

Package: kstools (via r-universe)

October 25, 2026

Type Package

Title Clinical and Data Science Utilities

Version 0.6.1

Description Utility functions for clinical trials and data science:
descriptive statistics (summaries with optional custom stats,
quartiles, median CI, geometric mean), SDTM/ADaM helpers (date
conversion, supplemental merge, XPT batch load), table
builders, batch data loading, and utilities (z-score, CV, SWOE,
AUC, EAIR, Newcombe CIs, AR(1) simulation).

License GPL-3

URL <https://crow16384.github.io/kstools/>,
<https://github.com/crow16384/kstools>

BugReports <https://github.com/crow16384/kstools/issues>

Encoding UTF-8

Depends R (>= 4.4.0)

Imports boot, cli, dplyr, knitr, ksformat, lubridate, magrittr, MASS,
methods, nlme, openxlsx2, parallel, purrr, Rcpp, rlang,
rootSolve, tibble, tidyr

LinkingTo Rcpp

SystemRequirements C++23

Suggests arrow, energy, foreign, haven, mnormt, mcpr, pkgdown, quarto,
rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr, quarto

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libicu-dev

Repository <https://crow16384.r-universe.dev>

Date/Publication 2026-10-25 14:06:54 UTC

RemoteUrl <https://github.com/crow16384/kstools>

RemoteRef HEAD

RemoteSha f0d76736a04b8bd2ee5edecc04b77210f3eefcf1

Contents

<code>ks_aligndec</code>	3
<code>ks_auc</code>	3
<code>ks_batch_load</code>	5
<code>ks_batch_load_from_single_xpt</code>	5
<code>ks_batch_load_xlsx</code>	7
<code>ks_be_2x2</code>	8
<code>ks_boot_ci</code>	9
<code>ks_code_to_fmt</code>	10
<code>ks_compare_na</code>	11
<code>ks_current_filename</code>	12
<code>ks_cv</code>	13
<code>ks_del_tmp</code>	13
<code>ks_desc_coded</code>	14
<code>ks_dtc2dtn</code>	16
<code>ks_dunnett_crit</code>	16
<code>ks_eairdiff_mn</code>	17
<code>ks_eairdiff_strata_mn</code>	18
<code>ks_extract_single_xpt</code>	19
<code>ks_geom_mean</code>	20
<code>ks_get_data</code>	21
<code>ks_get_labels</code>	23
<code>ks_get_numdec</code>	24
<code>ks_hosmer_lemeshow</code>	25
<code>ks_impute_dtc</code>	26
<code>ks_median_ci</code>	29
<code>ks_mianalyze</code>	30
<code>ks_mv_normality</code>	32
<code>ks_n_pct</code>	33
<code>ks_newcombe_hs</code>	35
<code>ks_newcombe_hs_strata</code>	36
<code>ks_p_norm</code>	38
<code>ks_q1</code>	38
<code>ks_q3</code>	39
<code>ks_read_xpt</code>	40
<code>ks_roc</code>	41
<code>ks_set_formats</code>	42
<code>ks_set_labels</code>	42
<code>ks_simulate_ar1</code>	43
<code>ks_sum_na</code>	44
<code>ks_summary</code>	45
<code>ks_supp_consistency_check</code>	46
<code>ks_supp_merge</code>	48
<code>ks_swap_cols</code>	49
<code>ks_swoe</code>	50
<code>ks_vswitch</code>	51
<code>ks_williams</code>	52

Arguments

data	A data frame or tibble.
by	Character vector of grouping column names, e.g. <code>c("usubjid", "trt")</code> . NULL (default) computes a single AUC across the whole dataset.
val	Unquoted name of the concentration (value) column.
time	Unquoted name of the time column (numeric).
type	AUC method: "trapez" (linear, default) or "log-trapez" (log-linear).

Details

Linear trapezoid (type = "trapez"):

$$AUC_i = \frac{C_i + C_{i+1}}{2} \cdot (t_{i+1} - t_i)$$

Log-linear trapezoid (type = "log-trapez"):

$$AUC_i = \frac{C_{i+1} - C_i}{\ln C_{i+1} - \ln C_i} \cdot (t_{i+1} - t_i)$$

The log-linear method is preferred during elimination phases where concentrations decline exponentially. It requires strictly positive concentrations; when two consecutive values are equal or one is ≤ 0 the interval automatically falls back to the linear rule.

Rows are sorted by `c(by, time)` before computing so the caller does not need to pre-sort the data.

Value

- When `by = NULL`: a single numeric value.
- When `by` is provided: a `data.frame` with one row per group and an AUC column appended to the grouping columns.

See Also

[ks_auc](#) for the vectorised helper

Examples

```
## Single subject - linear trapezoid
pk1 <- data.frame(conc = c(0, 4, 2, 1), time = c(0, 1, 2, 4))
ks_auc(pk1, val = conc, time = time)

## Two subjects - grouped
pk2 <- data.frame(
  id   = c("A", "A", "A", "B", "B", "B"),
  conc = c(0, 4, 1, 0, 6, 2),
  time = c(0, 1, 2, 0, 1, 2)
)
ks_auc(pk2, by = "id", val = conc, time = time)
```

ks_batch_load

*Batch load files from a directory or ZIP***Description**

Loads all files of one type from a directory (or .zip) in batch. Supported types are csv, xlsx, xpt, and rds. Results can be returned as a named list, assigned to .GlobalEnv, saved as individual .rds, and/or saved as one .RData.

Usage

```
ks_batch_load(
  data,
  type = c("csv", "xlsx", "xpt", "rds"),
  save.path = NULL,
  keep = FALSE,
  save.rdata = NULL,
  with_length = FALSE,
  recursive = FALSE
)
```

Arguments

data	Path to a directory or a .zip file.
type	One of "csv", "xlsx", "xpt", "rds".
save.path	Optional directory where loaded objects are saved as .rds.
keep	If TRUE, assign loaded objects into .GlobalEnv.
save.rdata	Optional path to a .RData file.
with_length	Passed to foreign::read.csv3 for CSV loading.
recursive	If TRUE, search recursively under data.

Value

Named list of loaded objects.

ks_batch_load_from_single_xpt

*Load or save all datasets from a multi-member SAS XPORT v8/v9 file***Description**

Iterates all members found by [ks_xpt_content](#) and either assigns them into .GlobalEnv or writes them to disk.

Usage

```
ks_batch_load_from_single_xpt(
  file,
  action = "load",
  save_type = NULL,
  save_path = NULL
)
```

Arguments

file	Path to the XPORT file.
action	"load" (assign to .GlobalEnv) or "save" (write to disk).
save_type	For action = "save": one of "rdata", "rds", or "parquet".
save_path	For action = "save": • "rdata": path <i>without</i> extension (appended automatically). • "rds" / "parquet": path to an existing <i>directory</i>.

Details

action = "load" Each dataset is assigned into .GlobalEnv using its lower-case member name.

action = "save", save_type = "rdata" All datasets are saved together into a single .RData file at save_path (the .RData extension is appended automatically). The global environment is *not* modified.

action = "save", save_type = "rds" Each dataset is saved as a separate .rds file inside the save_path directory.

action = "save", save_type = "parquet" Each dataset is saved as a separate Parquet file (requires the **arrow** package).

Members are extracted in parallel on non-Windows platforms ([mclapply](#)); [lapply](#) is used on Windows.

Value

Invisibly returns a named list of all extracted datasets.

See Also

[ks_xpt_content](#), [ks_extract_single_xpt](#)

Examples

```
## Not run:
## Load all members into .GlobalEnv
ks_batch_load_from_single_xpt("study.xpt", action = "load")

## Save each member as .rds
ks_batch_load_from_single_xpt("study.xpt",
                             action = "save",
```

```
        save_type = "rds",
        save_path = "data/")

## End(Not run)
```

ks_batch_load_xlsx	<i>Backwards-compatible wrapper for ks_batch_load with type = "xlsx"</i>
--------------------	--

Description

Calls [ks_batch_load](#) with type = "xlsx". Provided for compatibility with older scripts that used the xlsx-specific function name.

Usage

```
ks_batch_load_xlsx(
  data,
  save.path = NULL,
  keep = FALSE,
  save.rdata = NULL,
  recursive = FALSE
)
```

Arguments

data	Path to a directory or a .zip file.
save.path	Optional directory where loaded objects are saved as .rds.
keep	If TRUE, assign loaded objects into .GlobalEnv.
save.rdata	Optional path to a .RData file.
recursive	If TRUE, search recursively under data.

Value

Named list of loaded objects (see [ks_batch_load](#)).

See Also

[ks_batch_load](#)

ks_be_2x2

*Bioequivalence: 2x2 crossover analysis***Description**

Fits the standard two-period, two-sequence crossover model ‘response ~ sequence + treatment + period’ (subject nested in sequence in the random term, ‘random = ~ seq | subject’) and returns the test-minus-reference treatment difference with its confidence interval. The sequence main effect absorbs carryover. This is the R twin of the SAS ‘PROC MIXED’ bioequivalence analysis in the R tricks article (verified: estimate 0.0145, SE 0.0142, df 7, matching SAS ‘CL=DF=7’).

Usage

```
ks_be_2x2(
  data,
  response,
  trt,
  test,
  reference,
  seq,
  period,
  subject,
  log = TRUE,
  conf.level = 0.9
)
```

Arguments

data	Data frame with one row per subject-period.
response	Unquoted name of the numeric response column.
trt	Unquoted name of the treatment column (factor with exactly the two levels named in ‘test’ and ‘reference’).
test, reference	Character labels of the test and reference treatment levels (e.g. “T”, “R”).
seq, period, subject	Unquoted names of sequence, period, and subject columns.
log	Apply ‘log()’ to the response before fitting (default ‘TRUE’, standard for BE on concentration data). Requires all responses positive.
conf.level	Confidence level; BE conventionally uses ‘0.90’.

Value

A list with ‘estimate’, ‘se’, ‘df’, ‘t’, ‘p.value’, ‘lcl’, ‘ucl’ (log scale when ‘log = TRUE’), plus ‘ratio’, ‘ratio.lcl’, ‘ratio.ucl’ on the original scale when ‘log = TRUE’.

Examples

```
cross <- data.frame(
  per = as.factor(rep(c(1, 2), 9)),
  trt = as.factor(c(rep(c("T", "R"), 4), rep(c("R", "T"), 5))),
  seq = as.factor(c(rep(1, 8), rep(2, 10))),
  cow = as.factor(rep(1:9, each = 2)),
  my = c(29.9, 27.8, 54.0, 49.7, 41.6, 38.4, 28.5, 26.5, 22.2,
          21.4, 55.5, 49.1, 43.5, 41.3, 33.2, 34.3, 18.2, 17.1))
fit <- ks_be_2x2(cross, response = my, trt = trt, test = "T",
                 reference = "R", seq = seq, period = per, subject = cow)
c(TvR_log = fit$estimate, ratio = fit$ratio,
  lcl = fit$ratio.lcl, ucl = fit$ratio.ucl)
```

ks_boot_ci

Bootstrap confidence interval for a scalar statistic

Description

Convenience wrapper around **boot**: resamples, computes a user statistic, and returns a percentile (BCa when defined) confidence interval plus the point estimate — one line instead of the usual ‘boot()’ / ‘boot.ci()’ plumbing in report code.

Usage

```
ks_boot_ci(
  x,
  statistic,
  R = 2000L,
  conf = 0.95,
  seed = NULL,
  type = c("perc", "basic", "norm", "bca")
)
```

Arguments

x	Data resampled by row: a vector, matrix, or data frame.
statistic	Function with signature ‘function(data, indices)’ returning a **single numeric value** (scalar). ‘indices’ select rows of ‘x’.
R	Number of bootstrap replicates (default ‘2000’).
conf	Confidence level (default ‘0.95’).
seed	Integer seed for reproducibility (default ‘NULL’).
type	Interval type: “perc” (percentile, default), “basic”, “norm”, or “bca”.

Details

The statistic must be scalar-valued; `ks_boot_ci()` checks the shape of the replicate matrix and aborts otherwise. BCa can be undefined (e.g. degenerate statistics on small samples) — the function then falls back to the percentile interval with a warning.

Value

A list with `'estimate'`, `'std.error'`, `'lcl'`, `'ucl'`, `'type'`, `'conf.level'`, the full `'boot.ci'` list in `'$ci'`, and the raw `'boot'` object in `'$replicates'`.

Examples

```
set.seed(1)
x <- rnorm(50, 5, 2)
ks_boot_ci(x, function(d, i) mean(d[i]))

# ratio of two columns (paired resampling)
d <- data.frame(a = rnorm(60, 5, 1), b = rnorm(60, 3, 1))
ks_boot_ci(d, function(dd, i) mean(dd$a[i]) / mean(dd$b[i]), R = 999,
           seed = 42)
```

ks_code_to_fmt

Build a named format vector from a coded column

Description

Reads a column that contains "code=label" entries, splits them on =, and passes the resulting lookup to `ksformat::fnew()` as a numeric format.

Usage

```
ks_code_to_fmt(dt, col, name = NULL)
```

Arguments

dt	A data frame containing the coded column.
col	<tidy-select> Bare name of the column with "code=label" values.
name	Character scalar. Name to register the format under via <code>ksformat::fnew()</code> . Defaults to " <code><dt_name>_<col_name>_fmt</code> ".

Value

Invisibly returns the result of `ksformat::fnew()` (called for its side-effect of registering the format).

Examples

```
## Not run:
# dt has a column DECODE with entries like "1=Male", "2=Female"
ks_code_to_fmt(dm, DECODE)

## End(Not run)
```

ks_compare_na	<i>NA-aware element-wise comparison</i>
---------------	---

Description

Element-wise ‘==’ returns ‘NA’ whenever either side is missing, which makes it unusable for comparing vectors that legitimately contain ‘NA’. ‘ks_compare_na()’ treats ‘NA’ as an ordinary value, so two missing entries at the same position are considered ****equal**** — the R twin of SAS’s ‘NULL=’ comparison operators.

Usage

```
ks_compare_na(x, y)
```

Arguments

x, y	Vectors compared element by element; the shorter is recycled. Works for numeric, character, and factor vectors (levels need not match).
------	---

Value

A logical vector with the common recycled length, never ‘NA’.

See Also

[ks_sum_na](#) for NA-aware arithmetic.

Examples

```
ks_compare_na(c(NA, "A", "B"), c(NA, "A", "C"))
ks_compare_na(c(1, NA, 3), c(1, 2, NA))
```

ks_current_filename	<i>Get the filename of the currently executing R script</i>
---------------------	---

Description

Determines the filename of the currently executing R script by checking two sources in order:

Usage

```
ks_current_filename()
```

Details

1. **source() call** — inspects the call stack for the closest `source()` frame and reads its filename environment variable.
2. **CLI argument** — inspects the command-line arguments for a `-f` or `--file=` flag (as used by `Rscript foo.R` or `R --slave -f foo.R`).

Only the [basename](#) of the resolved path is returned.

Value

A character string with the base name of the current script. Returns an empty string ("") if the filename cannot be determined from either source (e.g. when running interactively in the R console).

See Also

[basename](#), [source](#), [commandArgs](#)

Examples

```
## Not run:
# Inside a sourced file or a script run with Rscript:
ks_current_filename()
#> "my_script.R"

# In an interactive session where no script name is available:
ks_current_filename()
#> ""

## End(Not run)
```

ks_cv	<i>Coefficient of variation</i>
-------	---------------------------------

Description

Calculates a coefficient of variation for given vector

Usage

```
ks_cv(x, na.rm = TRUE)
```

Arguments

x	Numeric vector
na.rm	Flag for ignoring NAs. Default = TRUE

Examples

```
ks_cv(c(1,2,3,NA,5,NA,7,2,3,2))
```

ks_del_tmp	<i>Remove temporary objects from global environment</i>
------------	---

Description

Removes objects whose names start with t. or o. (if o=TRUE), or only t. (if o=FALSE).

Usage

```
ks_del_tmp(o = TRUE)
```

Arguments

o	If TRUE, remove names matching <code>^[to]\.</code> ; if FALSE, only <code>^t\.</code>
---	--

ks_desc_coded

*Summarise hierarchically coded clinical items***Description**

Produces a tidy long-format count summary for data coded with a two-level hierarchy, such as MedDRA System Organ Class / Preferred Term (adverse events, medical history) or ATC group / drug (medications).

The function counts distinct subjects (n) and total events (e) at three levels:

1. **Overall total** — subjects with at least one record.
2. **Level-1 total** (e.g. SOC) — subjects in that top-level group.
3. **Level-2 detail** (e.g. PT) — subjects with that specific lower-level code.

The function returns raw counts only; apply `ks_n_pct` to the n column afterwards to produce formatted "n (pct%)" strings. Rows where <12> is NA are header / SOC-total rows (i.e. they would be rendered bold in a report).

The function is intentionally single-arm: call it once per analysis population or treatment arm and join the results on the <11>, <12>, ord0, and ord key columns.

Usage

```
ks_desc_coded(
  data,
  id = usubjid,
  l1 = meddra_soc,
  l2 = meddra_pt,
  tot_label = "Total subjects"
)
```

Arguments

data	A data frame or tibble containing the coded event records (one row per event, not one row per subject).
id	<[‘data-masking’][dplyr::dplyr_data_masking]> Bare name of the column holding the unique subject identifier.
l1	<[‘data-masking’][dplyr::dplyr_data_masking]> Bare name of the top-level code column (e.g. meddra_soc).
l2	<[‘data-masking’][dplyr::dplyr_data_masking]> Bare name of the lower-level code column (e.g. meddra_pt).
tot_label	Character string used as the label in the <11> column for the overall-total row. Default: "Total subjects".

Value

A [tibble](#) with one row per display line, ordered by top-level code then lower-level code:

ord0, ord Integer sort keys. ord0 = 1 / ord = 1: overall total; ord0 = 2 / ord = 2: level-1 (SOC) total; ord0 = 2 / ord = 3: level-2 (PT) detail row.

<l1 name> Top-level code value. Contains tot_label for the overall-total row; the SOC name for all other rows.

<l2 name> Lower-level code value. NA for overall-total and level-1-total rows (these are the bold / header rows); PT name for detail rows.

n Distinct subject count for the group.

e Event count (number of source rows for the group).

See Also

[ks_n_pct](#) for formatting n as "n (pct%)" once the population size is known.

Examples

```
ae <- data.frame(
  usubjid = c("01", "01", "02", "03", "03", "04"),
  soc = c("Cardiac", "Cardiac", "Nervous", "Cardiac", "Nervous", "Nervous"),
  pt = c("Palp", "Tachy", "Head", "Palp", "Dizzy", "Head"),
  stringsAsFactors = FALSE
)

## Raw counts (N unknown at this stage)
res <- ks_desc_coded(ae, id = usubjid, l1 = soc, l2 = pt)
res

## Add formatted value column once pop is known
dplyr::mutate(res, value = ks_n_pct(n, totals = 10))

## Multi-arm via group_modify(); pop looked up per arm
pop <- c(A = 5, B = 5)
ae$trt <- rep(c("A", "B"), each = 3)
ae |>
  dplyr::group_by(trt) |>
  dplyr::group_modify(
    ~ ks_desc_coded(.x, id = usubjid, l1 = soc, l2 = pt) |>
      dplyr::mutate(value = ks_n_pct(n, pop[.y$trt]))
  ) |>
  dplyr::ungroup()
```

ks_dtc2dtn	<i>Convert DTC (date-time character) to Date</i>
------------	--

Description

Parses ISO-like date strings (YYYY, YYYY-MM, YYYY-MM-DD) with optional NA/UN for month or day, and converts to Date. Missing month/day become 1.

Usage

```
ks_dtc2dtn(x)
```

Arguments

x Character vector of date strings (e.g. "2020-01-15", "2020-UN").

Value

A Date vector.

ks_dunnett_crit	<i>Exact Dunnett critical value (multivariate-t)</i>
-----------------	--

Description

Computes the two- or one-sided Dunnett critical value for 'k' comparisons against a common control by inverting the equicorrelated multivariate Student-t CDF ('mnormt::sadmvt' / 'mnormt::pmt' via uniroot). For balanced designs the control-correlation is 'rho = 1/2'. Numerically equivalent to 'nCDunnett::qNCDun' (2.5067 vs 2.506845 at 'p = 0.95, df = 24, k = 3').

Usage

```
ks_dunnett_crit(p, df, k, rho = 0.5, type = c("two-sided", "one-sided"))
```

Arguments

p Coverage probability (e.g. '0.95').

df Error degrees of freedom.

k Number of treatments compared to the control ('k >= 2').

rho Correlation between pairs of contrasts; '1/2' for balanced Dunnett designs.

type '"two-sided"' (default) or '"one-sided"'.

Value

A single numeric critical value.

Examples

```
ks_dunnett_crit(p = 0.95, df = 24, k = 3, rho = 0.5)
ks_dunnett_crit(p = 0.95, df = 24, k = 3, rho = 0.5, type = "one-sided")
```

ks_eairdiff_mn

Miettinen-Nurminen CI for the difference of two incidence rates

Description

Computes the difference (Test – Control) of two exposure-adjusted incidence rates (EAIRs) and its confidence interval using the score-based method of Miettinen & Nurminen (1985).

Usage

```
ks_eairdiff_mn(n1, n2, T1, T2, alpha = 0.05)
```

Arguments

n1, n2	Non-negative integer event counts for Test and Control.
T1, T2	Positive person-time at risk for Test and Control.
alpha	Significance level for the two-sided CI. Default 0.05.

Details

The CI is found by inverting the score statistic. For each candidate rate difference θ , the pooled estimate $\tilde{\lambda}_2$ is obtained by solving the MLE equations under the constraint $\lambda_1 - \lambda_2 = \theta$, and the score chi-squared is compared to $\chi_{1,1-\alpha}^2$. The two roots of this equation (found via [uniroot.all](#)) are the lower and upper confidence bounds.

The search interval for roots is fixed at $c(-1, 1)$, appropriate when incidence rates are measured in events per person-time unit and are well below 1. If rates exceed 1, expand the interval manually by calling [uniroot.all](#) directly.

The function is vectorised: when n1, n2, T1, T2 are vectors of equal length, one row per element is returned.

Value

A matrix with one row per input pair and three named columns: delta (point estimate), lcl (lower bound), ucl (upper bound). Bounds are NA when the score equation has fewer than two roots in $(-1, 1)$.

References

Miettinen O, Nurminen M (1985). Comparative analysis of two rates. *Statistics in Medicine*, 4(2), 213–226.

See Also

[ks_eairdiff_strata_mn, uniroot.all](#)

Examples

```
## Single comparison: Test rate = 0.10, Control rate = 0.05
ks_eairdiff_mn(n1 = 10, n2 = 5, T1 = 100, T2 = 100)

## Equal rates + delta = 0, CI straddles zero
ks_eairdiff_mn(n1 = 5, n2 = 5, T1 = 100, T2 = 100)

## Vectorised over multiple time-point comparisons
ks_eairdiff_mn(
  n1 = c(10, 20, 8),
  n2 = c( 5, 15, 6),
  T1 = c(100, 150, 90),
  T2 = c(100, 150, 90)
)

## 90 %% CI
ks_eairdiff_mn(n1 = 10, n2 = 5, T1 = 100, T2 = 100, alpha = 0.10)
```

ks_eairdiff_strata_mn *Stratified Miettinen-Nurminen CI for the difference of incidence rates*

Description

Stratified version of [ks_eairdiff_mn](#): pools evidence across strata using inverse-variance-like weights and returns a single CI for the common rate difference.

Usage

```
ks_eairdiff_strata_mn(n1, n2, T1, T2, alpha = 0.05)
```

Arguments

n1, n2	Numeric vectors of event counts per stratum (Test, Control).
T1, T2	Numeric vectors of person-time per stratum (Test, Control). All four vectors must have the same length.
alpha	Significance level for the two-sided CI. Default 0.05.

Details

Each stratum j contributes a weight $W_j = (\tilde{\lambda}_{1j}/(\tilde{\lambda}_{2j}T_{1j}) + 1/T_{2j})^{-1}$ and the combined score statistic is $\sum_j W_j^2 V_j^{-1} \cdot (\sum_j W_j (r_{1j} - r_{2j} - \theta))^2$.

The point estimate delta is the unweighted arithmetic mean of the stratum-specific rate differences.

Strata with NA in any of their inputs are silently dropped via `na.rm = TRUE` in the stratum sums.

Value

A named numeric vector of length 3: `delta` (point estimate), `lc1` (lower bound), `uc1` (upper bound).

References

Miettinen O, Nurminen M (1985). Comparative analysis of two rates. *Statistics in Medicine*, 4(2), 213–226.

See Also

[ks_eairdiff_mn](#)

Examples

```
## Two strata
ks_eairdiff_strata_mn(
  n1 = c(10, 5),
  n2 = c(5, 3),
  T1 = c(100, 80),
  T2 = c(100, 70)
)

## Single stratum – equivalent to ks_eairdiff_mn() up to point estimate
ks_eairdiff_strata_mn(n1 = 10, n2 = 5, T1 = 100, T2 = 100)

## 90 %% CI
ks_eairdiff_strata_mn(
  n1 = c(10, 5), n2 = c(5, 3),
  T1 = c(100, 80), T2 = c(100, 70),
  alpha = 0.10
)
```

`ks_extract_single_xpt` *Extract one dataset from a multi-member SAS XPORT v8/v9 file*

Description

Reads a single member from the file by splicing the file-level header bytes onto the member's raw bytes, then passing the combined buffer to [read_xpt](#).

Usage

```
ks_extract_single_xpt(name, file, content)
```

Arguments

<code>name</code>	Character name of the member to read (must match a name in <code>content</code> , i.e. lower-case).
<code>file</code>	Path to the XPORT file.
<code>content</code>	Named list returned by ks_xpt_content (file).

Details

A valid XPORT stream requires both the 240-byte file header (three 80-byte LIBV8 records) and the member data. This function prepends the file header so that `haven::read_xpt` can parse the member in isolation.

Value

A [tibble](#) (as returned by `read_xpt`).

See Also

[ks_xpt_content](#), [ks_batch_load_from_single_xpt](#)

Examples

```
## Not run:
content <- ks_xpt_content("study.xpt")
ae      <- ks_extract_single_xpt("ae", "study.xpt", content)

## End(Not run)
```

ks_geom_mean	<i>Geometric mean and confidence interval</i>
--------------	---

Description

Computes the geometric mean of a numeric vector and an approximate confidence interval based on the Student *t*-distribution on the **log scale** (assuming log-transformed values are approximately normal).

Usage

```
ks_geom_mean(x, na.rm = TRUE, zero.propagate = FALSE, alpha = 0.05)
```

Arguments

<code>x</code>	Numeric vector (non-negative values; see Details).
<code>na.rm</code>	Logical. If TRUE, NAs are removed first.
<code>zero.propagate</code>	Logical. If TRUE, the presence of any zero in <code>x</code> forces a geometric mean of 0 (and NA confidence limits). If FALSE, zeros are excluded from the mean (only strictly positive values are used).
<code>alpha</code>	Significance level for the two-sided CI (coverage nominal $1 - \alpha$). Must lie in $(0, 1)$.

Details

The geometric mean of positive values $x_1, \dots, x_k$ is $\exp(\frac{1}{k} \sum_{i=1}^k \log x_i)$.

If `zero.propagate = FALSE`, only $x_i > 0$ are used. If there are no positive values, the geometric mean is 0 (all zeros) or NA (empty or invalid input after removing NAs).

Negative values are not allowed: the function returns NaN for `geom.mean` (and CI components) because the geometric mean is not defined in the reals.

The confidence interval is $\exp(\bar{y} \pm t_{1-\alpha/2, df} s / \sqrt{n})$ where $y = \log x$ for the positive observations used, s is the sample SD of y , n is their count, and $df = n - 1$. For $n < 2$ the limits are NA.

Value

A list with components:

`geom.mean` Geometric mean (numeric scalar).

`geom.sd` Geometric standard deviation, $\exp(\text{sd}(\log x))$ on the positive data used (same as common PK notation; NA if not defined).

`geom.mean.lcl` Lower limit of the $100(1 - \alpha)\%$ CI.

`geom.mean.ucl` Upper limit of the $100(1 - \alpha)\%$ CI.

References

Kirkwood, T. B. (1979). Geometric means and measures of dispersion. *Biometrics*, **35**(4), 908–909.

See Also

[log](#), [quantile](#)

Examples

```
x <- abs(rnorm(100, 10, 5))
ks_geom_mean(x)
```

ks_get_data

Load domain files and return them

Description

Reads one or more data files from `path` and returns them as [tibbles](#). A single-domain call returns a tibble; a multi-domain call returns a named list of tibbles (names are the lowercased domain names).

Usage

```
ks_get_data(
  ...,
  path = getOption("path.data", "."),
  type = c("auto", "rds", "xpt", "xlsx", "csv", "csvks"),
  xpt_reader = getOption("kstools.xpt.reader", "haven")
)
```

Arguments

...	Unquoted (or quoted) domain names, e.g. dm, ae, lb.
path	Directory containing the data files. Defaults to <code>getOption("path.data", ".")</code> .
type	File type: "auto" (detect from extension), "rds", "xpt", "xlsx", "csv", or "csvks" (CSV with KeyStat comment metadata).
xpt_reader	Reader to use for .xpt files: "haven" (default) or "internal" (use ks_read_xpt). Defaults to <code>getOption("kstools.xpt.reader", "haven")</code> .

Details

Supported formats: .rds (R native), .xpt (SAS XPORT), .xlsx (Microsoft Excel), .csv (plain CSV), .csv with KeyStat comments (type = "csvks").

Domain names can be passed as bare names (dm, ae) or as quoted strings ("dm", "ae"); both forms are accepted. Lookup against the directory is **case-insensitive** for both stem and extension, so `ks_get_data(dm)` and `ks_get_data(DM)` match any of dm.xpt, DM.XPT, Dm.Xpt, etc.

When type = "auto" (default), the function scans the directory for files matching the requested domains and determines the format from the extension of the first match. All requested domains must use the same format.

The default search path is read from `getOption("path.data", ".")`. Set this option in .Rprofile to avoid repeating the path in every call:

```
options(path.data = "/my/project/data")
```

For .xpt (SAS XPORT) files, the reader is selected by the xpt_reader argument (default from option `kstools.xpt.reader`, fallback "haven"):

"haven" (default) use `haven::read_xpt()`.

"internal" use the package-internal [ks_read_xpt\(\)](#) parser. This avoids the "ignored N trailing byte(s) after row parsing" warning emitted by **haven** on some XPORT files. For multi-member XPORT files, the domain slot will contain a named list of tibbles (one per member).

When two or more files in path share the same stem but differ only in case (e.g. both dm.xpt and DM.xpt exist), the resolution strategy is controlled by the global option `kstools.get_data.case`:

"error" (default) abort with a message listing the candidates.

"first" pick the first match in alphabetical order.

"lower" require a candidate whose file stem `file_path_sans_ext` equals the domain in full lowercase (extension is always matched case-insensitively); otherwise abort.

"upper" require a candidate whose file stem equals the domain in full uppercase (extension case-insensitive); otherwise abort.

Value

For a single domain, a tibble. For two or more domains, a named list of tibbles keyed by the lowercased domain name.

See Also

[ks_batch_load](#), [read.csv3](#)

Examples

```
## Not run:
## Load a single domain
options(path.data = "/project/data")
dm <- ks_get_data(dm)

## Case-insensitive lookup: works whether folder has dm.xpt or DM.XPT
dm <- ks_get_data(DM)

## Load multiple domains: returns a named list
dat <- ks_get_data(dm, ae, lb, path = "/project/data/xpt")
head(dat$dm)

## Force a specific file type
dm <- ks_get_data(dm, path = "/project/data", type = "xpt")

## Resolve case-duplicate filenames
options(kstools.get_data.case = "lower") # prefer dm.xpt over DM.xpt
dm <- ks_get_data(dm, path = "/project/data")

## Quoted strings are also accepted
dat <- ks_get_data("dm", "ae", path = "/project/data")

## Use the internal XPT reader to avoid haven's trailing-byte warning
options(kstools.xpt.reader = "internal")
dm <- ks_get_data(dm, type = "xpt")

## Or override per call without changing the global option
dm <- ks_get_data(dm, type = "xpt", xpt_reader = "internal")

## End(Not run)
```

ks_get_labels

Get variable labels from a data frame

Description

Returns a two-column tibble with variable names and their "label" attributes (as set by **labelled**, **haven**, or manual `attr(x, "label") <- ...` calls). Columns without a label attribute are returned as `NA_character_`.

Usage

```
ks_get_labels(x)
```

Arguments

x A data frame (or tibble) whose column labels you want.

Value

A **tibble** with columns:

var Character. Column name taken from `names(x)`.

label Character. Value of the "label" attribute, or NA if absent.

Examples

```
df <- data.frame(a = 1:3, b = letters[1:3])
attr(df$a, "label") <- "Numeric variable"
ks_get_labels(df)
```

ks_get_numdec

Compute the maximum number of decimal places per group

Description

For each group defined by `by`, determines the maximum number of decimal digits observed in `var` (after rounding to 4 decimal places) and joins the result back as a new column `NUMDEC`.

Usage

```
ks_get_numdec(data, var, by)
```

Arguments

data A data frame or tibble.

var Character string naming the numeric column to inspect.

by Character vector of grouping column names.

Details

The decimal-digit count is derived by formatting the rounded value with `format(scientific = FALSE)`, stripping trailing zeros, and counting characters after the decimal point. NA values in `var` are ignored when computing the group maximum.

Value

data with an additional integer column `NUMDEC` giving the group-level maximum decimal count.

Examples

```
d <- data.frame(
  grp = c("A", "A", "B", "B"),
  val = c(1.1, 2.34, 3.0, 4.567)
)
ks_get_numdec(d, "val", "grp")
```

ks_hosmer_lemeshow	<i>Hosmer–Lemeshow goodness-of-fit test</i>
--------------------	---

Description

Groups fitted probabilities into quantile-based bins, cross-tabulates observed versus expected successes/failures, and returns the chi-squared statistic with ‘g - 2’ degrees of freedom (Hosmer & Lemeshow, 1980). A high p-value indicates no evidence of miscalibration.

Usage

```
ks_hosmer_lemeshow(y, yhat, g = 10)
```

Arguments

y	Binary response (0/1), same length as ‘yhat’.
yhat	Fitted probabilities from a logistic model.
g	Number of groups (default 10 = deciles).

Value

A list with components `chisq` and `p.value`.

References

Hosmer, D. and Lemeshow, S. (1980). Goodness of fit tests for the multiple logistic regression model. *Biometrics*, **36**, 1045–1054.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(200))
d$y <- rbinom(200, 1, plogis(0.8 * d$x))
m <- glm(y ~ x, binomial, data = d)
ks_hosmer_lemeshow(d$y, fitted(m))
```

ks_impute_dtc	<i>Impute incomplete ISO 8601 date/datetime strings</i>
---------------	---

Description

Parses truncated ISO 8601 date or datetime character strings and imputes missing components according to the chosen direction, returning a tibble with the resulting POSIXct and an imputation flag.

Usage

```
ks_impute_dtc(x, direction = "no", tz = "UTC", ref = NULL)
```

Arguments

x	Character vector of ISO 8601 truncated date/datetime strings. Accepted formats: "YYYY", "YYYY-MM", "YYYY-MM-DD", "YYYY-MM-DDTHH", "YYYY-MM-DDTHH:MM", "YYYY-MM-DDTHH:MM:SS". A space may replace T as the date-time separator. Trailing timezone designators (Z, +HH:MM, -HH:MM) are accepted but ignored; the tz argument governs the output timezone.
direction	One of "no", "start", or "end". Controls how missing components are filled. Validated via match.arg .
tz	IANA timezone string applied to the output POSIXct values. Defaults to "UTC".
ref	NULL (default), or a Date, POSIXct, or POSIXlt reference vector recycled to length(x). When provided, missing components whose year (and optionally month) match the corresponding reference are filled from that reference rather than using the directional fallback.

Details

Parsing

Each element of x is matched against a regular expression that captures year, month, day, hour, minute, and second independently. The date-time separator may be T or a single space. Trailing timezone designators (Z, +HH:MM, -HHMM) are silently stripped; all output is placed in the timezone given by tz. Elements that do not match the pattern produce dt = NA and imputation_flag = NA.

Completeness and imputation flags

A string is considered *complete* when year, month, and day are all present, and — if a time separator (T or space) is present — hour and minute are also present. Complete strings are assembled directly without imputation (flag NA), *except* for date-only strings under direction = "end" (Case D, see below).

The imputation_flag records the most significant component that was imputed:

Flag	Meaning
"M"	Month was imputed (year was the only component present).
"D"	Day was imputed (year and month were present).

"H"	Hour (and minute) were imputed.
"S"	Minute was imputed (hour was already present).
NA	No imputation was performed.

Direction

"start" Each missing component is filled with the *earliest* plausible value: month → 1, day → 1, hour → 0, minute → 0.

"end" Each missing component is filled with the *latest* plausible value: month → 12, day → last day of that month (correctly handling leap years), hour → 23, minute → 59.

"no" No imputation is performed. Incomplete strings return dt = NA and imputation_flag = NA.

Date-only strings with a directional imputation (Cases D and E)

A date-only string such as "2020-04-06" contains no time separator and is otherwise complete. Because the time was not recorded, it is always considered imputed when direction != "no":

Case D (direction = "end") Time is imputed as 23:59:00. Flag: "H".

Case E (direction = "start") Time is imputed as 00:00:00. Flag: "H".

A reference-date match (Case C) overrides both Case D and Case E.

Reference-guided imputation

When ref is provided and direction != "no", the function first checks whether the parsed year (and optionally month and day) match the corresponding reference value before applying the directional fallback. Cases are evaluated in priority order:

Case A — day missing, year + month match ref When the day (and time) are absent but year and month equal the reference, day, hour, and minute are taken from ref. Directional start/end values are *not* used. Typical flag: "D".

Case B — month missing, year matches ref When month (and therefore day and time) are absent but the year equals the reference year, month, day, hour, and minute are all taken from ref. Typical flag: "M".

Case C — date complete, time absent, full date matches ref A date-only string whose year, month, and day all equal the reference has its time filled from ref. This takes priority over the Case D / Case E directional time fallback. Flag: "H".

Fallback If none of Cases A–C applies — because ref is NULL, the year does not match, or the month does not match — the directional start/end values are used as described above.

ref is recycled to length(x) via rep_len(). A Date ref is coerced to POSIXct in timezone tz before its components are extracted.

Value

A **tibble** with one row per element of x and two columns:

dt POSIXct in timezone tz, or NA when the year is absent or imputation could not be completed.

imputation_flag character; the highest-priority component that was imputed: "M" (month), "D" (day), "H" (hour), "S" (minute). NA when no imputation was performed.

Examples

```
# --- direction = "no" -----
# Complete datetime - no imputation, flag NA
ks_impute_dtc("2020-03-15T10:30:00", "no")

# Truncated with direction = "no" - returns NA (nothing to impute)
ks_impute_dtc("2020-03", "no")

# Invalid string
ks_impute_dtc("not-a-date", "no")

# --- direction = "start" -----
# Year only → 2020-01-01 00:00:00, flag "M"
ks_impute_dtc("2020", "start")

# Year-month → first day of month, flag "D"
ks_impute_dtc("2020-03", "start")

# Date-only + end → end-of-day 23:59, flag "H" (Case D)
ks_impute_dtc("2020-03-15", "end")

# Date-only + start → midnight 00:00, flag "H" (Case E)
ks_impute_dtc("2020-03-15", "start")

# --- direction = "end" -----
# End of February in a leap year → 2020-02-29 23:59:00, flag "D"
ks_impute_dtc("2020-02", "end")

# End of March → 2020-03-31 23:59:00, flag "D"
ks_impute_dtc("2020-03", "end")

# --- ref: same-period guided imputation -----
ref <- as.POSIXct("2020-03-15 10:30:00", tz = "UTC")

# Case A - day missing, year+month match ref → take day/time from ref
ks_impute_dtc("2020-03", "start", ref = ref)

# Case C overrides D - date matches ref → take time from ref, not 23:59
ks_impute_dtc("2020-03-15", "end", ref = ref)

# ref year/month mismatch → directional fallback used instead
ks_impute_dtc("2020-04", "start", ref = ref)

# --- Vectorised with dplyr::bind_cols() -----
## Not run:
library(dplyr)
df <- tibble::tibble(
  dtc = c("2020-03-15T08:00", "2020-03-15", "2020-03", "2020", "bad"),
  ref = as.POSIXct(c(NA, "2020-03-15 14:30:00",
    "2020-03-10 08:00:00", "2020-05-01 00:00:00", NA),
    tz = "UTC")
)
```

```
dplyr::bind_cols(df, ks_impute_dtc(df$dtc, "start", ref = df$ref))

## End(Not run)
```

ks_median_ci

Confidence interval for the median

Description

Computes the sample median and a confidence interval using one of three approaches: bootstrap (**boot**), Olive's simple median interval, or a normal approximation based on the binomial distribution of ranks (the "z" method).

Usage

```
ks_median_ci(
  x,
  na.rm = TRUE,
  alpha = 0.05,
  method = c("boot", "olive", "z"),
  R = 10000L,
  seed = 0L,
  type = 2L
)
```

Arguments

x	Numeric vector.
na.rm	Logical. If TRUE, NAs are removed before computing (default TRUE).
alpha	Significance level; the interval has nominal coverage $1 - \alpha$. Must be in (0, 1).
method	Character: "boot" (default), "olive", or "z". See Details .
R	Number of bootstrap replicates when method = "boot".
seed	Integer passed to set.seed immediately before bootstrap resampling (for reproducibility). The default 0L fixes the RNG state; use another integer for a different stream.
type	Integer 1–9: quantile definition for the reported median when method = "z" (passed to quantile). Default 2 matches SAS PROC UNIVARIATE / Hyndman & Fan type 2.

Details

boot Uses [boot](#) with the sample median as the statistic and [boot.ci](#) to produce Basic, Normal, percentile, and BCa intervals (when BCa is defined).

olive Implements David Olive's simple median CI (order statistics with a *t*-based multiplier). Requires at least two non-missing observations.

z Approximate CI from the normal approximation to the binomial number of observations below the population median: bounds are the order statistics with ranks $n/2 \pm z_{1-\alpha/2} \sqrt{n}/2$ (rounded inward). The reported Median uses [quantile\(..., type\)](#).

Value

A named numeric vector. The first element is always Median. Remaining elements depend on method:

boot Basic.LCI, Basic.UCI, Normal.LCI, Normal.UCI, Percent.LCI, Percent.UCI, BCa.LCI, BCa.UCI (BCa may be NA in small samples).

olive Olive.LCI, Olive.UCI.

z z.LCI, z.UCI (limits are order statistics of the sorted data).

References

Olive, D. J. (2005). *Applied robust statistics*. Preprint. (median CI procedure.)

Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press.

See Also

[median](#), [quantile](#)

Examples

```
x <- rnorm(100, 56, 2.5)
ks_median_ci(x, method = "olive")

vv <- c(8, 11, 12, 13, 15, 17, 19, 20, 21, 21, 22, 23, 25, 26, 28)
ks_median_ci(vv, method = "boot", R = 1000, seed = 1L)
ks_median_ci(vv, method = "olive")
ks_median_ci(vv, method = "olive", alpha = 0.1)
ks_median_ci(vv, method = "z")
ks_median_ci(vv, method = "z", alpha = 0.1)
```

ks_mianalyze

Analogue of SAS PROC MIANALYZE (Rubin pooling)

Description

Pools point estimates and standard errors across multiple imputations according to Rubin's rules. Degrees of freedom follow the Barnard-Rubin (1999) small-sample adjustment when complete-data df are supplied via edf (equivalent to the SAS EDF= option).

Usage

```
ks_mianalyze(est, err, alpha = 0.05, theta0 = 0, edf = Inf, v0)
```

```
mianalyze(est, err, alpha = 0.05, theta0 = 0, edf = Inf, v0)
```

```
## S3 method for class 'ks_mianalyze'
```

```
summary(object, ...)

## S3 method for class 'ks_mianalyze'
knit_print(x, ...)
```

Arguments

<code>est</code>	Numeric vector of point estimates from each imputation.
<code>err</code>	Numeric vector of corresponding standard errors.
<code>alpha</code>	Significance level for confidence intervals. Default 0.05.
<code>theta0</code>	Null value for hypothesis test $H_0 : \theta = \theta_0$.
<code>edf</code>	Complete-data degrees of freedom (positive numeric scalar or Inf). Equivalent to the SAS EDF= option in PROC MIANALYZE. When finite, the Barnard-Rubin (1999) adjusted degrees of freedom v_m^* are used instead of the Rubin (1987) large-sample v_m . Default Inf (no adjustment, Rubin large-sample df).
<code>v0</code>	Deprecated alias for edf. Kept for backward compatibility; ignored when edf is also supplied.
<code>object</code>	A ks_mianalyze object (summary method).
<code>...</code>	Additional arguments, currently ignored.
<code>x</code>	A ks_mianalyze object (print method).

Details

Let Q_i and U_i be estimate and variance from imputation i . The pooled estimate is $\bar{Q} = m^{-1} \sum_i Q_i$; within variance $\bar{U} = m^{-1} \sum_i U_i$; between variance $B = (m-1)^{-1} \sum_i (Q_i - \bar{Q})^2$; total variance $T = \bar{U} + (1 + 1/m)B$.

Rubin's (1987) large-sample degrees of freedom are $v_m = (m-1)(1 + \bar{U}/((1 + m^{-1})B))^2$. When the complete-data degrees of freedom v_0 are small and the fraction of missing information is modest, v_m can greatly exceed v_0 , which is inappropriate. The Barnard-Rubin (1999) adjustment computes $v_m^* = (1/v_m + 1/\hat{v}_{obs})^{-1}$ where $\hat{v}_{obs} = (1 - \gamma)v_0(v_0 + 1)/(v_0 + 3)$ and $\gamma = (1 + m^{-1})B/T$.

When the within-imputation variance $\bar{U}$ is exactly zero, `df`, `r`, `lambda`, `re`, the confidence limits and the t -test are reported as NA and a warning is issued. This mirrors PROC MIANALYZE, which warns per zero standard error and prints "." for those quantities, computing no inference in that case - with or without EDF= - even when the between-imputation variance is positive. Validated against a live SAS PROC MIANALYZE run; the probe tooling is deliberately kept outside the package.

Value

An object of class `ks_mianalyze` with components:

- Estimate** Pooled estimate.
- StdErr** Pooled standard error.
- m** Number of imputations.
- Vb** Between-imputation variance.
- Vw** Within-imputation variance.

Vt Total variance.
df Degrees of freedom (adjusted when edf is finite).
r Relative increase in variance due to nonresponse.
lambda Fraction of missing information.
re Relative efficiency.
lcl, ucl Two-sided confidence limits for Estimate.
alpha Alpha used for CI.
theta0 Null value used in hypothesis test.
edf Complete-data degrees of freedom used.
minim, maxim Min/max of input est.
t Test statistic for H_0 .
pval Two-sided p-value.

References

Barnard, J. and Rubin, D. B. (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86, 948–955.
 Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.
 SAS Institute Inc. (2013). *SAS/STAT 12.3 User's Guide*. Chapter 58, The MIANALYZE Procedure.

See Also

`vignette("mianalyze", package = "kstools")` for worked examples and SAS comparisons.

Examples

```
est <- c(0.345, 0.303, 0.298, 0.378)
err <- c(0.056, 0.101, 0.099, 0.034)
out <- mianalyze(est, err)
out

## Barnard-Rubin adjusted df (equivalent to SAS EDF=120)
mianalyze(est, err, edf = 120)
```

ks_mv_normality

Multivariate normality tests (Mardia, energy)

Description

Mardia's skewness and kurtosis tests are implemented in base R (verified against 'QuantPsyc::mult.norm'); the energy test ('energy::mvnorm.etest') is attached automatically when the suggested **energy** package is installed, and silently skipped otherwise.

Usage

```
ks_mv_normality(x, R = 999, seed = NULL)
```

Arguments

x	A numeric matrix or data frame (rows = observations). 'NA' rows are dropped.
R	Number of energy-test replicates.
seed	Integer seed for the energy permutation (reproducibility).

Details

Mardia: under multivariate normality, $A = n \text{ b1_p} / 6$ is chi-squared with $p(p+1)(p+2)/6$ df, and the standardized b2_p is standard normal. The energy test is a consistent omnibus alternative; with $n < p + 2$ Mardia's test is not defined and 'NULL' is returned.

Value

A list with 'mardia' (a 2×3 matrix: Beta-hat, kappa, p-val; 'NULL' if the sample is too small), 'energy' (or 'NULL' when the **energy** package is missing), and 'n', 'p'.

Examples

```
set.seed(0)
dat <- data.frame(x1 = rnorm(50), x2 = rnorm(50), x3 = rnorm(50))
out <- ks_mv_normality(dat)
out$mardia
out$energy$statistic
```

ks_n_pct

*Format count and percentage pairs***Description**

Produces formatted "n (xx.x%)" strings from absolute counts and denominators. An S3 generic dispatching on the class of n:

- **numeric method** — formats a single vector of counts.
- **data.frame method** — formats one or more named count columns inside a data.frame or tibble simultaneously, leaving all other columns unchanged.

Usage

```
ks_n_pct(n, totals, na.rep = NA_character_)
```

Arguments

n	Numeric (integer) vector of counts, or a data.frame / tibble whose named columns contain counts.
totals	Numeric denominator(s). • <i>numeric method</i>: a scalar or a numeric vector recycled against n. • <i>data.frame method</i>: a named numeric vector whose names must match column names present in n.
na.rep	Replacement string for NA counts. • NA_character_ (default): propagate NA in the output. • Any character string, e.g. "" or "0 (\ 0.0%)": substitute missing counts with that string.

Details

Counts are converted to integer before formatting so that both integer and whole-number double vectors are accepted.

Percentages are formatted as "%5.1f%", giving one decimal place and a minimum field width of 5 characters (e.g. " 5.0%", "100.0%"). The fixed width ensures vertical alignment when columns are printed side-by-side.

A warning is issued when any element of totals is zero, as this produces Inf in the percentage.

Value

- *numeric method*: a character vector of the same length as n.
- *data.frame method*: the input data.frame with the named count columns replaced by formatted character strings; all other columns are unchanged.

Examples

```
## Numeric method - single total
ks_n_pct(c(10L, 20L, 30L), 56L)

## Numeric method - element-wise totals
ks_n_pct(c(10L, 20L, 30L), c(34L, 42L, 56L))

## NA propagation (default)
ks_n_pct(c(10L, NA_integer_, 30L), 50L)

## Replace NA with an explicit string
ks_n_pct(c(10L, NA_integer_, 30L), 50L, na.rep = "0 ( 0.0%)")

## Replace NA with an empty string
ks_n_pct(c(10L, NA_integer_, 30L), 50L, na.rep = "")

## data.frame method
dex <- data.frame(
  par      = c("A", "B", "C"),
  Therapy_A = c(11L, 23L, 14L),
```

```

    Therapy_B = c(20L, 1L, NA_integer_)
  )
  ks_n_pct(dex, c(Therapy_A = 56L, Therapy_B = 67L))

## data.frame method – keep only one column
  ks_n_pct(dex, c(Therapy_A = 56L))

```

ks_newcombe_hs

Newcombe Hybrid Score CI for a difference of two proportions

Description

Computes a confidence interval for $p_1 - p_2$ using Newcombe's Hybrid Score method (Newcombe, 1998, Method 10). Each group's proportion is first bounded by its Wilson score interval, and the two intervals are then combined to yield the CI for the difference.

Usage

```
ks_newcombe_hs(x1, n1, x2, n2, alpha = 0.05)
```

Arguments

x1	Integer vector of event counts for group 1 (≥ 0).
n1	Integer vector of sample sizes for group 1 (> 0).
x2	Integer vector of event counts for group 2 (≥ 0).
n2	Integer vector of sample sizes for group 2 (> 0).
alpha	Significance level; scalar in $(0, 1)$. Defaults to 0.05 (95% CI).

Details

Algorithm (Newcombe, 1998):

1. Compute the Wilson score CI $[l_i, u_i]$ for each group at level $1 - \alpha$:

$$l_i, u_i = \frac{2x_i + z^2 \mp z\sqrt{z^2 + 4x_i(n_i - x_i)/n_i}}{2(n_i + z^2)}$$

2. Form the CI for $\delta = p_1 - p_2$:

$$\text{LCL} = \hat{\delta} - \sqrt{(p_1 - l_1)^2 + (u_2 - p_2)^2}$$

$$\text{UCL} = \hat{\delta} + \sqrt{(u_1 - p_1)^2 + (p_2 - l_2)^2}$$

The function is vectorised: if vectors of length k are supplied, each corresponding tuple $(x_{1i}, n_{1i}, x_{2i}, n_{2i})$ is processed independently and the result is a $k \times 3$ matrix.

Value

A numeric matrix with one row per input pair and three columns:

delta Observed difference $\hat{p}_1 - \hat{p}_2$.

lc1 Lower confidence limit.

uc1 Upper confidence limit.

References

Newcombe, R. G. (1998). Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine*, **17**(8), 873–890. doi:10.1002/(SICI)1097-0258(19980430)17:8<873::AIDSIM779>3.0.CO;2I

See Also

[ks_newcombe_hs_strata](#), [ks_eairdiff_mn](#)

Examples

```
## Single comparison
ks_newcombe_hs(36L, 54L, 41L, 56L, alpha = 0.10)

## Zero events in one group
ks_newcombe_hs(0L, 50L, 10L, 50L)

## Vectorised: three comparisons at once
ks_newcombe_hs(
  x1 = c(36L, 38L, 3L),
  n1 = c(54L, 57L, 86L),
  x2 = c(41L, 42L, 3L),
  n2 = c(56L, 57L, 93L),
  alpha = 0.10
)
```

ks_newcombe_hs_strata *Stratified Newcombe Hybrid Score CI for a difference of two proportions*

Description

Extends Newcombe’s Hybrid Score method to stratified data using Mantel-Haenszel harmonic-mean weights. Within each stratum the Wilson score intervals are computed for both groups; the stratum-level contributions are then pooled via weighted combination to produce a single CI for the overall difference.

Usage

```
ks_newcombe_hs_strata(x1, n1, x2, n2, alpha = 0.05)
```

Arguments

x1	Integer vector of event counts for group 1, one element per stratum.
n1	Integer vector of sample sizes for group 1, one element per stratum (> 0).
x2	Integer vector of event counts for group 2, one element per stratum.
n2	Integer vector of sample sizes for group 2, one element per stratum (> 0).
alpha	Significance level; scalar in $(0, 1)$. Defaults to 0.05 (95% CI).

Details**Algorithm:**

1. Compute Wilson CIs $[l_{1i}, u_{1i}]$ and $[l_{2i}, u_{2i}]$ for each stratum i .
2. Assign stratum weights $W_i = n_{1i} n_{2i} / (n_{1i} + n_{2i})$ (harmonic-mean / Mantel-Haenszel weights).
3. Pooled point estimate: $\hat{\delta} = \sum W_i (p_{1i} - p_{2i}) / \sum W_i$.
4. Pooled lower limit:

$$\text{LCL} = \hat{\delta} - \sqrt{\frac{\sum W_i^2 (p_{1i} - l_{1i})^2}{(\sum W_i)^2} + \frac{\sum W_i^2 (u_{2i} - p_{2i})^2}{(\sum W_i)^2}}$$

5. Pooled upper limit: analogously.

A single stratum reduces to the un-stratified formula.

Value

A named numeric vector of length 3:

delta Weighted pooled point estimate.

lc1 Lower confidence limit.

uc1 Upper confidence limit.

References

Newcombe, R. G. (1998). Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine*, **17**(8), 873–890. doi:10.1002/(SICI)1097-0258(19980430)17:8<873::AIDSIM779>3.0.CO;2I

See Also

[ks_newcombe_hs](#)

Examples

```
## Two-stratum example
ks_newcombe_hs_strata(
  x1 = c(36L, 10L), n1 = c(54L, 30L),
  x2 = c(41L, 8L), n2 = c(56L, 30L)
)

## Single stratum – identical to ks_newcombe_hs()
ks_newcombe_hs_strata(36L, 54L, 41L, 56L)
ks_newcombe_hs(36L, 54L, 41L, 56L) # same result
```

ks_p_norm	Normality test p-value
-----------	------------------------

Description

Wrapper around `shapiro.test`; returns the p-value or NA on error.

Usage

```
ks_p_norm(x)
```

Arguments

x Numeric vector.

Value

A single numeric (p-value or NA).

ks_q1	25th percentile (Q1)
-------	----------------------

Description

Computes the 25th percentile using [quantile](#).
The quantile algorithm type can be set globally via `options(kstools.quantile.type = <1..9>)` (e.g. in `‘.Rprofile’`). Type 2 matches SAS default; type 7 is the R default. When the option is not set the package default of 2 is used.

Usage

```
ks_q1(x, na.rm = TRUE, type = getOption("kstools.quantile.type", default = 2L))
```

Arguments

x	Numeric vector.
na.rm	Logical; remove NAs before computing. Default TRUE.
type	Integer (1–9); quantile algorithm passed to quantile . Defaults to <code>getOption("kstools.quantile.type", 2L)</code> .

See Also

[ks_q3](#), [quantile](#)

Examples

```
## Use R default algorithm for this call
ks_q1(rnorm(100), type = 7)

## Set package-wide default (e.g. in .Rprofile)
old <- options(kstools.quantile.type = 7L)
ks_q1(rnorm(100)) # now uses type 7
options(old)      # restore
```

ks_q3	<i>75th percentile (Q3)</i>
-------	-----------------------------

Description

Computes the 75th percentile using [quantile](#).

The quantile algorithm type can be set globally via `options(kstools.quantile.type = <1..9>)` (e.g. in `‘.Rprofile’`). Type 2 matches SAS default; type 7 is the R default. When the option is not set the package default of 2 is used.

Usage

```
ks_q3(x, na.rm = TRUE, type = getOption("kstools.quantile.type", default = 2L))
```

Arguments

x	Numeric vector.
na.rm	Logical; remove NAs before computing. Default TRUE.
type	Integer (1–9); quantile algorithm passed to quantile . Defaults to <code>getOption("kstools.quantile.type", 2L)</code> .

See Also

[ks_q1](#), [quantile](#)

Examples

```
## Use SAS-compatible algorithm for this call
ks_q3(rnorm(100), type = 2)

## Set package-wide default (e.g. in .Rprofile)
old <- options(kstools.quantile.type = 7L)
ks_q3(rnorm(100))    # now uses type 7
options(old)          # restore
```

ks_read_xpt	<i>Read a SAS XPORT (.xpt) file</i>
-------------	-------------------------------------

Description

Reads a SAS XPORT v5 or v8/v9 file using a built-in C++ parser. Variable labels, formats, and informats are preserved as column attributes ("label", "format.sas", "sas.informat"). Multi-member files are returned as a named list of tibbles, one per dataset.

Usage

```
ks_read_xpt(file, convert_dates = TRUE, warn = TRUE, n_threads = NULL)
```

Arguments

file	Path to a SAS XPORT file.
convert_dates	Logical. If 'TRUE' (default), columns formatted as SAS dates or datetimes are converted to R 'Date' / 'POSIXct'. If 'FALSE', such columns are returned as numeric SAS day / second values.
warn	Logical. If 'TRUE' (default), parser warnings are emitted via [cli::cli_warn].
n_threads	Integer or 'NULL'. Number of decoder threads to use. 'NULL' (default) selects 'max(1, parallel::detectCores() - 1)'. Set to '1L' for fully deterministic single-threaded decoding.

Details

Date and datetime conversion, when enabled, applies the SAS-to-R epoch shift of 3653 days (315,619,200 seconds) and assigns the appropriate 'Date' / 'POSIXct' class to columns whose SAS format is recognised as a date or datetime format.

Value

A tibble for a single-member file, or a named list of tibbles for a multi-member file.

See Also

```
[ks_write_xpt()], [ks_xpt_labels()], [ks_xpt_formats()], [ks_set_labels()], [ks_set_formats()]
```

ks_roc	<i>Compare correlated ROC curve areas (AUC) via Mann-Whitney contrasts</i>
--------	--

Description

R analogue of the classic SAS ROC macro: computes AUC estimates for multiple predictors measured on the same subjects, estimates their covariance matrix using Mann-Whitney components with pairwise deletion, and performs Wald-style tests for user-specified linear contrasts.

Usage

```
ks_roc(data, var, response, contrast = NULL, details = FALSE, alpha = 0.05)
```

Arguments

data	A data frame containing predictors and the binary response.
var	Character vector of predictor column names.
response	Single character scalar naming the binary response column coded as 0/1.
contrast	Optional numeric contrast matrix. Columns must match <code>length(var)</code> . If <code>NULL</code> (default), all pairwise contrasts $AUC_i - AUC_j$ for $i < j$ are generated.
details	Logical; if <code>TRUE</code> , include intermediate matrices (<code>t</code> , <code>v</code> , <code>nx</code> , <code>ny</code> , <code>lv</code>) in output.
alpha	Significance level for two-sided confidence intervals.

Value

A list with elements:

areas Data frame with AUC, standard error, and confidence bounds.

contrasts Data frame with contrast estimates, SE, CI, chi-square, and p-value (1 df each row).

contrast_coefficients Contrast matrix used in the analysis.

overall_test Named numeric vector: chi-square, df, p-value for simultaneous contrast test.

pvalue Overall contrast-test p-value (same as `overall_test[["p_value"]]`).

details Only when `details = TRUE`: list of intermediate matrices `t`, `v`, `nx`, `ny`, `lv`.

ks_set_formats	<i>Set SAS variable formats on a data frame</i>
----------------	---

Description

Sets the "format.sas" attribute of selected columns from a named character vector.

Usage

```
ks_set_formats(data, formats)
```

Arguments

data	A data frame or tibble.
formats	Named character vector. Names are column names; values are SAS format strings (e.g. "DATE9.", "F8.2", "\$20.").

Value

The input 'data' with "format.sas" attributes updated.

See Also

[ks_xpt_formats()], [ks_set_labels()]

ks_set_labels	<i>Set SAS variable labels on a data frame</i>
---------------	--

Description

Sets the "label" attribute of selected columns from a named character vector. Names of 'labels' must match column names in 'data'. Unknown names are warned about and skipped.

Usage

```
ks_set_labels(data, labels)
```

Arguments

data	A data frame or tibble.
labels	Named character vector. Names are column names; values are labels. Use 'NA' or empty string to keep an existing label.

Value

The input 'data' with "label" attributes updated.

See Also

[ks_xpt_labels()], [ks_set_formats()]

ks_simulate_ar1

Simulate longitudinal data with AR(1) within-subject errors

Description

Generates a random sample of subjects with random intercepts and slopes, irregular observation times, and Gaussian AR(1) serially correlated residuals. The result is an [groupedData](#) object suitable for exploratory plots or simulation studies.

Usage

```
ks_simulate_ar1(
  n,
  beta0,
  beta1,
  ar.val = 0.4,
  sigma,
  tau0,
  tau1,
  tau01 = 0.3,
  m,
  seed = NULL
)
```

Arguments

n	Number of subjects (≥ 1).
beta0	Population mean intercept (fixed effect on the log-time scale).
beta1	Population mean slope (coefficient of $\log(\text{obs})$).
ar.val	Autoregressive coefficient ϕ for the within-subject errors (must lie strictly between -1 and 1).
sigma	Marginal standard deviation of the AR(1) error (after the scaling described in Details).
tau0, tau1	Standard deviations of random intercept and random slope.
tau01	Correlation between random intercept and slope (in $[-1, 1]$).
m	Maximum possible number of visits per subject (must be ≥ 4). Each subject receives between 4 and m observations, always including time $\text{obs} = 1$.
seed	Optional integer; if not NULL, passed to set.seed before random draws (reproducibility).

Details

Observation counts p_i are uniform on $\{4, \dots, m\}$ (rounded from continuous uniforms). Observation times are 1 plus a random subset of $\{2, \dots, m\}$ of size $p_i - 1$.

Random effects $(b_{0i}, b_{1i})'$ are bivariate normal with mean zero, SDs τ_{00} , τ_{11} and correlation τ_{01} . Outcomes are

$$y_{ij} = (\beta_0 + b_{0i}) + (\beta_1 + b_{1i}) \log(t_{ij}) + \varepsilon_{ij}$$

with ε_{ij} an AR(1) Gaussian series. Innovations are scaled so that the marginal residual SD is σ : `arma.sim(...)` is multiplied by $\sigma\sqrt{1 - \phi^2}$.

Value

An `groupedData` object (also inherits from `data.frame`) with columns `id`, `obs`, and `y`.

See Also

`arma.sim`, `mvrnorm`, `groupedData`

Examples

```
set.seed(1)
dt <- ks_simulate_ar1(
  n = 10, beta0 = 5, beta1 = 1, ar.val = 0.4, sigma = 2,
  tau0 = 1, tau1 = 0.5, tau01 = 0.2, m = 8, seed = 1L
)
head(dt)
```

ks_sum_na	<i>Add two values ignoring NA</i>
-----------	-----------------------------------

Description

Add two values ignoring NAs. Usually `5.0 + NA` gives `NA` as a result, but here `5.0 %+na% NA` will give `5.0`. `NA %+na% 5.0` also gives `5.0`. Applicable for scalars and vectors.

Usage

```
ks_sum_na(x, y)
```

```
x %+na% y
```

Arguments

<code>x</code>	Numeric scalar or vector
<code>y</code>	Numeric scalar or vector

Examples

```
NA %>na% 10
12.3 %>na% 10
12.3 %>na% NA
ks_sum_na(12.3, NA)
NA %>na% NA
c(1,2,NA) %>na% c(NA,2,3)
```

ks_summary

Descriptive summary for continuous variables

Description

Computes n, mean, SD, CV, median, Q1, Q3, min, max by group. Optionally adds a p-value for change from zero (paired t-test or Wilcoxon signed-rank, chosen by Shapiro-Wilk normality test).

Usage

```
ks_summary(
  data,
  var,
  ...,
  na.rm = TRUE,
  p.change = FALSE,
  extras = NULL,
  numdec = TRUE
)
```

Arguments

data	A data frame or tibble.
var	Unquoted name of the numeric variable to summarise.
...	Unquoted grouping variables. Must come before na.rm, p.change, extras, and numdec in the call, which are keyword-only arguments (they must always be named).
na.rm	Logical; remove NAs before computing. Default TRUE.
p.change	Logical; if TRUE, append a p.chg column (p-value for paired change, chosen by normality). Default FALSE.
extras	An optional named list of functions for additional per-group statistics. Each function is called as <code>f(x, na.rm = na.rm)</code> where <code>x</code> is the numeric vector of <code>var</code> for that group. Each function must return a single scalar value. The list names become column names in the result. Example: <code>extras = list(mad = stats::mad, sem = function(x, na.rm) sd(x, na.rm = na.rm) / sqrt(sum(!is.na(x))))</code> Names that clash with built-in columns (n, mean, sd, cv, median, q1, q3, min, max, p.chg, NUMDEC) will raise an error.

numdec Logical; if TRUE (the default), append a NUMDEC column giving the maximum number of decimal places observed in var within each group (via [ks_get_numdec](#) logic, capped at 4). Set to FALSE to suppress.

Details

Q1 and Q3 use [ks_q1](#) / [ks_q3](#), whose type argument defaults to `getOption("kstools.quantile.type", 2L)` (SAS-compatible). Set that option in `‘.Rprofile’` to change the algorithm globally.

Value

A tibble with one row per group and columns `n`, `mean`, `sd`, `median`, `q1`, `q3`, `min`, `max`, any extras columns, `cv`, (if `numdec = TRUE`) `NUMDEC`, and (if `p.change = TRUE`) `p.chg`.

See Also

[ks_q1](#), [ks_q3](#), [ks_get_numdec](#)

Examples

```
d <- data.frame(
  grp = rep(c("A", "B"), each = 10),
  val = c(rnorm(10, 5), rnorm(10, 8))
)
ks_summary(d, val, grp)

## Without NUMDEC
ks_summary(d, val, grp, numdec = FALSE)

## With extra statistics
ks_summary(d, val, grp,
  extras = list(
    mad = stats::mad,
    sem = function(x, na.rm) sd(x, na.rm = na.rm) / sqrt(sum(!is.na(x)))
  )
)
```

ks_supp_consistency_check

Check SDTM supplemental dataset structure

Description

Validates that a SUPP- domain data frame has the required SDTM structure and internally consistent IDVAR/IDVARVAL columns.

Usage

```
ks_supp_consistency_check(data)
```

Arguments

data A data frame or tibble representing a SUPP– domain.

Details

Two structural rules are enforced:

1. All seven standard SUPP columns must be present: STUDYID, USUBJID, IDVAR, IDVARVAL, QNAM, QLABEL, QVAL.
2. Every one of those seven columns must be of type character.

If both rules pass, the function further inspects IDVAR/IDVARVAL:

- **All blank** (empty string or NA): the merge key is USUBJID alone (noid = TRUE).
- **All non-blank**: a record-level key is available (noid = FALSE).
- **Mixed**: considered invalid — isvalid is set to FALSE.

Value

A named logical vector of length 3:

isvalid Dataset passes structural checks.

noid TRUE when all IDVAR/IDVARVAL are blank — merge key is USUBJID only.

multid TRUE when more than one distinct non-blank IDVAR value is present.

See Also

[ks_supp_merge](#)

Examples

```
supp <- data.frame(
  STUDYID = "S1",
  USUBJID = c("01", "02"),
  IDVAR   = "",
  IDVARVAL = "",
  QNAM    = "RACE2",
  QLABEL  = "Race (detail)",
  QVAL    = c("Asian", "White"),
  stringsAsFactors = FALSE
)
ks_supp_consistency_check(supp)

## Invalid: missing required column
bad <- supp[, -which(names(supp) == "QLABEL")]
ks_supp_consistency_check(bad)
```

ks_supp_merge

*Merge an SDTM supplemental domain with its primary domain***Description**

Transposes a SUPP– dataset from long (one row per variable) to wide and left-joins it onto the primary domain using the appropriate key columns.

Usage

```
ks_supp_merge(primary, supp, mvars = NULL, get_labels = TRUE, rep_na = FALSE)
```

Arguments

primary	A data frame or tibble: the primary SDTM domain (e.g. DM, EX).
supp	A data frame or tibble: the corresponding SUPP– domain (e.g. SUPPDM, SUPPEX). Must pass ks_supp_consistency_check .
mvars	Character vector of QNAM values to merge. NULL (default) merges all available variables.
get_labels	Logical; if TRUE (default) attach QLABEL values as column "label" attributes on the merged columns.
rep_na	Logical; if TRUE replace NA in merged columns with "". Default FALSE.

Details

The merge strategy depends on IDVAR/IDVARVAL:

Subject-level (noid = TRUE) All IDVAR/IDVARVAL are blank. The join key is STUDYID + USUBJID.

Record-level (noid = FALSE) Each row in the SUPP is tied to a specific record in the primary domain. IDVAR names the primary domain variable (e.g. EXSEQ) and IDVARVAL holds its value. When multiple distinct IDVAR values are present, each is processed in a separate join pass.

If get_labels = TRUE and the primary domain was loaded via haven (or another reader that preserves column "label" attributes), the per-variable labels from QLABEL are attached to the newly added columns.

Value

The primary data frame with SUPP variables added as new columns.

See Also

[ks_supp_consistency_check](#)

Examples

```
dm <- data.frame(
  STUDYID = "S1",
  USUBJID = c("01", "02", "03"),
  AGE     = c(34L, 52L, 47L),
  stringsAsFactors = FALSE
)
suppdm <- data.frame(
  STUDYID = "S1",
  USUBJID = c("01", "02", "03"),
  IDVAR   = "",
  IDVARVAL = "",
  QNAM    = "RACE2",
  QLABEL  = "Race (detail)",
  QVAL    = c("Asian", "White", "Asian"),
  stringsAsFactors = FALSE
)
ks_supp_merge(dm, suppdm)
```

ks_swap_cols

*Swap pairs of columns in a data frame***Description**

Exchanges column contents by name: for each named argument `lhs = rhs`, the column named `lhs` receives the values that were in `rhs`, and `rhs` receives the values that were in `lhs`. Multiple independent pairs can be given in one call.

Usage

```
ks_swap_cols(df, ...)
```

Arguments

<code>df</code>	A <code>data.frame</code> , <code>tibble</code> , or other object inheriting from <code>data.frame</code> .
<code>...</code>	Named pairs <code>col_a = col_b</code> (bare names only). Each name that appears on the left or right must be a column of <code>df</code> , and each column may appear at most once across all pairs (no overlapping swaps).

Details

The syntax `USUBJID = SUBJID` swaps the two columns: after the call, the column still named `USUBJID` holds the former `SUBJID` values and vice versa. This is useful when harmonising SDTM-style identifiers or fixing column order after merges.

With zero pairs in `...`, `df` is returned unchanged.

Value

`df` with the specified column pairs exchanged (same class as the input where possible).

See Also[ks_aligndec](#)**Examples**

```
d <- data.frame(SUBJID = c("001", "002"), USUBJID = c("01-001", "01-002"))
ks_swap_cols(d, USUBJID = SUBJID)
```

ks_swoe	<i>Smoothed weight of evidence (SWOE) for categorical predictors</i>
---------	--

Description

Encodes each level of a categorical variable with a smoothed log-odds of the response, relative to the overall response rate. Useful as a numeric replacement for a factor when screening logistic-type models. Designed for use inside `dplyr::mutate()`.

Usage

```
ks_swoe(x, y, prior.n = 24)
```

Arguments

<code>x</code>	Categorical predictor (factor, character, or other grouping vector).
<code>y</code>	Binary response (numeric 0/1 or logical), same length as <code>x</code> .
<code>prior.n</code>	Pseudo-count (prior strength) for the empirical-Bayes-style smoothing in the logit; the original implementation used 24.

Details

Let ρ be the overall mean of the response and, for each level k of x , let e_k be the number of events (response = 1) and n_k the number of non-missing responses. The smoothed logit for level k is

$$\log \frac{e_k + \rho \cdot m}{n_k - e_k + (1 - \rho) \cdot m}$$

with $m = \text{prior.n}$. A global logit $\log(\rho/(1 - \rho))$ is appended internally for completeness; row values are taken by matching each row's x level to the named level logits.

If $\rho \in \{0, 1\}$, the global logit is undefined; the function clamps ρ to $(\varepsilon, 1 - \varepsilon)$ for that term only, with a warning.

Value

A numeric vector of length `length(x)` (unnamed), giving the SWOE score for each row's level of x .

References

The construction follows common “weight of evidence” encodings with Dirichlet-style smoothing; the choice prior `.n = 24` matches the historical default in this package.

See Also

[glm](#) for logistic regression

Examples

```
set.seed(1234)
dt <- data.frame(
  yn = rbinom(50, 1, 0.3),
  sex = factor(rbinom(50, 1, 0.55), levels = c(0, 1), labels = c("F", "M"))
)
ks_swoe(dt$sex, dt$yn)
if (requireNamespace("dplyr", quietly = TRUE)) {
  dplyr::mutate(dt, sex_swoe = ks_swoe(sex, yn))
}
```

ks_vswitch	<i>Vectorized switch (SAS select-when maps)</i>
------------	---

Description

Looks up each element of ‘`x`’ among the named values of ‘`...`’ — a ‘`switch()`’ that works on a whole vector at once, like SAS ‘`select(when ...)`’ or a format-driven recode.

Usage

```
ks_vswitch(.x, ..., .na = NULL, .default = NULL)
```

Arguments

- | | |
|-----------------------|---|
| <code>.x</code> | A vector of lookup keys (character, factor, or numeric; coerced with <code>as.character()</code> for matching). |
| <code>...</code> | Named values of a single common type — the names are keys, the values are results. ‘ <code>vapply()</code> ’ errors when the types disagree. |
| <code>.na</code> | Value returned where ‘ <code>x</code> ’ is ‘ <code>NA</code> ’; defaults to an ‘ <code>NA</code> ’ of the same type as the first value of ‘ <code>...</code> ’. |
| <code>.default</code> | Value returned for keys present in ‘ <code>x</code> ’ but absent from ‘ <code>...</code> ’; defaults to an ‘ <code>NA</code> ’ of the same type as the first value of ‘ <code>...</code> ’. |

Value

A vector the same length as ‘`x`’ with the storage type of the values in ‘`...`’. The key argument is named ‘`.x`’ (leading dot) so that bare case names like ‘`x = ...`’ in dots cannot collide with it.

Examples

```
ks_vswitch(c("a", "b", "a", NA, "zz"), a = 10L, b = 20L)
ks_vswitch(c("a", "b", NA, "zz"), a = 10L, b = 20L, .na = 0L, .default = -1L)
ks_vswitch(factor(c("x", "y")), x = 1.5, y = 2.5)
```

ks_williams

Build a randomized Williams cross-over design and schedule

Description

R analogue of the SAS WILLIAMS macro. The function constructs a Williams design from a randomized cyclic Latin square and generates a block-randomized subject schedule.

Usage

```
ks_williams(tmts_num, tmt_names, sample_size, out_rtf = NULL, seed_num = NULL)
```

Arguments

tmts_num	Integer number of treatments.
tmt_names	Character vector of treatment labels (length must equal tmts_num).
sample_size	Integer target number of subjects.
out_rtf	Optional output file path. If provided, a simple RTF report with design and randomization schedule tables is written.
seed_num	Optional integer seed. If NULL, current system time is used.

Value

A list with:

design Data frame of Williams sequences by period.

schedule Data frame of randomized subject assignments.

seed Seed used.

seed_generated Logical; TRUE when seed was auto-generated.

n_sequences Number of unique sequences.

block_num Number of full randomization blocks used.

sample_size_warning Logical; TRUE when sample_size is not a multiple of n_sequences.

ks_write_xpt

*Write a data frame to a SAS XPORT (.xpt) file***Description**

Writes a data frame to a SAS XPORT V5 or V8 file using a built-in C++ writer. Variable labels and SAS formats are taken from per-column attributes ("label", "format.sas") or the optional 'labels' / 'formats' arguments, which override attributes.

Usage

```
ks_write_xpt(
  data,
  file,
  version = 5L,
  convert_dates = TRUE,
  dataset_name = NULL,
  dataset_label = NULL,
  labels = NULL,
  formats = NULL
)
```

Arguments

data	A data frame.
file	Output path.
version	Integer 5 or 8. Default '5L' (widest compatibility).
convert_dates	Logical. If 'TRUE' (default), 'Date' and 'POSIXct' columns are converted to SAS numeric (days / seconds since 1960-01-01).
dataset_name	SAS dataset name. Up to 8 characters for V5, up to 32 for V8. Defaults to "DATASET".
dataset_label	Up to 40 characters describing the dataset.
labels	Optional named character vector of variable labels keyed by column name. Overrides per-column "label" attributes.
formats	Optional named character vector of SAS formats (e.g. "DATE9.", "F8.2") keyed by column name. Overrides per-column "format.sas" attributes.

Details

Date and datetime conversion, when enabled, applies the R-to-SAS epoch shift to columns of class 'Date' or 'POSIXct'.

Value

The output file path, invisibly.

See Also

[ks_read_xpt()]

ks_xpt_content	<i>List datasets inside a SAS XPORT v8/v9 file</i>
----------------	--

Description

Scans the raw bytes of a SAS Transport Version 8 file for member (MEMBV8) headers and returns the byte range of each member.

Usage

```
ks_xpt_content(file, chunk_size = 5L * 80L * 1024L)
```

Arguments

file	Path to a .xpt or .v9xpt file.
chunk_size	Number of raw bytes to read per pass. Default 5 * 80 * 1024.

Details

The file is read in chunks of chunk_size bytes (default approximately 400 KB). A MEMBV8 header that spans two adjacent chunks would not be detected; this is exceedingly unlikely for practical SDTM files.

Value

A named list; each element corresponds to one dataset and contains a named integer vector c(start = <1-based pos>, stop = <1-based pos>).

See Also

[ks_extract_single_xpt](#), [ks_batch_load_from_single_xpt](#)

Examples

```
## Not run:
## content <- ks_xpt_content("path/to/study.xpt")
## names(content)          # dataset names (lower-case)
## content[["ae"]]         # byte range for the AE member

## End(Not run)
```

ks_xpt_formats	<i>Get SAS variable formats from a data frame</i>
----------------	---

Description

Returns the "format.sas" attribute of each column as a named character vector. Columns without a "format.sas" attribute are reported as the empty string.

Usage

```
ks_xpt_formats(data)
```

Arguments

data	A data frame or tibble.
------	-------------------------

Value

Named character vector of SAS format strings (one entry per column).

See Also

[ks_set_formats()], [ks_xpt_labels()], [ks_read_xpt()], [ks_write_xpt()]

ks_xpt_labels	<i>Get SAS variable labels from a data frame</i>
---------------	--

Description

Returns the "label" attribute of each column as a named character vector. Columns without a "label" attribute are reported as the empty string.

Usage

```
ks_xpt_labels(data)
```

Arguments

data	A data frame or tibble.
------	-------------------------

Value

Named character vector of variable labels (one entry per column).

See Also

[ks_set_labels()], [ks_xpt_formats()], [ks_read_xpt()], [ks_write_xpt()]

ks_z_score	<i>Calculate Z score</i>
------------	--------------------------

Description

For evaluation of how far a measured value is from the mean (in case of data normality), following formulation used

Usage

```
ks_z_score(x, na.rm = TRUE)
```

Arguments

x	Numeric vector
na.rm	Flag for ignoring NAs. Default = TRUE

Details

Useful values are +V-3.

Examples

```
set.seed(345)
ks_z_score(rnorm(10))
## -0.9102967 -0.4446436 -0.3358703 -0.4548544 -0.2493297
## -0.7708130 -1.0418824 1.3891393 1.3375406 1.4810102
```

read.csv3	<i>Read CSV file with column types and lengths in comments</i>
-----------	--

Description

Read a single CSV file into R. Optionally parses KeyStat-style comment headers that encode column type, length (for character variables), and labels.

Usage

```
read.csv3(file, with_length = FALSE)
```

Arguments

file	Path to the source CSV file.
with_length	If TRUE, the CSV file contains KeyStat metadata comments with types and lengths. Default is FALSE.

Details

File encoding should be UTF-8. Readable comments in the CSV file: #NAME\$character(length)|numeric\$Label.

This function was written for correct reading of CSV files. Some CSV files without metadata in comments cannot be read correctly into R.

Value

A data frame with column labels set from the comment metadata (when available).

See Also

[ks_get_data](#)

Index

`%+na%` (`ks_sum_na`), 44

`arima.sim`, 44

`basename`, 12

`boot`, 29

`boot.ci`, 29

`commandArgs`, 12

`glm`, 51

`groupedData`, 43, 44

`knit_print.ks_mianalyze` (`ks_mianalyze`), 30

`ks_aligndec`, 3, 50

`ks_auc`, 3, 4

`ks_batch_load`, 5, 7, 23

`ks_batch_load_from_single_xpt`, 5, 20, 54

`ks_batch_load_xlsx`, 7

`ks_be_2x2`, 8

`ks_boot_ci`, 9

`ks_code_to_fmt`, 10

`ks_compare_na`, 11

`ks_current_filename`, 12

`ks_cv`, 13

`ks_del_tmp`, 13

`ks_desc_coded`, 14

`ks_dtc2dtn`, 16

`ks_dunnett_crit`, 16

`ks_eairdiff_mn`, 17, 18, 19, 36

`ks_eairdiff_strata_mn`, 18, 18

`ks_extract_single_xpt`, 6, 19, 54

`ks_geom_mean`, 20

`ks_get_data`, 21, 57

`ks_get_labels`, 23

`ks_get_numdec`, 24, 46

`ks_hosmer_lemeshow`, 25

`ks_impute_dtc`, 26

`ks_median_ci`, 29

`ks_mianalyze`, 30

`ks_mv_normality`, 32

`ks_n_pct`, 14, 15, 33

`ks_newcombe_hs`, 35, 37

`ks_newcombe_hs_strata`, 36, 36

`ks_p_norm`, 38

`ks_q1`, 38, 39, 46

`ks_q3`, 39, 39, 46

`ks_read_xpt`, 22, 40

`ks_roc`, 41

`ks_set_formats`, 42

`ks_set_labels`, 42

`ks_simulate_ar1`, 43

`ks_sum_na`, 11, 44

`ks_summary`, 45

`ks_supp_consistency_check`, 46, 48

`ks_supp_merge`, 47, 48

`ks_swap_cols`, 49

`ks_swoe`, 50

`ks_vswitch`, 51

`ks_williams`, 52

`ks_write_xpt`, 53

`ks_xpt_content`, 5, 6, 19, 20, 54

`ks_xpt_formats`, 55

`ks_xpt_labels`, 55

`ks_z_score`, 56

`lapply`, 6

`log`, 21

`match.arg`, 26

`mclapply`, 6

`median`, 30

`mianalyze` (`ks_mianalyze`), 30

`mvnorm`, 44

`quantile`, 21, 29, 30, 38, 39

`read.csv3`, 23, 56

`read_xpt`, 19, 20

`set.seed`, 29, 43

source, [12](#)

summary.ks_mianalyze (ks_mianalyze), [30](#)

tibble, [15](#), [20](#), [21](#), [24](#), [27](#)

uniroot.all, [17](#), [18](#)