TRUE`, empty/non-applicable sections are returned with `visible = TRUE` and a status/message. If `FALSE`, such sections are marked hidden. Default `FALSE`.
posthoc_cells_view	Character string, currently one of `"long"` or `"matrix"`. Default `"long"`.
plan_diagnostics	Optional result returned by `plan.diagnostics()`. If supplied, it is normalized into an additional dynamic section.

Details

The function is intentionally package-side only: it does not build Jamovi result objects. Instead, it prepares plain R objects with predictable fields (`data`, `items`, `visible`, `status`, `message`).

Value

A named list with sections:

- `anova`
- `summary`
- `posthoc_cells`
- `posthoc_simple`
- `diagnostics`
- `plan_diagnostics`
- `meta`

`balance.chisq.datatable` *Count-balance chi-square diagnostics across factors*

Description

For one factor: chi-square goodness-of-fit vs equal proportions. For two factors: chi-square test of independence. For three or more: log-linear independence (Poisson, main effects only) via deviance and df.

Usage

```
balance.chisq.datatable(formula, data, force_factors = TRUE, correct = FALSE)
```

Arguments

`formula`	A model formula `y ~ A + B (+ C ...)`; the response is ignored.
`data`	A data frame with the variables.
`force_factors`	Logical; if TRUE, coerces RHS predictors to factors.
`correct`	Logical; continuity correction for 2x2 tables in `chisq.test` (default FALSE).

Details

Uses `stats::chisq.test` for 1–2 factors. For 3+ factors, prefers `MASS::loglm` if available; otherwise falls back to a Poisson GLM on the count table.

Value

A data.frame with one row per factor combination (`Effect`) and columns: `n`, `ChiSq` (4 decimals), `df`, `p.chisq` (4 decimals), `OK`.

See Also

`plan.diagnostics`

Examples

```
## Not run:
balance.chisq.datatable(liking ~ gender + condition + age_cat, data = mimicry)

## End(Not run)
```

`factorH` *factorH: Multifactor rank-based ANOVA utilities*

Description

Multifactor nonparametric analysis of variance based on ranks. Builds on the Kruskal-Wallis H test and its 2x2 Scheirer-Ray-Hare extension to handle any factorial designs. Provides effect sizes, Dunn-Bonferroni pairwise-comparison matrices, and simple-effects analyses. Tailored for psychology and the social sciences, with beginner-friendly R syntax and outputs that can be dropped into journal reports. Includes helpers to export tab-separated results and compact tables of descriptive statistics (to APA-style reports).

Details

What this package does (and why):

factorH provides a simple, single-call workflow for multifactor nonparametric, rank-based ANOVA and publication-ready outputs:

- ANOVA-like tables based on ranks
- rank-based effect sizes computed from H
- Dunn-Bonferroni post hoc comparison matrices
- simple-effects post hocs (pairwise comparisons within levels of conditioning factors)
- compact descriptive tables
- one-call diagnostics for factorial plans
- TSV export for quick formatting
- a Jamovi-ready normalization helper for backend integration

Why? Popular GUI stats tools do not offer a ready-made, user-friendly multifactor rank-based pipeline that mirrors standard H / SRH analyses in a way that is easy for beginners. *factorH* aims to fill that gap with clear R-like formula syntax and a one-command report function.

The package is intentionally small: most users will only ever need:

- `srh.kway.full(...)` to compute the full pipeline
- `write.srh.kway.full.tsv(...)` to export the results into a single tab-separated file

Advanced integrations can additionally use:

- `as_jamovi_srh_full(...)` to normalize `srh.kway.full()` output into a stable Jamovi/backend structure

Formula syntax at a glance:

All high-level functions use standard R model formulas:

```
response ~ factorA + factorB + factorC
```

- `+` lists the main effects.
- Interactions are handled internally, so you do not need to write `A:B` or `A*B`.
- The response (left of `~`) must be numeric (e.g., a Likert score coded as 1 to 5 and stored as numeric).

Examples below use the included dataset `mimicry`.

```
library(factorH)
data(mimicry, package = "factorH")
str(mimicry)
```

Predictors should be factors. If they are not, the functions will coerce them to factors internally.

What is allowed?

```
# One factor (KW-style):
liking ~ condition

# Two factors (SRH-style):
liking ~ gender + condition

# Three or more factors (k-way):
liking ~ gender + condition + age_cat
```

You do not need to write `gender:condition` or `gender*condition`. The package constructs the required interaction terms internally when needed.

Numeric response (Likert note):

The response must be numeric. For Likert-type responses (e.g., 1 = strongly disagree, ..., 5 = strongly agree), keep the variable numeric. Rank-based procedures can be used with such ordinal-like data.

If your Likert variable has been imported as a factor or character, coerce it safely:

```
# if stored as character "1", "2", ...:
mimicry$liking <- as.numeric(mimicry$liking)

# if stored as factor with labels "1", "2", ...:
mimicry$liking <- as.numeric(as.character(mimicry$liking))
```

Diagnostics at a glance:

Most users can cover assumption checks with a single command:

```
diag_out <- plan.diagnostics(response ~ factorA + factorB (+ factorC ...), data = your_data)
```

What it does:

1. Raw normality: Shapiro-Wilk in each subgroup and interaction cell of the specified factors.
2. Residual normality per cell: Shapiro-Wilk on residuals from the corresponding full-factorial ANOVA, tested within each cell.
3. Homogeneity of variances: Levene/Brown-Forsythe across full-plan cells (median by default).
4. Count balance: chi-square homogeneity / independence / log-linear independence across factors.
5. It prints a concise overall summary (share of OK and overall status) and returns all detailed tables in `diag_out$results`, with per-type OK percentages in `diag_out$summary`.

For most workflows, this single command is enough to document design diagnostics alongside rank-based analyses.

The one-call pipeline:

The main function `srh.kway.full()` runs:

1. an ANOVA-like table on ranks
2. a descriptive summary
3. post hoc matrices (Dunn; adjusted `P.adj`)
4. simple-effects post hocs

It also supports: - `type = 2` vs `type = 3` - `scope = "within"` vs `scope = "global"` for simple-effects Bonferroni - design diagnostics and warnings stored in `res$meta`

For 2 factors:

```
res2 <- srh.kway.full(liking ~ gender + condition, data = mimicry)
names(res2)
res2$anova
head(res2$summary)
names(res2$posthoc_cells)
names(res2$posthoc_simple)
```

For 2 factors with Type III SS:

```
res2_t3 <- srh.kway.full(liking ~ gender + condition, data = mimicry, type = 3)
res2_t3$anova
```

For 3 factors:

```
res3 <- srh.kway.full(liking ~ gender + condition + age_cat, data = mimicry)
res3$anova
```

For global simple-effects Bonferroni:

```
res3g <- srh.kway.full(
  liking ~ gender + condition + age_cat,
  data = mimicry,
  scope = "global"
)
names(res3g$posthoc_simple)
```

Export full result to a tab-separated file:

```
f <- file.path(tempdir(), "result.tsv")
write.srh.kway.full.tsv(res3, file = f, dec = ".")
file.exists(f)
```

If you need a decimal comma:

```
f2 <- file.path(tempdir(), "result_comma.tsv")
write.srh.kway.full.tsv(res3, file = f2, dec = ",")
file.exists(f2)
```

The TSV contains clearly separated sections:

- `## SRH: EFFECTS TABLE`
- `## SUMMARY STATS`

- ## POSTHOC CELLS
- ## SIMPLE EFFECTS
- ## META

and can be easily opened in spreadsheet software such as Excel or Google Sheets.

Jamovi/backend helper:
If you want to map the full pipeline into a Jamovi module or another structured frontend, use:

```
jam <- as_jamovi_srh_full(res3)
names(jam)
```

This helper does not build Jamovi result objects directly. Instead, it normalizes the `srh.kway.full()` output into a predictable list of sections and items that can be consumed by a backend.

What is in the example dataset?:
`mimicry` is a real study on the chameleon effect (Trzmielewska et al., 2025) about how movement conditions affect `liking` of an interlocutor. Potential moderators include `gender` and `age` (with dichotomized `age_cat` and a 3-level `age_cat2`). This makes it a natural playground for multifactor rank-based analyses.

```
table(mimicry$condition)
table(mimicry$gender)
table(mimicry$age_cat)
```

What the functions compute (high level):

- `srh.kway()`: rank-based k-way ANOVA table using Type II SS by default, with an optional switch to Type III SS; p-values are tie-corrected; H is reported with and without the correction factor; effect sizes are computed from unadjusted H.
- `srh.effsize()`: 2-factor SRH table with effect sizes (`eta2H`, `eps2H`) computed from H.
- `nonpar.datatable()`: compact descriptive tables with global mean ranks, medians, quartiles, IQR, etc., for all main effects and interactions.
- `srh.posthocs()`: Dunn-Bonferroni pairwise matrices (`P.adj`) for all effects (main effects and interactions).
- `srh.simple.posthoc()` / `srh.simple.posthocs()`: simple-effects pairwise comparisons within levels of conditioning factors (`scope = "within"` by default).
- `srh.kway.full()`: orchestrates all of the above.
- `write.srh.kway.full.tsv()`: exports everything into one TSV (with a dot or comma decimal mark).
- `as_jamovi_srh_full()`: normalizes full-pipeline results for Jamovi/backend integration.
- `plan.diagnostics()`: one-call diagnostics: raw normality, residuals cellwise normality, Levene (median), and balance chi-square; prints an overall summary and returns full tables.

That is it. For most users, the intro ends here: use `srh.kway.full()` and export with `write.srh.kway.full.tsv()`.

Author(s)

Maintainer: Tomasz Rak <tomasz.rak@upjp2.edu.pl>

Authors:

- Szymon Wrzesniowski <szymon.wrzesniowski@upjp2.edu.pl>

factorH_dataset *Datasets in factorH*

Description

Datasets in factorH

Details

What is in the example dataset?:

mimicry is a real study on the **chameleon effect** by Trzmielewska et al. (2025) doi:10.18290/rpsych2024.0019 about how `mimicry` vs other movement conditions affect `liking` of an interlocutor. Potential moderators include `gender` and `age` (with dichotomized age_cat, and a 3-level age_cat2). This makes it a natural playground for `multifactor` rank-based analyses.

```
table(mimicry$condition)
table(mimicry$gender)
table(mimicry$age_cat)
```

factorH_reference *factorH functions reference*

Description

factorH functions reference

Details

Function reference:

This document collects **call patterns** and **options** for each public function. All formulas follow `response ~ A + B (+ C ...)` with a **numeric** response and **factor** predictors.

srh.kway.full()

Purpose: one-call pipeline for rank-based ANOVA, descriptive statistics, post hocs, and simple effects. **Syntax:** `srh.kway.full(y ~ A + B (+ C ...), data, max_levels = 30, type = 2, scope = c("within", "g`

- Automatically chooses the ANOVA engine:
  - 1 factor: `srh.kway()`
  - 2 factors with `type = 2`: `srh.effsize()`
  - 2 factors with `type = 3`: `srh.kway()`
  - 3+ factors: `srh.kway()`
- Returns a list with the following components:
  - `anova`
  - `summary`
  - `posthoc_cells`

  - posthoc_simple
  - meta
- Placeholders:
  - "[not applicable]" when a component does not apply (e.g., simple effects for a one-factor design),
  - "[failed] ..." when a sub-step fails but the overall pipeline continues.

Example:

```
res <- srh.kway.full(liking ~ gender + condition + age_cat, data = mimicry)
names(res)
res$anova[1:3]
head(res$summary)
names(res$posthoc_cells)
names(res$posthoc_simple)
res$meta
```

Notes:

- Predictors are coerced to factors internally; each factor must have between 2 and max_levels levels.
- Rows with missing values in variables used in the formula are removed using complete-case filtering.
- type must be either 2 or 3.
- scope controls Bonferroni adjustment in posthoc_simple; the default is "within", which is passed down to srh.simple.posthocs() and srh.simple.posthoc().
- For one-factor designs, type is accepted for interface consistency, but it has no practical effect on the result.
- For two-factor designs, type = 2 keeps the SRH-style pipeline via srh.effsize(), whereas type = 3 routes the analysis through srh.kway() to follow the logic of Type III sums of squares.
- For designs with 3 or more factors, the ANOVA step is handled by srh.kway() using the requested type.
- For incomplete or sparse factorial plans, the analysis may still run; design-related warnings are stored in res$meta$warnings.

write.srh.kway.full.tsv()

Purpose: export the srh.kway.full() result into a single TSV file for fast formatting. **Syntax:** write.srh.kway.full.tsv(obj, file = "srh_kway_full.tsv", sep = "\t", na = "", dec = ".")

- dec = "." or "," controls the decimal mark.
- Numeric fields are written without scientific notation.
- Pretty-printed character tables (e.g., from post hocs) are normalized so that dec = "," also affects numbers embedded in strings.
- The META section exports n, levels, scope, design diagnostics, warnings, and the original call when available.

Example:

```
f <- file.path(tempdir(), "result.tsv")
write.srh.kway.full.tsv(res, file = f, dec = ",")
file.exists(f)
```

srh.kway()

Purpose: general k-way SRH-style ANOVA on ranks, tie-corrected p-values, and rank-based effect sizes. **Syntax:** `srh.kway(y ~ A + B (+ C ...), data, clamp0 = TRUE, force_factors = TRUE, type = 2, ...)`

- Reports: `Effect`, `Df`, `Sum Sq`, `H`, `Hadj` (tie correction), `p.chisq`, `k`, `n`, `eta2H`, `eps2H`.
- `eta2H` and `eps2H` are computed from **unadjusted H** (classical SRH practice).
- `force_factors = TRUE` coerces predictors to factor (recommended).
- `type` controls sums of squares. Default `type = 2` (Type II SS). Set `type = 3` for Type III SS (internally uses sum-to-zero contrasts; no global options are changed).

Example:

```
k3 <- srh.kway(liking ~ gender + condition + age_cat, data = mimicry)
k3
```

One-factor check (KW-like):

```
k1 <- srh.kway(liking ~ condition, data = mimicry)
k1
```

Two-factor Type III SS:

```
k2_ss3 <- srh.kway(liking ~ gender + condition, data = mimicry, type = 3)
k2_ss3
```

srh.effsize()

Purpose: 2-factor SRH table with effect sizes from H. **Syntax:** `srh.effsize(y ~ A + B, data, clamp0 = TRUE, ...)`

- Same columns as above but tailored to the 2-factor SRH pipeline.
- `clamp0 = TRUE` clamps small negatives to 0 for effect sizes.
- This is the default 2-factor engine used by `srh.kway.full(..., type = 2)`.

Example:

```
e2 <- srh.effsize(liking ~ gender + condition, data = mimicry)
e2
```

nonpar.datatable()

Purpose: compact descriptive tables (APA-style), with **global mean ranks**, medians, quartiles, and IQR. **Syntax:** `nonpar.datatable(y ~ A + B (+ C ...), data, force_factors = TRUE)`

- Returns rows for all **main effects** and all **interaction cells** constructed from the RHS.
- Mean ranks are computed on **global ranks** (all observations ranked together), which matches how omnibus rank-based factorial effects are formed.

Example:

```
dt <- nonpar.datatable(liking ~ gender + condition, data = mimicry)
head(dt)
```

srh.posthoc()

Purpose: Dunn-Bonferroni **pairwise comparison matrix** for one specified effect. **Syntax:** `srh.posthoc(y ~ A (+ B + ...), data, method = "bonferroni", digits = 3, triangular = c("lower","upper","fu numeric = FALSE, force_factors = TRUE, sep = ".")`

- Builds a single grouping variable (cells) from the RHS factors and runs `FSA::dunnTest()`.
- Returns a list of three matrices (as data frames): `Z`, `P.unadj`, `P.adj`.
- `triangular = "lower"` (default) shows only the lower triangle; diagonal and upper triangle are masked.
- `numeric = FALSE` returns pretty-printed character tables; set `TRUE` to get numeric tables.

Example:

```
ph <- srh.posthoc(liking ~ condition, data = mimicry)
```

srh.posthocs()

Purpose: Dunn-Bonferroni **pairwise matrices for all effects** (main effects and interactions).
Syntax: `srh.posthocs(y ~ A + B (+ C ...), data, ...)`

- Iterates `srh.posthoc()` over: `A`, `B`, `C`, `A:B`, `A:C`, `B:C`, `A:B:C`, ...
- Returns a named list: names are `"A"`, `"B"`, `"A:B"`, etc.; each value is a `P.adj` matrix.

Example:

```
phs <- srh.posthocs(liking ~ gender + condition + age_cat, data = mimicry)
names(phs)
phs[["gender:condition"]][1:5, 1:5]
```

srh.simple.posthoc()

Purpose: Simple-effects post hocs (pairwise comparisons **within** levels of conditioning factors).
Syntax: `srh.simple.posthoc(y ~ A + B (+ C ...), data, compare = NULL, scope = c("within","global"), digi`

- `compare` selects the target factor for pairwise comparisons (default: the first RHS factor).
- **Scope**:
  - `"within"` (default): Bonferroni **within each by-table** (SPSS-like),
  - `"global"`: one Bonferroni correction across **all** tests from all by-tables combined.
- Returns a data frame with conditioning columns (`BY`), `Comparison`, `Z`, `P.unadj`, `P.adj`, `m.tests`, `adj.note`. An `"adjustment"` attribute describes the correction.

Example:

```
simp <- srh.simple.posthoc(
  liking ~ gender + condition + age_cat,
  data = mimicry,
  compare = "gender",
  scope = "within"
)
head(simp)
```

srh.simple.posthocs()

Purpose: enumerate **all simple-effect configurations** for a given design. **Syntax:** `srh.simple.posthocs(y ~ A + B (+ C`

- For each target factor and each non-empty combination of the remaining factors as `BY`, runs `srh.simple.posthoc(..., compare = target, scope = scope)`.
- Default is `scope = "within"`, which applies Bonferroni adjustment within each simple-effects table.
- Set `scope = "global"` to apply one Bonferroni adjustment across all pairwise tests within each simple-effects table.

- Returns a named list, with names like `COMPARE(gender) | BY(condition x age_cat)`.

Example:

```
sps <- srh.simple.posthocs(liking ~ gender + condition + age_cat, data = mimicry)
head(names(sps), 6)
```

Global-adjustment variant:

```
sps_g <- srh.simple.posthocs(
  liking ~ gender + condition + age_cat,
  data = mimicry,
  scope = "global"
)
head(names(sps_g), 6)
```

as_jamovi_srh_full()

Purpose: normalize `srh.kway.full()` output into a stable Jamovi-ready list structure. **Syntax:** `as_jamovi_srh_full(x, show_diagnostics = TRUE, show_intercept = FALSE, keep_empty = FALSE, posthoc_cells_view = c("long", "matrix"), plan_diagnostics = NULL)`

- Converts the pipeline result into plain R sections and items that can be mapped by a Jamovi backend.
- Normalizes ANOVA, descriptives, post hoc matrices, simple-effects tables, compact diagnostics, optional full plan diagnostics, and meta/info blocks.
- Supports `"long"` or `"matrix"` view for post hoc cell comparisons.

Example:

```
res <- srh.kway.full(liking ~ gender + condition, data = mimicry)
jam <- as_jamovi_srh_full(res)
names(jam)
```

normality.datatable()

Purpose: Shapiro-Wilk normality tests for the raw response within each subgroup for all factor combinations. **Syntax:** `normality.datatable(y ~ A + B (+ C ...), data, force_factors = TRUE)`

- Returns `Effect`, factor columns, `count`, `W`, `p.shapiro` (fixed-format to 4 decimals, no scientific notation), and `OK/NOT OK` (`p < 0.05 => NOT OK`).

Example:

```
normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)
```

residuals.normality.datatable()

Purpose: Shapiro-Wilk tests on global residuals from a classical ANOVA fitted to the selected factors; one test per model. **Syntax:** `residuals.normality.datatable(y ~ A + B (+ C ...), data, force_factors =`

- Returns one row per `Effect` (`A`, `B`, `A:B`, ...), with `count`, `W`, `p.shapiro` (4 decimals), `OK/NOT OK`.
- This function is retained mainly for continuity with older workflows; for stricter ANOVA-style checking, use the cellwise residual variant.

Example:

```
residuals.normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)
```

residuals.cellwise.normality.datatable()

Purpose: Shapiro-Wilk tests of residuals from a classical ANOVA model, tested separately within each cell. **Syntax:** `residuals.cellwise.normality.datatable(y ~ A + B (+ C ...), data, force_factors = TRUE`

- This matches the classical ANOVA assumption of normal errors per cell.
- Returns rows for every cell across all `Effects`, with `count`, `W`, `p.shapiro` (4 decimals), `OK/NOT OK`.

Example:

```
residuals.cellwise.normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)
```

balance.chisq.datatable()

Purpose: count-balance diagnostics across design factors. **Syntax:** `balance.chisq.datatable(y ~ A + B (+ C ...), da`

- For one factor: chi-square goodness-of-fit vs equal proportions.
- For two factors: chi-square test of independence.
- For three or more: log-linear independence model (Poisson; main effects only), assessed via deviance and df.
- Returns `Effect`, `n`, `ChiSq` (4 decimals), `df`, `p.chisq` (4 decimals), `OK/NOT OK` (`p < 0.05 => NOT OK`).
- The response is ignored; only RHS factors are used to build the tables.

Example:

```
balance.chisq.datatable(liking ~ gender + condition + age_cat, data = mimicry)
```

levene.plan.datatable()

Purpose: Levene/Brown-Forsythe test for homogeneity of variances across full-plan cells (highest-order interaction of RHS factors). **Syntax:** `levene.plan.datatable(y ~ A + B (+ C ...), data, center = "median",`

- This is the primary variance-equality diagnostic for a full factorial plan.
- Returns `F`, `df.num`, `df.den`, `p` (4 decimals), and `OK/NOT OK` (`p < 0.05 => NOT OK`).

Examples:

```
levene.plan.datatable(liking ~ gender + condition + age_cat, data = mimicry)
levene.plan.datatable(liking ~ gender + condition, data = mimicry, center = "mean")
```

plan.diagnostics()

Purpose: orchestrates all diagnostics in one call. **Syntax:** `plan.diagnostics(y ~ A + B (+ C ...), data, force_facto`

- Runs raw normality (cellwise on the response), residuals cellwise normality, Levene/Brown-Forsythe for the full plan (median by default), and balance chi-square tests for all factor combinations.
- Prints a concise console summary and returns full tables in a list.

Returned list:

```
$summary: percent_ok, ok_count, total, overall, plus per-type percentages:
percent_ok_normality_raw, percent_ok_residuals_cellwise, percent_ok_balance_chisq, percent_ok_lever

$results: normality_raw, residuals_cellwise_normality, levene_full_plan, balance_chisq.
```

Examples:

```
diag_out <- plan.diagnostics(liking ~ gender + condition + age_cat, data = mimicry)
diag_out$results$normality_raw
diag_out$results$residuals_cellwise_normality
diag_out$results$levene_full_plan
diag_out$results$balance_chisq
diag_out$summary
```

Formula tips and pitfalls

- Do **not** write `A:B` or `A*B`. Use `A + B (+ C ...)`; the package computes all necessary interaction structures internally.
- The response must be **numeric**. For Likert data, keep it numeric `1..k`.
- Predictors should be **factors**. If they are not, they will be coerced internally.

Example:

```
mimicry$gender <- factor(mimicry$gender)
mimicry$condition <- factor(mimicry$condition)
```

Performance and reproducibility

- The package combines SRH-style logic, rank-based linear-model ANOVA, and Dunn post hocs depending on the function and the selected `type`.
- P-values use tie correction where appropriate; rank-based effect sizes are derived from unadjusted H (classical SRH practice).
- Outputs are plain data frames and lists, easy to save, normalize, and post-process.

`factorH_syntax` *Syntax and formula patterns*

Description

Syntax and formula patterns

Details

Formula syntax at a glance:

All high-level functions use standard R model formulas:

response ~ factorA + factorB + factorC

- + lists the main effects.
- Interactions are handled internally, so you do not need to write `A:B` or `A*B`.
- The response (left of ~) must be numeric (e.g., a Likert score coded as 1 to 5 and stored as numeric).

Examples below use the included dataset `mimicry`.

```
library(factorH)
data(mimicry, package = "factorH")
str(mimicry)
```

Predictors should be factors. If they are not, the functions will coerce them to factors internally.

What is allowed?

```
# One factor (KW-style):
liking ~ condition

# Two factors (SRH-style):
liking ~ gender + condition

# Three or more factors (k-way):
liking ~ gender + condition + age_cat
```

You do not need to write `gender:condition` or `gender*condition`. The package constructs the required interaction terms internally when needed.

Numeric response (Likert note):

The response must be numeric. For Likert-type responses (e.g., 1 = strongly disagree, ..., 5 = strongly agree), keep the variable numeric. Rank-based procedures can be used with such ordinal-like data.

If a Likert variable has been imported as a factor or character, coerce it safely:

```
# if stored as character "1", "2", ...:
mimicry$liking <- as.numeric(mimicry$liking)

# if stored as factor with labels "1", "2", ...:
mimicry$liking <- as.numeric(as.character(mimicry$liking))
```

---

levene.plan.datatable *Levene/Brown-Forsythe test for full-plan cells*

Description

Tests homogeneity of variances across the highest-order interaction (all RHS factors combined), using Levene's test (Brown-Forsythe with median by default).

Usage

```
levene.plan.datatable(
  formula,
  data,
  center = c("median", "mean"),
  force_factors = TRUE
)
```

Arguments

formula	A model formula y ~ A + B (+ C ...).
data	A data frame with the variables.
center	Character, "median" (default) for Brown-Forsythe or "mean" for classical Levene.
force_factors	Logical; if TRUE, coerces RHS predictors to factors.

Details

Internally relies on car::leveneTest. If fewer than two groups or any group has < 2 observations, NA values are returned with a warning.

Value

A one-row data.frame with columns: Effect, n.groups, min.n, df.num, df.den, F, p, OK. Values F and p are formatted to 4 decimals (no scientific notation); OK is "OK" if p >= 0.05, otherwise "NOT OK".

See Also

plan.diagnostics

Examples

```
## Not run:
levene.plan.datatable(liking ~ gender + condition + age_cat, data = mimicry)
levene.plan.datatable(liking ~ gender + condition, data = mimicry, center = "mean")

## End(Not run)
```

---

mimicry *Mimicry dataset*

Description

A dataset used to demonstrate rank-based (nonparametric) multifactor ANOVA.

Usage

```
data(mimicry)
```

Format

A data frame with 533 rows and 7 variables:

condition factor; 5 levels

gender factor; 2 levels

age numeric

age_cat factor; 2 levels

age_cat2 factor; 3 levels

field factor; 2 levels

liking numeric; dependent variable

Details

Factor encodings follow the original SPSS labels converted to R factors.

Source

Converted from an SPSS file as part of the factorH package examples.

References

Trzmielewska, W., Duras, J., Juchacz, A., & Rak, T. (2025). Examining the impact of control condition design in mimicry–liking link research: how motor behavior may impact liking. *Annals of Psychology*, 4, 351–378. doi:10.18290/rpsych2024.0019

`nonpar.datatable` *Compact descriptive tables (APA-style) with global mean ranks*

Description

Produces descriptive statistics for all main effects and interaction cells implied by the RHS of `formula`. The function reports classical descriptive statistics together with mean ranks computed from a single, global ranking of the response variable across all complete observations.

Usage

```
nonpar.datatable(formula, data, force_factors = TRUE)
```

Arguments

`formula`	A formula of the form `y ~ A (+ B + ...)`.
`data`	A `data.frame` containing `y` and the grouping factors.
`force_factors`	Logical; if `TRUE` (default), variables on the right-hand side of the formula are coerced to plain `factor`, even when they are stored numerically (e.g., `0`/`1` instead of labels such as `"male"`/`"female"`).

Details

In rank-based factorial workflows, mean ranks can be computed in two complementary ways.

First, they can be computed *globally*: all observations relevant to a given effect are pooled, ranked together, and then averaged within each subgroup. This is the approach used here. For example, for a main effect such as `condition`, all complete observations are ranked in one common pool, and the resulting ranks are then averaged separately within each level of `condition`. These global mean ranks are the recommended descriptive companion for omnibus rank-based factorial effects.

Second, mean ranks can be computed *within slices* of a moderator, which is typical for simple-effects follow-up analyses. In that case, observations are ranked separately within each conditioning table (for example, separately within each level of another factor), and subgroup mean ranks are then calculated inside that restricted subset only. That slice-wise ranking logic is appropriate for simple-effects post hoc procedures, but it is not what this function reports.

Here, the response variable is always ranked globally using standard midranks, i.e., `ties.method = "average"`. Thus, tied observations receive averaged ranks in the usual way. In inferential parts of the package, significance testing additionally uses the classical tie correction to obtain appropriate p-values; this function itself is descriptive and returns the mean ranks derived from the global ranking only.

The option `force_factors = TRUE` affects only the grouping variables on the right-hand side of the formula. It forces them to be treated as categorical, even if they are stored numerically. This does *not* alter the ranking of the dependent variable and does *not* change the rank values themselves. It only ensures that the package correctly defines comparison cells and the groups within which descriptive summaries and mean ranks are averaged, instead of accidentally treating grouping variables as continuous covariates.

The function first subsets the data to complete cases on `y` and all RHS factors, then computes global ranks of `y`. For each effect (every non-empty combination of factors up to full order), it returns a row per observed cell with: `count`, `mean`, `sd`, `median`, quartiles (`q1`, `q3`), `IQR`, and `mean_rank`. The column `Effect` identifies the effect (e.g., `"A"`, `"B"`, `"A:B"`). Missing factor columns for a given effect are filled with `NA_character_` values so that all effect-specific summaries can be combined into one base-R `data.frame` without type conflicts.

Value

A base `data.frame` with columns:

- `Effect` (character),
- factor columns for all RHS factors (character, possibly `NA` in some rows),
- `count`, `mean`, `sd`, `median`, `q1`, `q3`, `IQR`, `mean_rank`.

The original call is attached as attribute `"call"`.

Examples

```
data(mimicry, package = "factorH")

# One factor
nonpar.datatable(liking ~ condition, data = mimicry)
```

```
# Two factors: rows for gender, for condition, and for gender:condition
nonpar.datatable(liking ~ gender + condition, data = mimicry)

# Three factors: all mains + 2-way and 3-way cells
nonpar.datatable(liking ~ gender + condition + age_cat, data = mimicry)
```

`normality.datatable` *Raw normality per subgroup (Shapiro–Wilk) across factor combinations*

Description

Runs Shapiro–Wilk tests on the raw response within each subgroup for all non-empty combinations of RHS factors (main effects and interaction cells).

Usage

```
normality.datatable(formula, data, force_factors = TRUE)
```

Arguments

`formula`	A model formula `y ~ A + B (+ C ...)`.
`data`	A data frame with the variables.
`force_factors`	Logical; if TRUE, coerces RHS predictors to factors.

Value

A data.frame with rows per subgroup/cell. Columns: `Effect`, factor columns, `count`, `W`, `p.shapiro` (4 decimals), `OK`.

See Also

`plan.diagnostics`

Examples

```
## Not run:
normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)

## End(Not run)
```

plan.diagnostics *Plan-level diagnostics for ANOVA/rank-based workflows*

Description

Runs all assumption checks in one call: raw normality per subgroup (Shapiro-Wilk), residual normality per cell (from a full-factorial ANOVA on the specified factors), Levene/Brown-Forsythe for the full plan (median by default), and count-balance chi-square tests for all factor combinations. Prints a concise summary and returns all detailed tables in a list.

Usage

```
plan.diagnostics(formula, data, force_factors = TRUE)
```

Arguments

formula	A model formula of the form y ~ A + B (+ C ...).
data	A data frame containing the variables in the model.
force_factors	Logical; if TRUE, coerces RHS predictors to factors.

Details

Requires helper functions defined in this package: normality.datatable, residuals.cellwise.normality.datatable, levene.plan.datatable, balance.chisq.datatable. Levene's test uses **car**; if unavailable, the Levene block returns NA rows with a warning.

Value

An invisible list with:

- $summary: overall percent_ok, ok_count, total, overall, plus per-type percentages (percent_ok_normality_raw, percent_ok_residuals_cellwise, percent_ok_balance_chisq, percent_ok_levene_full_plan).
- $results: data.frames for normality_raw, residuals_cellwise_normality, levene_full_plan, balance_chisq.

See Also

normality.datatable, residuals.cellwise.normality.datatable, levene.plan.datatable, balance.chisq.datatable

Examples

```
## Not run:
diag_out <- plan.diagnostics(liking ~ gender + condition + age_cat, data = mimicry)
diag_out$summary
diag_out$results$normality_raw

## End(Not run)
```

`residuals.cellwise.normality.datatable` *Cellwise residual normality (Shapiro–Wilk) from ANOVA models*

Description

Fits, for each subset of RHS factors, a full-factorial ANOVA to the response and tests Shapiro–Wilk normality of residuals within each cell defined by those factors. Matches the classical ANOVA assumption of normal errors per cell.

Usage

```
## S3 method for class 'cellwise.normality.datatable'
residuals(formula, data, force_factors = TRUE)
```

Arguments

`formula`	A model formula `y ~ A + B (+ C ...)`.
`data`	A data frame with the variables.
`force_factors`	Logical; if TRUE, coerces RHS predictors to factors.

Value

A data.frame with rows per cell across all factor combinations. Columns include: `Effect`, factor columns (with NA for factors not in the current subset), `count`, `W`, `p.shapiro` (4 decimals), `OK`.

See Also

`normality.datatable`, `plan.diagnostics`

Examples

```
## Not run:
residuals.cellwise.normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)

## End(Not run)
```

residuals.normality.datatable *Global residual normality (Shapiro–Wilk) from ANOVA models*

Description

For each subset of RHS factors, fits a full-factorial ANOVA and runs a single Shapiro–Wilk test on the model residuals (global test per model). Use `residuals.cellwise.normality.datatable` for the stricter per-cell assumption.

Usage

```
## S3 method for class 'normality.datatable'
residuals(formula, data, force_factors = TRUE)
```

Arguments

`formula`	A model formula `y ~ A + B (+ C ...)`.
`data`	A data frame with the variables.
`force_factors`	Logical; if TRUE, coerces RHS predictors to factors.

Value

A data.frame with one row per `Effect` (A, B, A:B, ...), with `count`, `W`, `p.shapiro` (4 decimals), `OK`.

See Also

`residuals.cellwise.normality.datatable`

Examples

```
## Not run:
residuals.normality.datatable(liking ~ gender + condition + age_cat, data = mimicry)

## End(Not run)
```

srh.effsize *SRH with effect sizes for two-factor designs*

Description

Extends `rcompanion::scheirerRayHare()` by adding popular rank-based effect sizes for each SRH term: eta^2_H and epsilon^2_H, and stores the original function call.

Usage

```
srh.effsize(formula, data, clamp0 = TRUE, ...)
```

Arguments

formula	A formula of the form y ~ A + B.
data	A data.frame containing all variables in formula.
clamp0	Logical; if TRUE (default), negative eta^2_H is truncated to 0 and epsilon^2_H truncated to the interval $[0, 1]$.
...	Passed to rcompanion::scheirerRayHare().

Details

Let H be the SRH H-statistic for a given term, n the sample size used by SRH (complete cases on y and factors), and k the number of groups compared by that term (for interactions, the number of observed combinations).

Effect sizes computed:

- **Eta^2_H**: $(H - k + 1)/(n - k)$.
- **Epsilon^2_H** (KW-like): $H * (n + 1)/(n^2 - 1)$.

The original call is stored as an attribute and can be retrieved with getCall().

Value

A data.frame (classed as c("srh_with_call","anova","data.frame")) with the SRH table extended by columns: k, n, eta2H, eps2H.

Examples

```
data(mimicry, package = "factorH")
res <- srh.effsize(liking ~ gender + condition, data = mimicry)
res
getCall(res)
```

---

srh.kway — *K-way SRH on ranks with tie-corrected p-values and rank-based effect sizes*

Description

Generalizes the Scheirer–Ray–Hare (SRH) approach to *k*-factor designs by using sums of squares from a linear model on ranks, with a standard tie correction D applied to p-values. The function returns H, tie-corrected H (Hadj), p-values and rank-based effect sizes (eta2H, eps2H) for each main effect and interaction up to the full order (i.e., (A + B + ...)^k).

Usage

```
srh.kway(formula, data, clamp0 = TRUE, force_factors = TRUE, type = 2, ...)
```

Arguments

formula	A formula of the form y ~ A + B (+ C ...).
data	A data.frame with the variables in formula.
clamp0	Logical; if TRUE (default), negative eta2H is truncated to 0 and eps2H truncated to the interval $[0, 1]$.
force_factors	Logical; coerce grouping variables to factor (default TRUE).
type	Integer; the SS type to use in car::Anova. Defaults to 2 (Type II). Set type = 3 for Type III (internally uses sum-to-zero contrasts for factors in the model fit; global options are not modified).
...	Passed to stats::lm() if applicable.

Details

Ranks are computed globally on y with ties.method = "average". Sums of squares are obtained from car::Anova() on the rank model R ~ (A + B + ...)^k. Tie correction:

$$D = 1 - \frac{\sum(t^3 - t)}{n^3 - n},$$

where t are tie block sizes and n is the number of complete cases. We report Hadj = H / D and $p = P(\chi^2_{df} \geq Hadj)$.

Rank-based effect sizes are computed from the *uncorrected* H (classical SRH convention): eta2H = (H - k + 1) / (n - k) and eps2H = H * (n + 1) / (n^2 - 1), where k is the number of non-empty groups compared by the term.

For type = 3, the model is fitted with sum-to-zero contrasts (stats::contr.sum) for RHS factors having at least 2 levels, so that Type III tests have the standard interpretation. Global contrast options are not altered.

Value

A data.frame with class c("srh_kway","anova","data.frame") containing columns: Effect, Df, Sum Sq, H, Hadj, p.chisq, k, n, eta2H, eps2H. The original call is attached as an attribute and can be retrieved with getCall().

See Also

Anova

Examples

```
## Not run:
data(mimicry, package = "factorH")
# One factor (KW-style check)
srh.kway(liking ~ condition, data = mimicry)
```

```
# Two factors (Type II by default)
srh.kway(liking ~ gender + condition, data = mimicry)

# Three factors
srh.kway(liking ~ gender + condition + age_cat, data = mimicry)

# Type III SS (with sum-to-zero contrasts set locally)
srh.kway(liking ~ gender + condition, data = mimicry, type = 3)

## End(Not run)
```

`srh.kway.full` *Full pipeline: rank-based k-way ANOVA + descriptives + post hocs*

Description

Runs a complete nonparametric, rank-based workflow for factorial designs: (1) SRH-style ANOVA table, (2) compact descriptive stats with global ranks, (3) Dunn-Bonferroni post hoc matrices for all effects, and (4) simple-effects post hocs.

Usage

```
srh.kway.full(
  formula,
  data,
  max_levels = 30,
  type = 2,
  scope = c("within", "global")
)
```

Arguments

`formula`	A formula `y ~ A (+ B + ...)`.
`data`	A `data.frame` with variables present in `formula`.
`max_levels`	Safety cap for number of levels per factor (default 30).
`type`	Sums-of-squares type for designs routed through `srh.kway()`; must be `2` or `3`. Default is `2`.
`scope`	`"within"` (default) applies Bonferroni adjustment within each simple-effects table; `"global"` applies one Bonferroni adjustment across all pairwise tests within each table.

Details

Choice of the ANOVA engine:

- 1 factor: `srh.kway()` (KW-like),
- 2 factors: `srh.effsize()` when `type = 2`,
- 2+ factors: `srh.kway()` when `type = 3`,
- 3+ factors: `srh.kway()` (general k-way on ranks).

For designs handled by `srh.kway()` (1 factor and 3+ factors), the `type` argument controls sums of squares passed to `car::Anova()` (`type = 2` or `type = 3`). For 2-factor designs, `type = 2` keeps the original SRH-style pipeline via `srh.effsize()`, whereas `type = 3` routes the analysis through `srh.kway()` / `car::Anova()`.

The `scope` argument controls Bonferroni adjustment in `srh.simple.posthocs()` and is propagated down to `srh.simple.posthoc()`. By default, `scope = "within"` applies Bonferroni adjustment within each simple-effects table; `"global"` applies a single Bonferroni adjustment across all tests within each table.

The function performs basic design diagnostics before fitting the model. Fatal input problems stop the analysis. Potentially problematic but still analyzable designs (e.g., empty cells or very small cell sizes) are not stopped; instead, warnings are stored in `$meta$warnings`.

Value

A list with elements:

- `anova` – ANOVA-like table,
- `summary` – descriptive stats data.frame,
- `posthoc_cells` – list of adjusted p-value matrices for all effects (from `srh.posthocs`), or a string when failed,
- `posthoc_simple` – list of simple-effect tables (from `srh.simple.posthocs`); for 1 factor: `"[not applicable]"`,
- `meta` – list with call, n, factor levels, design diagnostics, selected `scope`, and warnings.

Components that cannot be computed for the given design are returned as the string `"[not applicable]"`; failures are reported as `"[failed] <message>"`.

Examples

```
data(mimicry, package = "factorH")

# 1 factor
f1 <- srh.kway.full(liking ~ condition, data = mimicry)

# 2 factors
f2 <- srh.kway.full(liking ~ gender + condition, data = mimicry)

# 2 factors with global simple-effects correction
f2g <- srh.kway.full(liking ~ gender + condition, data = mimicry,
                     scope = "global")
```

```
# 3 factors
f3 <- srh.kway.full(liking ~ gender + condition + age_cat, data = mimicry)

# 3 factors with Type III SS
f3_t3 <- srh.kway.full(liking ~ gender + condition + age_cat,
                       data = mimicry, type = 3)
```

`srh.posthoc` *Dunn post hoc in a symmetric matrix form (one specified effect)*

Description

Computes Dunn's rank-based pairwise comparisons for the effect implied by `formula` and returns symmetric matrices for Z, unadjusted p-values, and adjusted p-values. Cells on one triangle (or both) can be blanked for compact reporting. For multi-factor RHS, factors are combined into a single grouping via `interaction()` (e.g., `"A:B"` cells).

Usage

```
srh.posthoc(
  formula,
  data,
  method = "bonferroni",
  digits = 3,
  triangular = c("lower", "upper", "full"),
  numeric = FALSE,
  force_factors = TRUE,
  sep = "."
)
```

Arguments

`formula`	A formula of the form `y ~ factor` or `y ~ A + B` (the latter is treated as *one* combined grouping via interaction).
`data`	A `data.frame` containing variables in `formula`.
`method`	P-value adjustment method passed to `FSA::dunnTest()`. Default `"bonferroni"`. See `p.adjust.methods` for options.
`digits`	Number of digits for rounding in the returned matrices when `numeric = FALSE`. Default `3`.
`triangular`	Which triangle to show (`"lower"`, `"upper"`, or `"full"`). Default `"lower"`.
`numeric`	Logical; if `TRUE`, return numeric matrices/data frames with `NA` on the masked triangle/diagonal. If `FALSE` (default), return character data frames with masked cells as empty strings.
`force_factors`	Logical; coerce grouping variables to `factor` (default `TRUE`).
`sep`	Separator used in `interaction()` when combining factors. Default `"."`.

Details

The function subsets to complete cases on `y` and RHS factors, optionally coerces factors, builds a single grouping variable (`._grp`) and calls `FSA::dunnTest(y ~ ._grp, data = ..., method = ...)`. The pairwise results are placed into symmetric matrices `Z`, `P.unadj`, and `P.adj`. By default only the lower triangle (excluding diagonal) is shown for compactness.

Value

A list with three `data.frames`:

- `Z` – Z statistics,
- `P.unadj` – unadjusted p-values,
- `P.adj` – adjusted p-values (per `method`).

The original call is attached as attribute `"call"`.

Examples

```
data(mimicry, package = "factorH")

# One factor
ph1 <- srh.posthoc(liking ~ condition, data = mimicry)
ph1$`P.adj`    # gotowa macierz p po korekcji

# Two factors combined (all A:B cells vs all A:B cells)
ph2 <- srh.posthoc(liking ~ gender + condition, data = mimicry)
ph2$`P.adj`

# Upper triangle, numeric frames
ph3 <- srh.posthoc(liking ~ condition, data = mimicry,
                   triangular = "upper", numeric = TRUE)
ph3$Z
```

---

srh.posthocs *Dunn post hoc tables (p.adj only) for all effects in a factorial design*

Description

For a given `y ~ A (+ B + ...)` formula, runs `srh.posthoc` for every main effect and interaction implied by the RHS (all non-empty combinations of factors) and returns a named list of adjusted p-value matrices (`P.adj`) for each effect.

Usage

```
srh.posthocs(
  formula,
  data,
  method = "bonferroni",
  digits = 3,
  triangular = c("lower", "upper", "full"),
  numeric = FALSE,
  force_factors = TRUE,
  sep = "."
)
```

Arguments

formula	A formula of the form y ~ A (+ B + ...).
data	A data.frame containing variables in formula.
method	P-value adjustment method passed to FSA::dunnTest() via srh.posthoc. Default "bonferroni".
digits	Rounding used inside srh.posthoc when numeric = FALSE. Default 3.
triangular	Which triangle to show in each matrix ("lower", "upper", "full"). Default "lower".
numeric	Logical; if TRUE, return numeric data frames with NAs on the masked triangle/diagonal; if FALSE (default), return character data frames with masked cells as empty strings.
force_factors	Logical; coerce grouping variables to factor before analysis (default TRUE).
sep	Separator for combined factor labels when needed (passed through to srh.posthoc). Default ".".

Details

The function enumerates all non-empty subsets of RHS factors (mains, 2-way, ..., k-way) and calls srh.posthoc on each corresponding sub-formula. If a subset has fewer than 2 observed levels (e.g., due to missing data after subsetting to complete cases), that effect is skipped.

Value

A named list where each element is a data.frame of adjusted p-values (P.adj) for an effect. Names use "A", "B", "A:B", ..., matching the effect structure. The original call is attached as attribute "call".

Examples

```
data(mimicry, package = "factorH")

# Two-factor design: p.adj for 'gender', 'condition', and 'gender:condition'
L2 <- srh.posthocs(liking ~ gender + condition, data = mimicry)
names(L2)
```

```
L2$gender
L2$condition
L2$`gender:condition`

# Three-factor design: includes mains, all 2-ways, and the 3-way effect
L3 <- srh.posthocs(liking ~ gender + condition + age_cat, data = mimicry)
names(L3)
```

`srh.simple.posthoc` *Simple-effects post hoc (Dunn) with Bonferroni adjustment*

Description

Computes Dunn's pairwise comparisons for simple effects of one target factor (`compare`) within levels of the remaining conditioning factors (`by`). Adjustment can be done within each conditioning table (SPSS-like) or globally across all tests.

Usage

```
srh.simple.posthoc(
  formula,
  data,
  compare = NULL,
  scope = c("within", "global"),
  digits = 3
)
```

Arguments

`formula`	A formula of the form `y ~ A + B (+ C ...)`; requires at least two RHS factors to define a simple effect.
`data`	A `data.frame` containing variables in `formula`.
`compare`	Character; the factor to compare pairwise. By default, the first factor on the RHS of `formula`.
`scope`	`"within"` (default) applies Bonferroni adjustment within each by-table; `"global"` applies one Bonferroni adjustment across all pairwise tests produced for all by-tables combined.
`digits`	Number of digits for rounding numeric columns (`Z`, `P.unadj`, `P.adj`). Default `3`.

Details

The data are subset to complete cases on `y` and all RHS factors. All RHS variables are coerced to `factor`. The table is split by all factors except `compare` and Dunn's test (`FSA::dunnTest`) is run per split. With `scope = "within"`, the Bonferroni correction is applied separately in each split (with `m.tests = choose(k, 2)` for that split). With `scope = "global"`, `P.adj` is re-computed once with `stats::p.adjust(..., method = "bonferroni")` across all pairwise tests from all splits, and `m.tests` is set to the total number of tests.

Value

A `data.frame` with columns:

- conditioning factor columns (one value repeated per split),
- `Comparison`, `Z`, `P.unadj`, `P.adj`,
- `m.tests` (number of tests used for Bonferroni),
- `adj.note` (human-readable note).

Attributes: `"adjustment"` (one-line description) and `"call"`.

Examples

```
data(mimicry, package = "factorH")

# Two factors: pairwise comparisons for 'gender' within levels of 'condition'
tab1 <- srh.simple.posthoc(liking ~ gender + condition, data = mimicry)
head(tab1)
attr(tab1, "adjustment")

# One global family of tests (global Bonferroni across all subgroup tests)
tab2 <- srh.simple.posthoc(liking ~ gender + condition, data = mimicry,
                           scope = "global")
head(tab2)
attr(tab2, "adjustment")

# Three factors: compare 'gender' within each condition x age_cat cell
tab3 <- srh.simple.posthoc(liking ~ gender + condition + age_cat, data = mimicry)
head(tab3)

# Choose a different target factor to compare
tabA <- srh.simple.posthoc(liking ~ gender + condition + age_cat, data = mimicry,
                           compare = "condition")
head(tabA)
```

`srh.simple.posthocs` *Simple-effects post hoc tables for all possible effects*

Description

For a formula `y ~ A + B (+ C ...)`, enumerates all simple-effect setups of the form `COMPARE(target) | BY(other factors)` and runs `srh.simple.posthoc` for each. Returns a named list of data frames (one per simple-effect configuration).

Usage

```
srh.simple.posthocs(formula, data, scope = c("within", "global"))
```

Arguments

formula	A formula y ~ A + B (+ C ...) with at least two RHS factors.
data	A data.frame containing the variables in formula.
scope	"within" (default) applies Bonferroni adjustment within each by-table; "global" applies one Bonferroni adjustment across all tests produced in each simple-effects table.

Details

For each choice of the comparison factor target from the RHS, all non-empty combinations of the remaining factors are treated as conditioning sets BY. For each pair (target, BY) the function calls srh.simple.posthoc with compare = target and the chosen scope. Effects where the conditioning subset has fewer than 2 observed levels of target are skipped; messages are collected in attribute "skipped".

Labels use ASCII: "COMPARE(A) | BY(B x C)".

Value

A named list of data.frames. Each element contains the columns produced by srh.simple.posthoc (e.g., Comparison, Z, P.unadj, P.adj, m.tests, adj.note). Attributes: "call" and (optionally) "skipped" with messages.

Examples

```
data(mimicry, package = "factorH")

# All simple-effect tables for a 2-factor design
tabs2 <- srh.simple.posthocs(liking ~ gender + condition, data = mimicry)
names(tabs2)

# Global Bonferroni propagated to each simple-effect table
tabs2g <- srh.simple.posthocs(liking ~ gender + condition, data = mimicry,
                              scope = "global")
names(tabs2g)

# Three factors: all COMPARE(target) | BY(conditioning) combinations
tabs3 <- srh.simple.posthocs(liking ~ gender + condition + age_cat, data = mimicry)
names(tabs3)
attr(tabs3, "skipped")
```

---

write.srh.kway.full.tsv

Write full SRH pipeline result to a TSV file

Description

Exports the result of `srh.kway.full` into a single, tab-separated text file, in the order: *ANOVA* > *SUMMARY* > *POSTHOC CELLS* > *SIMPLE EFFECTS* > *META*. Supports choosing the decimal mark for numeric values.

Usage

```
write.srh.kway.full.tsv(
  obj,
  file = "srh_kway_full.tsv",
  sep = "\t",
  na = "",
  dec = "."
)
```

Arguments

`obj`	A list produced by `srh.kway.full`.
`file`	Path to the output TSV file. Default `"srh_kway_full.tsv"`.
`sep`	Field separator (default tab `"\t"`).
`na`	String to use for missing values (default empty string).
`dec`	Decimal mark for numbers: dot `"."` (default) or comma `","`.

Details

Each section is preceded by a header line (e.g., `## SRH: EFFECTS TABLE`). For post hoc sections, each effect/table is prefixed with a subheader (e.g., `### posthoc_cells: gender:condition`). For simple-effect tables, the attribute `"adjustment"` (if present) is written as a comment line beginning with `"# "`.

Components that are not applicable (e.g., simple effects in 1-factor designs) or failed computations are written as literal one-line messages.

Value

(Invisibly) the normalized path to `file`.

Examples

```
data(mimicry, package = "factorH")
res <- srh.kway.full(liking ~ gender + condition, data = mimicry)

# Write to a temporary file (CRAN-safe)
f <- tempfile(fileext = ".tsv")
write.srh.kway.full.tsv(res, file = f, dec = ".")
file.exists(f)
```

Index