Package 'colleyRstats'

November 15, 2026

Type Package

Title Functions to Streamline Statistical Analysis and Reporting

Version 0.3.2

Description Built upon popular R packages such as 'ggstatsplot' and 'ARTool', this collection offers a wide array of tools for simplifying reproducible analyses, generating high-quality visualizations, and producing 'APA'-compliant outputs. The primary goal of this package is to significantly reduce repetitive coding efforts, allowing you to focus on interpreting results. Whether you're dealing with ANOVA assumptions, reporting effect sizes, or creating publication-ready visualizations, this package makes these tasks easier.

URL https://github.com/M-Colley/colleyRstats, https://m-colley.github.io/colleyRstats/

BugReports https://github.com/M-Colley/colleyRstats/issues

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.3.0), ggplot2 (>= 3.5.0)

Imports dplyr (>= 1.1.0), effectsize, ggpmisc (>= 0.5.0), ggsignif, ggstatsplot (>= 1.0.0), grid, lifecycle, purrr, rlang, see, stats, statsExpressions (>= 2.0.0), stringr, tidyr, utils

Suggests afex, ARTool (>= 0.11.0), av, Cairo, callr, car, clipr, conflicted, easystats, emmeans, emoa, FSA, ggtext, glmmTMB, Hmisc, knitr, lme4, lmerTest, MASS, mclogit, moocore, nnet, nparLD, ordinal, parameters, patchwork, performance, pkgload, psych, readxl, renv, report, rmarkdown, rstatix, scales, testthat (>= 3.3.0), tibble, withr, WRS2, writexl, xtable

Config/testthat/edition 3

Config/testthat/parallel false

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Mark Colley [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5207-5029>)

Maintainer Mark Colley <mark.colley@yahoo.de>

Repository CRAN

Date/Publication 2026-11-15 21:40:15 UTC

Contents

add_pareto_emoa_column . . . 3
add_pareto_moocore_column . . . 5
analyze_and_report . . . 6
animate_mobo2 . . . 8
assumption_methods_text . . . 9
checkAssumptionsForAnova . . . 11
check_homogeneity_by_group . . . 13
check_normality_by_group . . . 14
check_questionnaire . . . 15
cite_methods . . . 17
classify_outcome . . . 18
colleyRstats_setup . . . 19
colley_theme . . . 21
debug_contr_error . . . 22
define_questionnaire . . . 23
define_result_macro . . . 25
emit_name_macros . . . 26
emit_overleaf . . . 27
expand_latex_macros . . . 28
figure_base_size . . . 29
fit_recommended . . . 30
generateEffectPlot . . . 33
generateMoboPlot . . . 35
generateMoboPlot2 . . . 38
ggbetweenstatsWithPriorNormalityCheck . . . 40
ggbetweenstatsWithPriorNormalityCheckAsterisk . . . 42
ggwithinstatsWithPriorNormalityCheck . . . 43
ggwithinstatsWithPriorNormalityCheckAsterisk . . . 45
latexify_report . . . 47
latex_escape . . . 49
latex_preamble . . . 49
list_questionnaires . . . 50
na.zero . . . 50
normalize . . . 51
not_empty . . . 52
not_in . . . 52
n_fun . . . 53
pathPrep . . . 53
plot_fit . . . 54

questionnaire_items . . . 55
recommend_test . . . 56
remove_outliers_REI . . . 58
replace_values . . . 60
reportART . . . 61
reportArtCon . . . 62
reportArtConTable . . . 64
reportCLMM . . . 67
reportDunnTest . . . 68
reportDunnTestTable . . . 70
reportggstatsplot . . . 72
reportggstatsplotPostHoc . . . 73
reportGLMM . . . 76
reportMeanAndSD . . . 78
reportNparLD . . . 79
reportNPAV . . . 81
report_all . . . 82
reshape_data . . . 83
reverse_code . . . 85
rFromNPAV . . . 86
rFromWilcox . . . 87
rFromWilcoxAdjusted . . . 88
save_paper_figure . . . 89
score_questionnaire . . . 90
score_reliability . . . 93
stat_sum_df . . . 96
summarize_sickness . . . 97
use_colleyrstats_sty . . . 98
use_study_project . . . 98

Index **101**

`add_pareto_emoa_column`

Add `PARETO_EMOA` *Column to a Data Frame*

Description

This function calculates the Pareto front using emoa for a given set of objectives in a data frame and adds a new column, `PARETO_EMOA`, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_emoa_column(data, objectives, maximise = FALSE)
```

Arguments

`data`	A data frame containing the data, including the objective columns.
`objectives`	A character vector specifying the names of the objective columns in `data`. These columns should be numeric and will be used to calculate the Pareto front.
`maximise`	Direction of optimisation. `FALSE` (the default) treats every objective as one to be *minimised*, which is what **emoa** does natively. Pass `TRUE` when larger is better for every objective – as it is for trust, acceptance, perceived safety and most other rating-scale outcomes – or a logical vector with one entry per objective for a mixed problem, e.g. `c(TRUE, TRUE, FALSE)` to maximise the first two and minimise the third. Objectives flagged `TRUE` are negated internally, so you no longer need to pass negated copies of your own columns.

Value

A data frame with the same columns as `data`, along with an additional column, `PARETO_EMOA`, which is `TRUE` for rows that are on the Pareto front and `FALSE` otherwise. Identical rows share one verdict (a copy of a non-dominated point is non-dominated too), as in `add_pareto_moocore_column()`. Rows with a missing objective value get `NA`, with a warning, and the front is computed from the complete rows.

See Also

`add_pareto_moocore_column()`, which answers the same question via **moocore** and accepts the same `maximise` argument.

Examples

```
if (requireNamespace("emoa", quietly = TRUE)) {
  # Define objective columns
  objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

  # Example data frame
  main_df <- data.frame(
    trust = runif(10),
    predictability = runif(10),
    perceivedSafety = runif(10),
    Comfort = runif(10)
  )

  # Add the Pareto front column (minimising, the default)
  main_df <- add_pareto_emoa_column(data = main_df, objectives)
  head(main_df)

  # All four objectives are ratings where higher is better
  main_df <- add_pareto_emoa_column(main_df, objectives, maximise = TRUE)
}
```

add_pareto_moocore_column

Add PARETO_MOOCORE *Column to a Data Frame*

Description

This function calculates the Pareto front using moocore for a given set of objectives in a data frame and adds a new column, PARETO_MOOCORE, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_moocore_column(data, objectives, maximise = FALSE)
```

Arguments

data	A data frame containing the data, including the objective columns.
objectives	A character vector specifying the names of the objective columns in data. These columns should be numeric and will be used to calculate the Pareto front.
maximise	Direction of optimisation, passed through to moocore::is_nondominated(). FALSE (the default) treats every objective as one to be *minimised*. Pass TRUE when larger is better for every objective – as it is for trust, acceptance, perceived safety and most other rating-scale outcomes – or a logical vector with one entry per objective for a mixed problem, e.g. c(TRUE, TRUE, FALSE) to maximise the first two and minimise the third. This removes the need to pass negated copies of your own columns.

Value

A data frame with the same columns as data, along with an additional column, PARETO_MOOCORE, which is TRUE for rows that are on the Pareto front and FALSE otherwise. Identical rows share one verdict: every copy of a non-dominated point is kept (keep_weakly = TRUE), as in add_pareto_emoa_column(), where moocore::is_nondominated() on its own would mark only the first copy. Rows with a missing objective value get NA, with a warning, and the front is computed from the complete rows.

See Also

add_pareto_emoa_column(), which answers the same question via **emoa** and accepts the same maximise argument.

Examples

```
if (requireNamespace("moocore", quietly = TRUE)) {
  # Define objective columns
  objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

  # Example data frame
```

```
  main_df <- data.frame(
    trust = runif(10),
    predictability = runif(10),
    perceivedSafety = runif(10),
    Comfort = runif(10)
  )

  # Add the Pareto front column (minimising, the default)
  main_df <- add_pareto_moocore_column(data = main_df, objectives)
  head(main_df)

  # All four objectives are ratings where higher is better
  main_df <- add_pareto_moocore_column(main_df, objectives, maximise = TRUE)

  # Mixed: maximise the ratings, minimise a workload score
  main_df$workload <- runif(10)
  main_df <- add_pareto_moocore_column(
    main_df,
    c(objectives, "workload"),
    maximise = c(TRUE, TRUE, TRUE, TRUE, FALSE)
  )
}
```

`analyze_and_report` *Analyze one dependent variable and produce everything a paper needs*

Description

One-call pipeline for a single dependent variable: checks the assumptions (producing a ready-made methods sentence via `assumption_methods_text()`), builds the matching **ggstatsplot** figure with automatic parametric/non-parametric selection, reports the omnibus test via `reportggstatsplot()`, and – for more than two groups – reports the significant post-hoc comparisons via `reportggstatsplotPostHoc()`.

Usage

```
analyze_and_report(
  data,
  dv,
  iv,
  design = c("between", "within"),
  ylab = dv,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_to = NULL,
  subject = NULL
)
```

Arguments

data	the data frame (long format for design = "within": one row per participant and condition)
dv	the dependent variable (column name as string)
iv	the independent variable (column name as string); treated as a factor, so numeric condition codes are categories
design	"between" for between-subjects data (default) or "within" for repeated measures
ylab	label for the dependent variable; defaults to dv
xlabels	optional labels for the x-axis
plotType	either "box", "violin", or "boxviolin" (default)
sink_to	optional path of a .tex file; the methods sentence, omnibus result, and post-hoc sentences are written there so a manuscript can \input{} them
subject	the participant-ID column (a string). Required for design = "within", where it pairs the observations; without it they would be paired by row order. For design = "between" it is optional and only used to check that no participant contributes more than one row, which a between-subjects test would wrongly treat as independent observations.

Details

Every part of the output is computed from one data set: the rows with both dv and iv observed and, for design = "within", only the participants measured in every condition (the others are dropped with a message naming them; more than one row per participant and condition is an error). The methods sentence, the test, the figure and the means quoted in the post-hoc sentences therefore all describe the same participants.

Value

Invisibly returns a list with components plot (the ggplot), methods (assumption-check sentence), text (omnibus result), posthoc (post-hoc sentences, or NULL for two groups), and sentences (all text combined, in manuscript order).

Examples

```
result <- analyze_and_report(mtcars, dv = "mpg", iv = "cyl")
result$plot

# Repeated measures: name the participant column
set.seed(1)
rm_df <- data.frame(
  participant = rep(1:15, times = 3),
  condition   = rep(c("A", "B", "C"), each = 15)
)
rm_df$score <- rnorm(45, mean = c(A = 5, B = 6, C = 7)[rm_df$condition])
analyze_and_report(rm_df, dv = "score", iv = "condition",
                   design = "within", subject = "participant")
```

`animate_mobo2` *Animate a Multi-objective Optimization Plot*

Description

Renders `generateMoboPlot2()` one iteration at a time and encodes the frames into a video, so a talk or a supplement can show an optimisation run building up rather than only its end state. Frame *i* is the same plot restricted to the data up to iteration *i*: the mean points, their bootstrapped intervals, the fitted line and its equation all move as the run proceeds.

Usage

```
animate_mobo2(
  data,
  x = "Iteration",
  y,
  filename,
  ...,
  fps = 5,
  end_pause = 2,
  width = 8,
  height = 5,
  dpi = 150,
  label_iterations = TRUE
)
```

Arguments

`data`	A data frame holding the run, as for `generateMoboPlot2()`.
`x`	A string naming the iteration column in `data`. Default `"Iteration"`.
`y`	A string naming the objective column in `data`.
`filename`	Path of the video to write. Its extension chooses the container: `.mp4`, `.gif`, `.mov`, `.webm` – whatever the `av` package's FFmpeg build can encode.
`...`	Further arguments for `generateMoboPlot2()`, e.g. `phaseCol`, `fillColourGroup`, `ytext`, `legendPos`, `fillLabels`.
`fps`	Frames per second. One frame is drawn per iteration, so this is equally the number of iterations shown per second. Default 5.
`end_pause`	Seconds to hold the final frame, so the finished plot can be read before the video ends or loops. Default 2; use 0 for no hold.
`width, height`	Frame size in inches. Default 8 x 5.
`dpi`	Resolution. Default 150, so the default frame is 1200 x 750 px.
`label_iterations`	Whether to caption each frame with the iteration it shows. Default `TRUE`, which writes the plot's subtitle.

Details

The plot is built once from the complete data, and a frame only ever hides rows. That is what keeps the axes, the sampling/optimisation guides and the legend still while the data grows – and it is also the only way the early frames can be drawn at all, since `generateMoboPlot2()` requires both phases to be present and the first iterations are all sampling.

Value

Invisibly returns `filename`.

See Also

`generateMoboPlot2()` for the static plot.

Examples

```
# Kept deliberately tiny: every frame is a full ggplot render, so the cost
# of the example is set by the number of iterations, not by the frame size.
if (requireNamespace("av", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    Iteration   = rep(1:3, each = 4),
    ConditionID = rep(rep(c("A", "B"), each = 2), 3),
    Phase       = rep(c("Sampling", "Optimization"), times = c(2 * 4, 4))
  )
  df$score <- 0.3 + 0.05 * df$Iteration + stats::rnorm(nrow(df), sd = 0.05)

  animate_mobo2(
    df,
    y = "score", ytext = "Score",
    filename = file.path(tempdir(), "mobo.mp4"),
    width = 4, height = 2.5, dpi = 96, end_pause = 0
  )
}
```

---

assumption_methods_text

Methods-section sentence justifying the test selection

Description

Runs `check_normality_by_group()` (and optionally a test for homogeneity of variances) and turns the outcome into a ready-made methods-section sentence that says what was tested and reports the statistic the decision rests on. This is the justification reviewers expect next to the choice of a parametric or non-parametric test.

Usage

```
assumption_methods_text(
  data,
  x,
  y,
  include_homogeneity = FALSE,
  subject = NULL
)
```

Arguments

data	the data frame
x	the grouping variable (column name as string)
y	the dependent variable (column name as string)
include_homogeneity	whether to also report the homogeneity-of-variance test run by check_homogeneity_by_group() (named as that function reports it: the Brown-Forsythe, i.e. median-centred Levene's, test). Between subjects only: with subject given it is skipped with a message, because equal variances across conditions are not an assumption of a repeated-measures analysis (its counterpart, sphericity, is tested and corrected by the ANOVA itself). Default FALSE.
subject	the participant-ID column for a within-subjects design, as a string; NULL (default) for a between-subjects design. Passed to check_normality_by_group().

Details

The sentence follows the check exactly:

- **Between subjects**: one Shapiro–Wilk test per group, Holm-corrected across the groups; the adjusted p-values are labelled p_{Holm}. A rejection names the group(s) with their W and p; otherwise the group with the smallest p is reported.
- **Within subjects** (subject given): one test on the paired differences (two conditions) or on the residuals of the additive participant + condition model (more conditions). Participants excluded for lacking a condition are counted in the text.
- A group that could not be tested (fewer than three values, or all values identical – e.g. everyone ticked the top of a rating scale) is named and stated to have been treated as non-normal, which sends the analysis to the non-parametric branch.

p-values are printed so that rounding never moves them across .05, .01 or .10, and all statistics are plain LaTeX math (no custom macros needed).

Value

Invisibly returns the sentence(s) as a single string; the text is also emitted via message().

Examples

```
set.seed(1)
d <- data.frame(g = rep(c("A", "B"), each = 20), v = rnorm(40))
assumption_methods_text(d, x = "g", y = "v")

# within subjects: the paired differences are tested
d$id <- rep(1:20, times = 2)
assumption_methods_text(d, x = "g", y = "v", subject = "id")
```

`checkAssumptionsForAnova` *Check the assumptions for an ANOVA with a variable number of factors: Normality and Homogeneity of variance assumption.*

Description

Checks the normality and variance assumptions of a factorial ANOVA and recommends the parametric or the non-parametric analysis. Every factor is treated as categorical – numeric condition codes (1, 2, 3) included, which `stats::lm()` would otherwise fit as a single linear covariate – and the message names the tests that were run and their results.

Usage

```
checkAssumptionsForAnova(data, y, factors, subject = NULL)

check_assumptions_anova(data, y, factors, subject = NULL)
```

Arguments

`data`	the data frame
`y`	The dependent variable for which assumptions should be checked
`factors`	A character vector of factor names
`subject`	the participant-ID column for a design with within-subject factors, as a string; `NULL` (default) for a between-subjects design.

Details

- **Between subjects** (`subject = NULL`, the default): a Shapiro-Wilk test on the residuals of the linear model `y ~ A * B * ...`; a Shapiro-Wilk test within every cell of the design, Holm-corrected across the cells (as in `check_normality_by_group()`); and the Brown-Forsythe test (median-centred Levene's test) across the cells. This assumes one row per participant: a within-subject factor analysed this way is tested on residuals that still contain each participant's overall level, so pass `subject` for repeated measures.

- **Within subjects or mixed** (`subject` given): a Shapiro-Wilk test on the residuals of `y ~ A * B * ... + subject`, with the participant as a fixed factor – the residuals after removing each participant's overall level, which is what a repeated-measures ANOVA assumes to be normal. The raw scores per cell are not tested, as they also carry the between-participant spread. Variance homogeneity is checked only for between-subject factors (those constant within a participant), with the Brown-Forsythe test across their groups at each combination of the within-subject factors (Holm-corrected). For within-subject factors the corresponding assumption is sphericity, which `rstatix::anova_test()` tests (Mauchly) and corrects (Greenhouse-Geisser) itself. More than one row per participant and cell is an error: aggregate repeated trials first.

A residual set or cell that cannot be tested – fewer than three values, or no variance – counts as **not** normal, so the advice errs towards the non-parametric analysis. Rows with a missing value in `y`, `factors` or `subject` are left out.

Value

The guidance text (also emitted as a message), invisibly, with attributes:

`parametric` `TRUE` if every check passed.

`design` `"between"`, `"within"` or `"mixed"`.

`method` named character vector naming the tests run (`residuals`, `groupwise`, `homogeneity`).

`residuals` the Shapiro-Wilk result on the model residuals (columns as in `check_normality_by_group()`'s `tests`).

`groupwise` between subjects: the per-cell Shapiro-Wilk results with Holm-adjusted p-values; `NULL` otherwise.

`homogeneity` the Brown-Forsythe result(s) (`df1`, `df2`, `statistic`, `p`, `p_adjusted`), or `NULL` when no between-subject factor exists.

Naming

`check_assumptions_anova()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`checkAssumptionsForAnova()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("rstatix", quietly = TRUE)) {
  set.seed(123)

  main_df <- data.frame(
    tlx_mental      = rnorm(40),
    Video           = factor(rep(c("A", "B"), each = 20)),
    DriverPosition  = factor(rep(c("Left", "Right"), times = 20))
  )

  checkAssumptionsForAnova(
```

```
  data    = main_df,
  y       = "tlx_mental",
  factors = c("Video", "DriverPosition")
)

# The same two factors measured within each of 10 participants
within_df <- expand.grid(
  id             = 1:10,
  Video          = c("A", "B"),
  DriverPosition = c("Left", "Right")
)
within_df$tlx_mental <- rnorm(10)[within_df$id] + rnorm(40)
checkAssumptionsForAnova(
  data    = within_df,
  y       = "tlx_mental",
  factors = c("Video", "DriverPosition"),
  subject = "id"
)
}
```

`check_homogeneity_by_group` *Check homogeneity of variances across groups*

Description

Runs the **Brown-Forsythe test**: Levene's test computed on the absolute deviations from each group's *median* rather than its mean (`rstatix::levene_test()`'s default, `center = median`). The median-centred version keeps close to its nominal error rate when the data are skewed or heavy-tailed, as rating-scale data often are, where Levene's mean-centred original rejects too often (Brown & Forsythe, 1974). Report it under that name; the `"method"` attribute carries it, e.g. for `assumption_methods_text()`.

Usage

```
check_homogeneity_by_group(data, x, y)
```

Arguments

`data`	the data frame
`x`	the grouping variable (column name as string)
`y`	the dependent variable (column name as string)

Details

The grouping column is treated as a factor whatever its type, so numeric condition codes (1, 2, 3) define groups rather than a covariate, and non-syntactic column names such as `"Mental Demand"` work.

Value

TRUE if the test is non-significant (p >= .05), FALSE otherwise. Attributes: `"test"`, the test result (columns `df1`, `df2`, `statistic`, `p`); `"method"`, the name of the test that was run (`"Brown-Forsythe test (median-centred Levene's test)"`).

References

Brown, M. B., & Forsythe, A. B. (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association, 69*(346), 364–367. doi:10.1080/01621459.1974.10482955

`check_normality_by_group` *Check normality for groups*

Description

Decides between a parametric and a non-parametric analysis from Shapiro-Wilk tests, testing the quantity the parametric test actually assumes to be normal.

Usage

```
check_normality_by_group(data, x, y, subject = NULL, p_adjust = "holm")
```

Arguments

`data`	the data frame
`x`	the grouping (condition) column, as a string
`y`	the outcome column, as a string
`subject`	the participant-ID column for a within-subjects design, as a string; `NULL` (default) for a between-subjects design.
`p_adjust`	multiplicity correction across groups, passed to `stats::p.adjust()`. Default `"holm"`; only used between subjects.

Details

- **Between subjects** (`subject = NULL`): one test per group, with the p-values corrected for the number of groups (`p_adjust`, Holm by default). Without the correction, six perfectly normal groups would send about one analysis in four to the non-parametric branch by chance alone.
- **Within subjects** (`subject` given): a paired t-test assumes the *differences* are normal, and a repeated-measures ANOVA the *residuals* after removing participant and condition effects – not the raw scores per condition, which also carry the between-participant spread. With two conditions the per-participant differences are tested; with more, the residuals of the additive model `y ~ x + subject`. Participants lacking any condition are left out (they cannot enter a paired analysis), and more than one row per participant and condition is an error.

A group that cannot be tested – fewer than three values, or no variance at all, as in a rating scale where everyone ticked the top box – counts as **not** normal, so the decision errs towards the non-parametric test rather than letting an untestable group pass silently.

Testing assumptions with significance tests has well-known limits (low power in small samples, trivial deviations flagged in large ones); report the check, and treat it as one input to the choice rather than the whole of it.

Value

`TRUE` if no test rejects normality and every group could be tested, `FALSE` otherwise. Attributes:

`tests` data frame with columns `group` (the condition, or `"differences"` / `"residuals"` within subjects), `n`, `W`, `p_value`, `p_adjusted` and `testable`, e.g. for `assumption_methods_text()`.

`method` `"groupwise"`, `"differences"` or `"residuals"`.

`p_adjust` the correction applied (`"none"` for a single test).

`untestable` the groups that could not be tested.

`dropped_subjects` within subjects: the participants left out for lacking a condition.

For a group with more than 5000 values, Shapiro-Wilk is computed on a random sample of 5000 (a warning is emitted), so the result is only reproducible with a seed set beforehand.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:20, 2), cond = rep(c("A", "B"), each = 20))
d$score <- rnorm(40, mean = ifelse(d$cond == "A", 5, 6))
check_normality_by_group(d, "cond", "score")                  # between
check_normality_by_group(d, "cond", "score", subject = "id")  # within
```

---

check_questionnaire *Show how a questionnaire will be scored, before scoring it*

Description

Prints the mapping `score_questionnaire()` would use – which column supplies which item, which subscale it loads on, whether it is reverse-coded – along with the observed range of each column and the instrument's scoring notes. Run it once per study: it is the cheapest available check against the failure mode in which a shifted survey export produces perfectly plausible, wrong scores.

Usage

```
check_questionnaire(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
```

```
  reverse_items = NULL,
  unreverse_items = NULL
)
```

Arguments

`data`	A data frame with one row per respondent (or per respondent and condition) and one column per item.
`instrument`	Instrument key, e.g. `"sus"`, `"nasa_tlx"`, `"ueq_s"`, `"tia"`, `"ssq"`. See `list_questionnaires()`.
`items`	Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (`c(mental = "tlx_md", effort = "tlx_ef", ...)`), which is order-proof and the safer choice for a survey export you did not lay out yourself.
`prefix`	Optional column-name prefix selecting the item columns, e.g. `"sus_"`. Must select exactly as many columns as the instrument has items. The columns are then matched to items *by their names*, never by sort order: either what follows the prefix is the item's code, label or (for a single-item subscale) subscale name, compared without case or punctuation (`tlx_mental`, `tlx_Mental_Demand`, `tia_rc1`, `ipq_SP2`); or it carries one item number per column, and those numbers are exactly `1..n` (`sus_2`, `SUS[2]`, `Q5_2`; in `SUS_2_1` the number that varies across the columns is the item). Anything else – numbering from 0, numbers that do not run `1..n`, names that only partly match – is an error asking for a named `items` mapping, rather than a guess.
`scale`	Optional two-element vector giving the response range your survey used, e.g. `c(1, 21)` for the 21-point NASA-TLX sheet or `c(0, 4)` for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale. Without `scale`, a coding that fits the range but looks shifted against it draws a warning (see Details); passing `scale` explicitly confirms the coding and silences it.
`reverse_items`	Optional items to reverse-code in addition to the scoring key's own, given as item numbers or item codes – for a survey that printed a pair the other way round. For backward compatibility it *toggles*: naming an item the key already reverses un-reverses it, and because passing an instrument's own reverse set (e.g. `c(2, 4, 6, 8, 10)` for the SUS) is a common mistake that would do exactly that, it raises a warning. Use `unreverse_items` to undo a key reversal on purpose.
`unreverse_items`	Optional items the scoring key reverses that should *not* be reversed, because the export already stored them reverse-coded. Naming an item the key does not reverse is an error.

Details

Problems that `score_questionnaire()` would stop on (a column that is not numeric, responses outside the assumed range) are reported here as warnings instead, together with the warnings scoring would raise (a coding that looks shifted, an item named in `reverse_items` that the key already reverses), so the check can explain them rather than abort.

Value

Invisibly, a data frame with one row per item: the item number and code, the column it maps to, its subscale, whether it is reverse-coded, and the observed minimum, maximum and number of missing values of that column.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself. A `prefix` maps columns by what their names say – an item code or label, or one item number per column running 1 to the number of items – never by sort order, and stops when the names do not identify the items. Note that the `attrakdiff` key is blocked by dimension with the negative pole first; data stored as answered on the official AttrakDiff sheet belong to `attrakdiff_official`.

See Also

`score_questionnaire()`, `questionnaire_items()`

Examples

```
set.seed(1)
d <- as.data.frame(matrix(sample(1:5, 10 * 4, TRUE), nrow = 4))
names(d) <- paste0("sus_", 1:10)
check_questionnaire(d, "sus", prefix = "sus_")
```

`cite_methods` *Citations and methods boilerplate for the analyses used*

Description

Prints a ready-made methods phrase plus the BibTeX entries for the R packages behind the requested analysis methods, so a manuscript's methods section and bibliography can be filled in one step.

Usage

```
cite_methods(methods = c("ggstatsplot", "effectsize"), bibtex = TRUE)
```

Arguments

methods	Character vector of analysis methods to cite. Any of "art" (Aligned Rank Transform via ARTool), "dunn" (Dunn's test via FSA), "nparld" (nparLD), "ggstatsplot", "effectsize", and "colleyrstats" (this package).
bibtex	whether to include the BibTeX entries. Default TRUE.

Value

Invisibly returns the generated lines as a character vector; the text is also emitted via message(). Methods whose package is not installed are skipped with a message. Every BibTeX entry gets a citation key – the package name, with -2, -3, ... for a package's further entries – because R's citation entries carry none, and BibTeX keeps only the first of several keyless entries ("Repeated entry"). The methods phrase then cites those keys (\cite{ARTool,ARTool-2}).

Examples

```
cite_methods("ggstatsplot", bibtex = FALSE)
```

---

classify_outcome *Classify the measurement scale of an outcome variable*

Description

Decides how a dependent variable should be modelled by inspecting its type and distribution of values. The measurement scale is the first branch of a principled model choice: it dictates the *family* (Gaussian, binomial, Poisson, cumulative-link) before any distributional assumption is checked.

Usage

```
classify_outcome(y, ordinal_max_levels = 7L)
```

Arguments

y	The outcome vector.
ordinal_max_levels	Integer. Integer-valued numerics with at most this many distinct values are treated as ordinal (Likert-like). Default 7.

Details

The rules are deliberately simple and transparent:

- ordered factor → `"ordinal"`;
- logical, a factor with two observed levels, a character vector with two distinct values, or a numeric coded `0`/`1` → `"binary"`;
- unordered factor/character with more than two levels → `"nominal"`;
- a numeric with at most two distinct values that are *not* `0`/`1` – a 1–7 item on which only 6 and 7 were ticked, or a yes/no item coded 1/2 – is *not* taken for binary: it is classified as `"ordinal"` (whole numbers) or `"continuous"`, with a warning;
- integer-valued numeric with at most `ordinal_max_levels` distinct values (a Likert-type item) → `"ordinal"`;
- non-negative whole numbers that are all multiples of 5 within $[0, 100]$ (the shape of a raw NASA-TLX subscale or a 0–100 slider in steps of 5) → `"continuous"`, with a message;
- any other non-negative whole numbers → `"count"`. This is always announced with a message that says how to override it, and the message becomes a *warning* when the values lie in a closed range without zeros ($[1, 100]$) – the typical shape of a summed questionnaire score or a 1–10 rating, which a Poisson model would misdescribe;
- any other numeric → `"continuous"`.

The heuristics can never be perfect (a 1–7 Likert item and a small count are genuinely ambiguous); pass an explicit `outcome_type` to `recommend_test()` or `fit_recommended()` when you want to override them.

Value

A single string, one of `"continuous"`, `"ordinal"`, `"binary"`, `"count"`, or `"nominal"`.

Examples

```
classify_outcome(rnorm(50)) # "continuous"
classify_outcome(factor(sample(1:5, 50, TRUE), ordered = TRUE)) # "ordinal"
classify_outcome(sample(0:1, 50, TRUE)) # "binary"
classify_outcome(rpois(50, 3)) # "count", with a message
```

`colleyRstats_setup` *Configure Global R Environment for colleyRstats*

Description

Sets the ggplot2 theme, and on request the `conflicted` preferences, to match the standards used in the colleyRstats workflow.

Usage

```
colleyRstats_setup(
  set_options = FALSE,
  set_theme = TRUE,
  base_size = 17,
  set_conflicts = FALSE,
  print_citation = TRUE,
  verbose = TRUE
)
```

Arguments

set_options	**[Deprecated]** Ignored. colleyRstats no longer changes global options(); passing TRUE warns and does nothing. Default is FALSE.
set_theme	Logical. If TRUE, sets the default ggplot2 theme to see::theme_lucid with custom modifications. Default is TRUE.
base_size	Base font size in points for the theme. Every text element is sized relative to it (see colley_theme()), so a figure destined for a 3.33 in column and one destined for a slide can share the same theme and differ only in this number. Default 17 reproduces the sizes used before this argument existed. save_paper_figure() overrides it per figure to match the width actually being written.
set_conflicts	Logical. If TRUE, registers this package's conflicted preferences, which favor dplyr and other tidyverse packages. Default is FALSE. Registering them activates conflicted, which takes over library() for the rest of the session, so it is opt-in and belongs after your library() calls – see Details.
print_citation	Logical. If TRUE, prints the citation information for this package. Default is TRUE.
verbose	Logical. If TRUE, emit informational messages. Default is TRUE.

Details

set_conflicts **changes what** library() **does, so call it last.** conflicted::conflict_prefer() does not merely record a preference: it activates conflicted, which attaches a .conflicts environment carrying its own library() and require() shims. Every later attach in the session goes through them, and they forward quietly and verbose into base::library() as unevaluated symbols. A package whose .onAttach inspects the calling library() frame – meta-packages that attach their own constituents do this to decide whether to print a banner – then evaluates those symbols in a frame that does not bind them, and the attach fails outright:

```
colleyRstats_setup(set_conflicts = TRUE)
library(easystats)
#> Error: .onAttach failed in attachNamespace() for 'easystats':
#>   object 'quietly' not found
```

The two mechanisms do not compose, and colleyRstats_setup() cannot know whether more library() calls are coming. The preferences are therefore opt-in, and the contract when you opt in is that colleyRstats_setup() runs *after* everything else is attached:

```
library(colleyRstats)
library(easystats)
library(dplyr)
colleyRstats_setup(set_conflicts = TRUE)   # last
```

That is also the ordering in which the preferences are worth most: `conflicted` resolves only those names that are ambiguous among the packages attached at the time, so a call made before the rest of the script's `library()` calls has less to work with.

Value

Invisibly returns `NULL`.

Examples

```
# Runs everywhere, no extra packages, no session side effects
colleyRstats::colleyRstats_setup(
  set_options = FALSE,
  set_theme = FALSE,
  set_conflicts = FALSE,
  print_citation = FALSE,
  verbose = FALSE
)


# Full setup (requires suggested packages; changes session defaults)
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("see", quietly = TRUE)) {
  local({
    old_theme <- ggplot2::theme_get()
    on.exit(ggplot2::theme_set(old_theme), add = TRUE)

    colleyRstats::colleyRstats_setup(
      set_options = FALSE,
      set_conflicts = FALSE,   # the default; see Details
      print_citation = FALSE,
      verbose = TRUE
    )

    ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
      ggplot2::geom_point()
  })
}
```

`colley_theme` *The colleyRstats ggplot2 theme*

Description

see::theme_lucid() with the sizing and legend conventions used across colleyRstats figures. Every text element is expressed as a multiple of base_size rather than in absolute points, which is what lets the same theme serve a 3.33 in journal column and a full-width figure: only base_size changes.

Usage

```
colley_theme(base_size = 17, base_family = "")
```

Arguments

base_size	Base font size in points. Default 17, which reproduces the sizes colleyRstats used before this function was factored out.
base_family	Base font family, passed to see::theme_lucid().

Details

Sizing text in absolute points is the usual reason a figure comes out unreadable. Point sizes do not shrink when the canvas does, so a theme tuned on a large canvas puts 20 pt axis titles on a 3.33 in figure, where they overrun the panel and the axis labels collide. Choose base_size from the size the figure is finally PLACED at – roughly the body-text size of the document, or a point or two below it. save_paper_figure() does this for you.

The multipliers are, relative to base_size: axis titles 1.15, axis text 1.0, plot title 1.65, subtitle 1.0, legend text 0.9, caption 0.8, strip text 1.3.

Value

A ggplot2 theme object.

Examples

```
if (requireNamespace("see", quietly = TRUE)) {
  ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
    ggplot2::geom_point() +
    colley_theme(base_size = 8)   # sized for a single journal column
}
```

---

debug_contr_error *Debug contrast errors in ANOVA-like models*

Description

Debug contrast errors in ANOVA-like models

Usage

```
debug_contr_error(dat, subset_vec = NULL)
```

Arguments

dat	A data frame of predictors.
subset_vec	Optional logical or numeric index vector used to subset rows before checks.

Value

A list with two elements:

nlevels Integer vector giving the number of levels for each factor variable in dat.

levels List of factor level labels for each factor variable in dat.

Examples

```
dat <- data.frame(
  group = factor(rep(letters[1:3], each = 3)),
  score = rnorm(9)
)

debug_contr_error(dat = dat)
```

---

define_questionnaire *Register your own questionnaire*

Description

Adds an instrument to the registry so that score_questionnaire(), check_questionnaire() and score_reliability() handle it exactly like a built-in one. Use it for a lab-specific scale, a translated or shortened form, or a published instrument this package does not ship – and put the call in a project's setup script so every analysis in that project scores it the same way.

Usage

```
define_questionnaire(
  key,
  name,
  scale,
  subscale,
  code = NULL,
  label = NULL,
  reverse = integer(0),
  recode = c("none", "center", "zero_base"),
  total = NULL,
```

```
  total_name = "Total",
  reference = NA_character_,
  higher = NA_character_,
  notes = character(),
  integer_responses = TRUE
)
```

Arguments

key	Short identifier, used as the `instrument` argument.
name	Full name of the instrument, shown in messages and listings.
scale	Two-element vector giving the response range, e.g. `c(1, 7)`.
subscale	Character vector, one entry per item, naming the subscale that item loads on. Use a comma-separated string (`"Nausea,Oculomotor"`) for an item that loads on two.
code	Optional short code per item, used to refer to items in `reverse_items` and in a named `items` mapping. Defaults to `item1`, `item2`, ...
label	Optional item wording, one entry per item.
reverse	Optional items that are reverse-scored, as item numbers (`c(2, 4)`) or item codes (`c("item2", "item4")`).
recode	One of `"none"` (default), `"center"` (subtract the midpoint, giving the $-3..+3$ coding of a 7-point semantic differential), or `"zero_base"` (subtract the minimum).
total	How to form an overall score across all items: `"mean"`, `"sum"`, or `NULL` (default) for none – which is the honest choice for a multidimensional instrument whose authors define no total.
total_name	Column name for that overall score. Default `"Total"`.
reference	Optional citation, shown alongside the instrument.
higher	Optional one-word note on what a high score means, e.g. `"better"` or `"worse"`.
notes	Optional character vector of scoring notes.
integer_responses	Logical. `TRUE` (default) for an instrument answered in whole scale points (Likert items, semantic differentials): fractional responses then draw a warning, and a coding that looks shifted against `scale` is checked for. Set `FALSE` for a visual-analogue scale or slider.

Value

Invisibly, the instrument definition.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself. A `prefix` maps columns by what their names say – an item code or label, or one item number per column running 1 to the number of items – never by sort order, and stops when the names do not identify the items. Note that the `attrakdiff` key is blocked by dimension with the negative pole first; data stored as answered on the official AttrakDiff sheet belong to `attrakdiff_official`.

See Also

`list_questionnaires()`, `score_questionnaire()`

Examples

```
define_questionnaire(
  key = "acceptance",
  name = "Van der Laan acceptance scale",
  reference = "Van der Laan, Heino & De Waard (1997), Transp. Res. C 5(1)",
  scale = c(-2, 2),
  subscale = c(
    "Usefulness", "Satisfying", "Usefulness", "Satisfying", "Usefulness",
    "Satisfying", "Usefulness", "Satisfying", "Usefulness"
  ),
  label = c(
    "useful - useless", "pleasant - unpleasant", "bad - good",
    "nice - annoying", "effective - superfluous", "irritating - likeable",
   "assisting - worthless", "undesirable - desirable", "raising alertness - sleep-inducing"
  ),
  reverse = c(1, 2, 4, 5, 7, 9),
  higher = "better"
)
list_questionnaires()[1, ]
```

`define_result_macro` *Define a named LaTeX macro for a single result (single source of truth)*

Description

Emits `\newcommand{\<name>}{<value>}` so you can write `\<name>` in your prose and have it always reflect the latest analysis – re-run the R code and the number updates everywhere it is referenced, the gold standard for reproducible manuscripts. The `name` is sanitised to a letters-only LaTeX command name: other characters start a new camel-case word and digits are spelled out,

so "tlx_1" and "tlx_2" become \tlxOne and \tlxTwo rather than colliding. A name that would redefine an existing LaTeX or **colleyRstats** command (e.g. "p", which is the package's own \p, or "time") is refused with an error that suggests an alternative.

Usage

```
define_result_macro(name, value, path = NULL)
```

Arguments

name	A label for the result, e.g. "tlx_mental_omnibus" (becomes \tlxMentalOmnibus).
value	The rendered result string, e.g. "F(2, 57) = 4.50, p = .02". It is inserted verbatim (already-formatted LaTeX), not escaped. Several strings are joined with spaces, and line breaks become spaces, so each definition occupies exactly one line.
path	Optional .tex path. When it already exists, the definition is added to it (so many results can accumulate in one file); a definition of the same command already in the file is replaced rather than duplicated, because a second \newcommand for one name stops the build. Each line ends in a % comment naming the label it came from, so that a different label mapping to the same command ("tlx1" and "tlx_1" are both \tlxOne) is reported when it replaces the other's result.

Value

Invisibly, a named character scalar: the \newcommand line, named by the generated command. Also emitted via message().

Examples

```
define_result_macro("tlx_mental_omnibus", "F(2, 57) = 4.50, p = .02")
```

---

emit_name_macros *Generate \newcommand stubs for variable/factor names*

Description

The report functions can emit variable and factor-level names as LaTeX commands (e.g. \Video) so their typography is controlled centrally. This writes the matching \newcommand definitions so those commands are never undefined – the classic "Undefined control sequence" that stops an Overleaf build. A name gets a macro only when it can safely become a new command: letters only, and not already a LaTeX command. time, L or small, for instance, are TeX/LaTeX commands already, so \newcommand{\time} would stop the build ("Command \time already defined") and redefining them would break LaTeX itself; names starting with end are refused by LaTeX too. Such names are skipped with a warning, and the reporters emit them as escaped plain text instead.

Usage

```
emit_name_macros(vars, path = NULL, labels = NULL)
```

Arguments

vars	Character vector of variable/level names (e.g. the columns you pass as iv/dv), or a named character vector / list mapping a name to the display label it should expand to (unnamed elements are their own label).
path	Optional .tex/.sty path to write the definitions to.
labels	Optional named character vector mapping a name to its display label (overrides names taken from vars).

Value

Invisibly, the \newcommand lines as a character vector (one per distinct name); also emitted via message().

Examples

```
emit_name_macros(c("Video", "DriverPosition"))
emit_name_macros(c(tlxMental = "TLX Mental Demand"))
```

---

emit_overleaf *Bundle an analysis into an Overleaf-ready folder*

Description

Writes everything a manuscript needs into one directory you can drag into Overleaf and compile immediately: a main.tex that already \inputs the results, one .tex per result section, the figures, a references.bib, and – unless macros are expanded inline – colleyRstats.sty. This is the one-call end of the "R analysis to compiled PDF" pipeline.

Usage

```
emit_overleaf(
  x,
  dir,
  figures = TRUE,
  methods = c("ggstatsplot", "effectsize"),
  title = "Results",
  plain = NULL,
  columns = 1,
  overwrite = FALSE
)
```

Arguments

x	What to emit. Accepts a report_all() result (one section per dependent variable, with figures), an analyze_and_report() result, a named list of sentence vectors, or a single character vector.

dir	Output directory (created if needed).
figures	Whether to save figures for sections that carry a plot. Default TRUE.
methods	Methods to cite (passed to cite_methods()) for references.bib; NULL to skip the bibliography. Every entry is listed via \nocite{*}, since the generated text cites nothing and BibTeX writes an empty bibliography without a citation. When none of the methods can be cited (their packages are not installed), no references.bib is written, with a warning.
title	Title used in the generated main.tex. Default "Results".
plain	Whether to expand the colleyRstats macros to plain LaTeX (so no .sty / \usepackage is needed). Default NULL follows getOption("colleyRstats.macros") (i.e. plain when that is FALSE).
columns	Figure width preset passed to save_paper_figure(). A single-column figure is included at its natural size, because save_paper_figure() chose its type size for exactly that width (scaling it to the line width would enlarge the text with it); a full-width figure (columns = 2) goes into a figure* at \textwidth.
overwrite	Overwrite existing files in dir? Default FALSE: if any file the project consists of already exists, nothing is written.

Details

Section keys become file names with only letters, digits and hyphens kept ("tlx_1" is written to sections/tlx-1.tex); keys that would share a file name – including names differing only in case, which Windows and macOS treat as one file – get a numbered suffix, so no section can overwrite another. The keys themselves are escaped with latex_escape() wherever they are typeset (headings, captions). Name macros in the text (\Video) get \providecommand stubs in names.tex, and a {} where a space follows them, because TeX would otherwise swallow that space ("\cyl on" typesets as "cylon").

Value

Invisibly, a list with the paths written (dir, main, results, sections, figures, bib, sty, names); bib, sty and names are NULL when the file was not needed.

Examples

```
out <- report_all(mtcars, dvs = c("mpg", "disp"), iv = "cyl")
emit_overleaf(out, dir = file.path(tempdir(), "paper"), overwrite = TRUE)
```

---

expand_latex_macros *Expand the colleyRstats LaTeX macros to plain standard LaTeX*

Description

The report functions normally emit compact custom macros (\F, \p, \m, \sd, \df, \chisq, \padj, \padjminor, \pminor, \rankbiserial, \effectsize) that require latex_preamble() definitions. This expands them into equivalent plain math (e.g. \F{2}{57}{4.50} becomes $F(2, 57) = 4.50$) so the text compiles in any document with no custom preamble – the "zero-setup Overleaf" path. It is applied automatically by the sink_to/emit_overleaf() writers when options(colleyRstats.macros = FALSE).

Usage

```
expand_latex_macros(x)
```

Arguments

x	A character vector of report text.

Value

The text with the macros expanded to standard LaTeX math.

Examples

```
expand_latex_macros("A significant effect (\\F{2}{57}{4.50}, \\p{0.012}).")
```

---

figure_base_size *Base font size for a figure of a given width*

Description

The rule save_paper_figure() uses to pick a type size from a figure width. Text in a figure should read at roughly the body-text size of the document the figure is placed in; since a figure is usually placed at 100\ means the type size has to follow the physical width. The rule is calibrated so that the two standard widths land on sensible values: 3.33 in (a two-column journal column) gives 7 pt and 7 in (full text width) gives 9 pt, with linear interpolation between and clamping outside.

Usage

```
figure_base_size(width, min_size = 6, max_size = 12)
```

Arguments

width	Figure width in inches.
min_size, max_size	Bounds, so that very small or very large figures still get a usable size. Defaults 6 and 12 points.

Value

A single number: the base font size in points.

Examples

```
figure_base_size(3.33)  # 7
figure_base_size(7)     # 9
```

`fit_recommended` *Fit the model that the data call for, and report it*

Description

Runs `recommend_test()` to choose an analysis, then actually fits it: coerces the outcome and predictors into the classes that model family needs, builds the random-effect structure for a repeated-measures design, fits, computes post-hoc contrasts with the matching machinery, and produces the APA/LaTeX sentence via this package's reporter for that model family.

Usage

```
fit_recommended(
  data,
  outcome,
  predictors = NULL,
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L,
  interaction = TRUE,
  contrasts = TRUE,
  adjust = "holm",
  sink_to = NULL,
  verbose = TRUE,
  factors = NULL,
  random_slopes = NULL
)
```

Arguments

`data`	The data frame.
`outcome`	The dependent variable (column name as string).
`predictors`	Character vector of independent variable column names.
`cluster`	Optional column identifying the subject or cluster for repeated-measures data – the random-effect grouping factor.
`design`	One of `"auto"` (default; clustered when some `cluster` has several rows), `"between"`, or `"within"`.

`outcome_type` "auto" (default; use `classify_outcome()`) or an explicit "continuous", "ordinal", "binary", "count", "nominal" to override the classification. Pass it when the automatic choice is wrong – a 1-7 Likert item and a small count are genuinely ambiguous from the data alone.

`ordinal_max_levels` Passed to `classify_outcome()`. Default 7.

`interaction` Logical. With more than one predictor, fit the full factorial model (TRUE, default) or main effects only (FALSE). The aligned rank transform always needs the full model, so FALSE is refused on that route.

`contrasts` Logical. Compute post-hoc contrasts between the levels of the categorical predictors. Default TRUE.

`adjust` Multiplicity adjustment for those contrasts, passed to **emmeans** or **ARTool**. Default "holm".

`sink_to` Optional path of a .tex file; the methods sentence and the result sentence are written there (LaTeX-escaped) so a manuscript can \input{} them.

`verbose` Logical. If TRUE (default), report every coercion, the model being fitted, and every simplification of the random effects.

`factors` Optional character vector naming the numeric predictors that are categorical (condition codes); see `recommend_test()`. NULL (default) decides from the data, and every resulting coercion is announced.

`random_slopes` Optional character vector of by-cluster random-slope predictor names; NULL selects slopes from the design and `character(0)` requests random intercepts only. See `recommend_test()`.

Details

The point is that the test you justify in the methods section and the model you actually ran are produced by the same call, from the same data, and therefore cannot drift apart – the failure mode of a pipeline where `recommend_test()` prints advice that someone then re-types by hand.

Value

An object of class "colley_fit": a list with `recommendation` (the `recommend_test()` object, updated to record the function, call and random effects actually used), `formula`, `model` (the fitted object), `data` (the final analyzed data, with model coercions and any incomplete-participant exclusions applied), `engine`, `model_function` (the function actually called, e.g. "lmerTest::lmer"), `random` (the random-effects term kept), `singular` and `converged` (flags), `anova` (the omnibus table: term, df1, df2, statistic, stat_name, p), `emmeans` (a named list of the estimated marginal means grids) and `contrasts` (a data frame with one row per comparison: `term`, `by`, `contrast`, `estimate`, `scale`, `SE`, `df`, confidence limits, `statistic`, `p.value`, `adjust`; or NULL), `methods` (plain text), `methods_tex` (LaTeX-escaped), `text` (result sentences, LaTeX), and `sentences` (`methods_tex` and `text`, in manuscript order – what `sink_to` writes). A `print` method summarises it.

What it can fit

Cumulative link models with and without random effects (**ordinal**), generalized and linear mixed models (**lme4**, **lmerTest**, **glmmTMB** for negative-binomial counts), Poisson, negative-binomial

(**MASS**) and logistic GLMs, Type III factorial ANOVA (**afex**, or **car**), heteroscedasticity-robust (HC3) factorial ANOVA (**car**), the aligned rank transform (**ARTool**), rank-based repeated measures (**nparLD**), multinomial regression (**nnet**) and its mixed counterpart (**mclogit**), and the classical one-way ANOVA / Welch / Kruskal-Wallis / Wilcoxon tests. Model packages live in `Suggests`: a branch you use needs its package installed, and says which one if it is not.

Random effects

A repeated-measures model starts from the maximal random-effects structure the design supports (Barr et al., 2013): by-cluster random slopes for every within-cluster factor whose levels are replicated within a cluster, numeric covariates with enough repeated observations to estimate a slope, and the interaction slope when there are several trials per cluster and cell. When that fit is singular or does not converge, the structure is simplified step by step down to random intercepts; what was dropped, and why, is announced and written into `$methods`. A singular final fit is flagged in `$singular`.

Post-hoc contrasts

Each categorical predictor gets its own family of pairwise comparisons of its marginal means (averaged over the other factors), adjusted with `adjust` within that family. Simple effects – one factor compared within each level of the others – are added only for interactions that are in the model and significant in the Type III omnibus test.

References

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language, 68*(3), 255–278.

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language, 94*, 305–315.

See Also

`recommend_test()` for the choice alone, `plot_fit()` for a figure using this model's tests, `analyze_and_report()` for the figure-plus-sentence path through **ggstatsplot**.

Examples

```
set.seed(1)
d <- data.frame(
  id = factor(rep(1:20, each = 3)),
  cond = factor(rep(c("A", "B", "C"), times = 20)),
  score = rnorm(60)
)
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- fit_recommended(d, outcome = "score", predictors = "cond", cluster = "id")
  fit
}
```

`generateEffectPlot` *Function to define a plot, either showing the main or interaction effect in bold.*

Description

Function to define a plot, either showing the main or interaction effect in bold.

Usage

```
generateEffectPlot(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
  numberColors = lifecycle::deprecated(),
  subject = NULL
)

plot_effect(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
  numberColors = lifecycle::deprecated(),
  subject = NULL
)
```

Arguments

`data`	the data frame
`x`	factor shown on the x-axis
`y`	dependent variable
`fillColourGroup`	group to color
`ytext`	label for y-axis
`xtext`	label for x-axis
`legendPos`	position for legend
`legendHeading`	custom heading for legend
`shownEffect`	either "main" or "interaction"
`effectLegend`	TRUE: show legend for effect (Default: FALSE)
`effectDescription`	custom label for effect
`xLabelsOverwrite`	custom labels for x-axis
`useLatexMarkup`	use latex font and markup
`numberColors`	**[Deprecated]** Never had any effect: the colours come from `see::scale_colour_see()`, which uses one colour per group. Passing it warns.
`subject`	optional participant-ID column (a string). Give it when `x` and/or `fillColourGroup` vary within participants, to get within-subject error bars (see Details). Default `NULL`.

Details

Points and lines. The small points and the dashed (main) or bold (interaction) per-group lines are the cell means. The bold main-effect line and its large points are the *unweighted* marginal means of `x`: for each level of `x`, the mean of the cell means across `fillColourGroup`. That is the quantity a main effect in a factorial ANOVA is about; the mean of the raw observations would weight each cell by its size and, in an unbalanced design, follow whichever group happens to be larger.

Error bars are 95\

- `subject` = `NULL` (default): nonparametric bootstrap intervals (`ggplot2::mean_cl_boot()`, requires 'Hmisc'). These treat every observation as independent, i.e. they are **between-subject** intervals. That is right for between-subject factors, but for a factor that varies within participants they include the between-participant spread a within-subject comparison does not depend on, and look misleadingly wide.
- `subject` given: **within-subject** intervals after Cousineau (2005), with the Morey (2008) correction. Each participant's scores are centred on their own mean, the variance of the normalised scores is inflated by $M/(M-1)$ (M = number of within-subject conditions), and a factor that does not vary within participants is respected by normalising within its groups. Participants without a value in every within-subject condition are dropped (with a message), and more than one row per participant and cell is an error.

Either way, the intervals describe single means, not differences between them; overlapping bars do not by themselves imply a non-significant difference.

Value

a plot

Naming

`plot_effect()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`generateEffectPlot()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

References

Cousineau, D. (2005). Confidence intervals in within-subject designs: A simpler solution to Loftus and Masson's method. *Tutorials in Quantitative Methods for Psychology, 1*(1), 42–45.

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to Cousineau (2005). *Tutorials in Quantitative Methods for Psychology, 4*(2), 61–64.

Examples

```
set.seed(123)
main_df <- data.frame(
  strategy    = factor(rep(c("A", "B"), each = 20)),
  Emotion     = factor(rep(c("Happy", "Sad"), times = 20)),
  trust_mean  = rnorm(40, mean = 5, sd = 1)
)

generateEffectPlot(
  data = main_df,
  x = "strategy",
  y = "trust_mean",
  fillColourGroup = "Emotion",
  ytext = "Trust",
  xtext = "Strategy",
  legendPos = c(0.1, 0.23)
)
```

---

generateMoboPlot *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases.

Usage

```
generateMoboPlot(
  data,
  x,
  y,
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)

plot_mobo(
  data,
  x,
  y,
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)
```

Arguments

`data`	A data frame containing the data to be plotted.
`x`	A string representing the column name in `data` to be used for the x-axis. Can be either numeric or a factor whose labels are numbers; it is plotted on a continuous axis either way, so the data, the phase guides and the fitted equation all refer to the iteration values (not to factor positions).
`y`	A string representing the column name in `data` to be used for the y-axis. This should be a numeric variable.
`fillColourGroup`	A string representing the column name in `data` that defines the fill color grouping for the plot. Default is `"ConditionID"`.
`ytext`	A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of `y`.
`legendPos`	A numeric vector of length 2 specifying the position of the legend inside the plot. Default is `c(0.65, 0.85)`.
`numberSamplingSteps`	An integer specifying the number of initial sampling steps before the optimization phase begins, counted in iteration steps from the first iteration (the spacing being the smallest gap between iterations), so it also works when iterations start at 0 or are spaced by more than 1, and a missing iteration does not move the

boundary. A value that covers every iteration in the data gives a warning. Default is 5.

`labelPosFormulaY`
: A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are `"top"`, `"center"`, or `"bottom"`. Default is `"top"`.

`verticalLinePosY`
: A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn.

Value

A `ggplot` object representing the multi-objective optimization plot, ready to be rendered.

Naming

`plot_mobo()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`generateMoboPlot()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y")


# Example with factor x-axis
df <- data.frame(
  x = factor(rep(1:5, each = 4)),
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y", numberSamplingSteps = 3)
```

`generateMoboPlot2` *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases. Appropriate if you use https://github.com/Pascal-Jansen/Bayesian-Optimization-for-Unity in version 1.1.0 or higher.

Usage

```
generateMoboPlot2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
  fillLabels = NULL,
  annotationTextSize = 5
)

plot_mobo2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
  fillLabels = NULL,
  annotationTextSize = 5
)
```

Arguments

`data`	A data frame containing the data to be plotted.

x — A string representing the column name in data to be used for the x-axis. Can be either numeric or a factor whose labels are numbers; it is plotted on a continuous axis either way, so the data, the phase guides and the fitted equation all refer to the iteration values (not to factor positions). Default is "Iteration".

y — A string representing the column name in data to be used for the y-axis. This should be a numeric variable.

phaseCol — the name of the column for the color of the phase (sampling or optimization)

fillColourGroup — A string representing the column name in data that defines the fill color grouping for the plot. Default is "ConditionID".

ytext — A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of y.

legendPos — A numeric vector of length 2 specifying the position of the legend inside the plot. Default is c(0.65, 0.85).

labelPosFormulaY — A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are "top", "center", or "bottom". Default is "top".

labelPosFormulaX — A string specifying the position of the polynomial equation label in the plot. Acceptable values are "left", "center", or "right". Default is "left".

horizontalLinePosY — A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn. Default is 0.75

horizontalLineDistToText — A numeric value of the y-coordinate where the "sampling" and "optimization" text should be drawn below the line. Default is 0.3

fillLabels — An optional named character vector mapping raw factor levels to display labels for the fill/colour legend (e.g. c("value_only" = "Value Only", "llm_only" = "LLM Only")). If NULL (default), the original factor levels are used as-is.

annotationTextSize — numeric. The font size for embedded text annotations inside the plot (e.g., "Sampling", "Optimization" labels, and the regression equations). Default is 5.0.

Value

A ggplot object representing the multi-objective optimization plot, ready to be rendered.

Naming

plot_mobo2() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

generateMoboPlot2() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10),
  Phase = rep(c("Sampling", "Optimization"), 10)
)
generateMoboPlot2(data = df, x = "x", y = "y")
```

`ggbetweenstatsWithPriorNormalityCheck` *Check the data's distribution. If non-normal, take the non-parametric variant of* ggbetweenstats. *x and y have to be in parentheses, e.g., "ConditionID".*

Description

For independent groups: one row per participant. If the group-wise normality check (`check_normality_by_group()`) passes, ggstatsplot runs Welch's t-test (two groups) or Welch's one-way ANOVA with Games-Howell post-hoc tests and labels the means; none of these assume equal variances, which is why no Levene test is run. Otherwise it runs a Wilcoxon rank-sum (Mann-Whitney) or Kruskal-Wallis test with Dunn post-hoc tests and labels the medians. Post-hoc p-values are Holm-adjusted. Rows without a group or an outcome are dropped, and the grouping column is treated as a factor, so numeric condition codes are categories.

Usage

```
ggbetweenstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)

plot_between_stats(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
```

```
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)
```

Arguments

data	the data frame
x	the independent variable, most likely "ConditionID"
y	the dependent variable under investigation
ylab	label to be shown for the dependent variable
xlabels	labels to be used for the x-axis
showPairwiseComp	whether to show the significant pairwise comparisons (TRUE, default) or none (FALSE). With FALSE no pairwise table is computed either, so reportggstatsplotPostHoc() has nothing to report from the plot.
plotType	either "box", "violin", or "boxviolin" (default)

Value

A ggplot object produced by ggstatsplot::ggbetweenstats, which can be printed or further modified with +.

Naming

plot_between_stats() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggbetweenstatsWithPriorNormalityCheck() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy between-subject data: each participant sees one condition
main_df <- data.frame(
  CondID      = factor(rep(c("A", "B", "C"), each = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID",
```

```
  y = "tlx_mental", ylab = "Mental Demand",
  xlabels = labels_xlab,
  showPairwiseComp = TRUE
)
```

`ggbetweenstatsWithPriorNormalityCheckAsterisk` *Check the data's distribution. If non-normal, take the non-parametric variant of* ggbetweenstats. *x and y have to be in parentheses, e.g., "ConditionID".*

Description

As `ggbetweenstatsWithPriorNormalityCheck()`, but the significant comparisons are drawn as asterisk brackets (*** p < .001, ** p < .01,

- p < .05) instead of p-values. The brackets come from the same test as the figure: with two groups, the omnibus test in the subtitle (ggstatsplot runs no post-hoc test then, and a second test of the same pair could disagree with it); with more, the Holm-adjusted Games-Howell (parametric) or Dunn (non-parametric) tests ggstatsplot itself would compute.

Usage

```
ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin"
)

plot_between_stats_asterisk(data, x, y, ylab, xlabels, plotType = "boxviolin")
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggbetweenstats` with additional significance annotations, which can be printed or modified.

Naming

plot_between_stats_asterisk() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggbetweenstatsWithPriorNormalityCheckAsterisk() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy between-subject data: each participant sees one condition
main_df <- data.frame(
  CondID      = factor(rep(c("A", "B", "C"), each = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
  x = "CondID", y = "tlx_mental", ylab = "Mental Demand", xlabels = labels_xlab
)
```

---

ggwithinstatsWithPriorNormalityCheck

Check the data's distribution. If non-normal, take the non-parametric variant of ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Observations are paired by the participant ID in subject, never by row order. Before anything is computed, rows without a condition, outcome or ID are dropped, participants lacking any condition are left out (with a message saying how many and which), and more than one row per participant and condition is an error – aggregate repeated trials first. The normality check (check_normality_by_group() with subject, i.e. on the paired differences or on the residuals of the repeated-measures model), the figure and the pairwise tests all use this same data. The condition column is treated as a factor, so numeric condition codes are categories.

Usage

```
ggwithinstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin",
  subject
)

plot_within_stats(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin",
  subject
)
```

Arguments

`data`	the data frame, in long format (one row per participant and condition)
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`showPairwiseComp`	whether to show the significant pairwise comparisons (`TRUE`, default) or none (`FALSE`). With `FALSE` no pairwise table is computed either, so `reportggstatsplotPostHoc()` has nothing to report from the plot.
`plotType`	either "box", "violin", or "boxviolin" (default)
`subject`	the participant-ID column, as a string (e.g. `"participant"`). Required. It is the last argument so that existing positional calls keep their meaning.

Details

If the check passes, ggstatsplot runs a paired t-test (two conditions) or a repeated-measures ANOVA with paired Student's t post-hoc tests, and labels the means; otherwise a Wilcoxon signed-rank or Friedman test with Durbin-Conover post-hoc tests, and labels the medians. Post-hoc p-values are Holm-adjusted.

Value

A ggplot object produced by ggstatsplot::ggwithinstats with additional significance annotations, which can be printed or modified.

Naming

plot_within_stats() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggwithinstatsWithPriorNormalityCheck() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy within-subject data: every participant sees every condition
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand",
  xlabels = labels_xlab,
  showPairwiseComp = TRUE,
  subject = "Participant"
)
```

---

ggwithinstatsWithPriorNormalityCheckAsterisk

Check the data's distribution. If non-normal, take the non-parametric variant of ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID". Add Asterisks instead of p-values.*

Description

As ggwithinstatsWithPriorNormalityCheck() – including pairing by subject and dropping participants who lack a condition – but the significant comparisons are drawn as asterisk brackets (*** p < .001, ** p < .01, * p < .05) instead of p-values. The brackets come from the same test as the figure: with two conditions, the omnibus paired test in the subtitle; with more, the Holm-adjusted paired Student's t (parametric) or Durbin-Conover (non-parametric) tests ggstatsplot itself would compute, paired by participant.

Usage

```
ggwithinstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin",
  subject
)

plot_within_stats_asterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin",
  subject
)
```

Arguments

data	the data frame, in long format (one row per participant and condition)
x	the independent variable, most likely "ConditionID"
y	the dependent variable under investigation
ylab	label to be shown for the dependent variable
xlabels	labels to be used for the x-axis
plotType	either "box", "violin", or "boxviolin" (default)
subject	the participant-ID column, as a string (e.g. "participant"). Required. It is the last argument so that existing positional calls keep their meaning.

Value

A ggplot object produced by ggstatsplot::ggwithinstats with additional significance annotations, which can be printed or modified.

Naming

`plot_within_stats_asterisk()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`ggwithinstatsWithPriorNormalityCheckAsterisk()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy within-subject data: every participant sees every condition
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand", xlabels = labels_xlab,
  subject = "Participant"
)
```

---

latexify_report *Transform text from* `report::report()` *into LaTeX-friendly output.*

Description

This function transforms the text output from `report::report()` by performing several substitutions to prepare the text for LaTeX typesetting. Every LaTeX special character (`\ { } $ & # _ % ^`) is escaped – `report()` keeps variable and level names such as `tlx_mental` verbatim – `~`, `<` and `>` become `$\sim$`, `$<$` and `$>$`, and `R2` and `Rhat` become `$R^2$` and `$\hat{R}$`. Additionally, it provides options to:

- Omit bullet items marked as "non-significant" (when `only_sig = TRUE`).
- Remove a concluding note about standardized parameters (when `remove_std = TRUE`).
- Wrap bullet items in a LaTeX `itemize` environment or leave them as plain text (controlled by `itemize`).

Usage

```
latexify_report(
  x,
  print_result = TRUE,
  only_sig = FALSE,
  remove_std = FALSE,
  itemize = TRUE
)
```

Arguments

x	Character vector or a single string containing the report text.
print_result	Logical. If TRUE (default), the formatted text is printed to the console.
only_sig	Logical. If TRUE, bullet items containing "non-significant" are omitted. Default is FALSE.
remove_std	Logical. If TRUE, the final standardized parameters note is removed. Default is FALSE.
itemize	Logical. If TRUE (default), bullet items are wrapped in a LaTeX itemize environment; otherwise the bullet markers are simply removed.

Value

A single string with the LaTeX-friendly formatted report text.

Examples

```
if (requireNamespace("report", quietly = TRUE)) {
  # Simple linear model on the iris dataset
  model <- stats::lm(
    Sepal.Length ~ Sepal.Width + Petal.Length,
    data = datasets::iris
  )

  # Format the report output, showing only significant items, removing the
  # standard note, and wrapping bullet items in an itemize environment.
  report_text <- try(report::report(model), silent = TRUE)
  if (!inherits(report_text, "try-error")) {
    latexify_report(
      report_text,
      only_sig = TRUE,
      remove_std = TRUE,
      itemize = TRUE
    )
  }
}
```

latex_escape *Escape LaTeX special characters in plain text*

Description

Makes an arbitrary string safe to drop into a LaTeX document by escaping the characters that would otherwise be interpreted as markup (\ { } $ & # _ % ~ ^ < >). Use it on variable names, factor-level labels, captions – anything user-supplied that reaches the `.tex`. This is what prevents a dependent variable called `tlx_mental` from producing an un-compilable `tlx_mental` (a subscript error) in Overleaf.

Usage

```
latex_escape(x)
```

Arguments

x	A character vector (or something coercible to one).

Value

A character vector with LaTeX specials escaped; `NA` is preserved.

Examples

```
latex_escape("tlx_mental")
latex_escape("cost (%) & margin")
```

latex_preamble *LaTeX preamble required by the report functions*

Description

All report functions emit LaTeX text that relies on a small set of custom commands. This helper prints the complete set, ready to paste into a manuscript preamble, or writes it to a file that can be included with `\input{}` (or renamed to `.sty` and loaded via `\usepackage`).

Usage

```
latex_preamble(path = NULL)
```

Arguments

path	Optional path of a `.tex` file to write the definitions to. If the path ends in `.sty`, a `\ProvidesPackage` header is added so it can be uploaded to Overleaf and loaded with `\usepackage{colleyRstats}` (see also `use_colleyrstats_sty()`).

Value

Invisibly returns the macro definitions as a character vector; the text is also emitted via `message()`.

Examples

```
latex_preamble()
```

`list_questionnaires` *List the questionnaires this package can score*

Description

List the questionnaires this package can score

Usage

```
list_questionnaires()
```

Value

A data frame with one row per instrument: its key (the value to pass as `instrument`), full name, number of items, number of subscales, the response range it assumes, the direction a high score means, and its reference.

See Also

`questionnaire_items()`, `score_questionnaire()`, `define_questionnaire()`

Examples

```
list_questionnaires()
```

`na.zero` *Replace NA values with zero*

Description

Replace NA values with zero

Usage

```
na.zero(x)
```

Arguments

x	A vector.

Value

A vector with NAs replaced by zeros.

Examples

```
na.zero(c(NA, 1, NA, 2))
```

`normalize` *This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].*

Description

This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].

Usage

```
normalize(x_vector, old_min, old_max, new_min, new_max)
```

Arguments

`x_vector`	A numeric vector that you want to normalize.
`old_min`	The minimum value in the original scale of the data.
`old_max`	The maximum value in the original scale of the data.
`new_min`	The minimum value in the new scale to which you want to normalize the data.
`new_max`	The maximum value in the new scale to which you want to normalize the data.

Value

A numeric vector with the normalized values.

Examples

```
normalize(c(1, 2, 3, 4, 5), 1, 5, 0, 1)
```

not_empty *Ensure input is not empty*

Description

Stops execution if x is NULL, empty, contains only NAs, or is a data frame without rows.

Usage

```
not_empty(x, msg = NULL)
```

Arguments

x	The object to check
msg	The error message to display. The default names the offending argument, e.g. "`data` must not be empty.".

Value

Invisible TRUE if valid.

not_in *Negate %in% membership*

Description

Negate %in% membership

Usage

```
not_in(x, y)

x %!in% y
```

Arguments

x	Vector of values to test.
y	Vector of values to match against.

Value

Logical vector indicating non-membership.

n_fun *Build a median/size label for plot annotations*

Description

Build a median/size label for plot annotations

Usage

```
n_fun(x)
```

Arguments

x	A numeric vector.

Value

A data frame with the median and label.

pathPrep *Convert Windows paths to R-friendly format*

Description

Convert Windows paths to R-friendly format

Usage

```
pathPrep(path = "clipboard", read_fn = NULL, write_fn = NULL)
```

Arguments

path	Path to convert or the string "clipboard" to read from the clipboard.
read_fn	Optional custom function to read from the clipboard.
write_fn	Optional custom function to write to the clipboard.

Value

A normalized path string.

plot_fit *Plot observations with the tests from a fitted analysis*

Description

Draws the analyzed observations and annotates them with the model and omnibus tests stored by `fit_recommended()`. No model, assumption check or significance test is run while building the figure. The annotations therefore describe the same fit as the manuscript sentences.

Usage

```
plot_fit(fit, x = NULL, ylab = NULL, xlabels = NULL, show_test = TRUE)
```

Arguments

`fit`	A `colley_fit` returned by `fit_recommended()`.
`x`	Predictor column to display. Defaults to the first predictor; an intercept-only fit is displayed as one group.
`ylab`	Outcome axis label. Defaults to the outcome column name.
`xlabels`	Optional labels for a categorical predictor, in factor order.
`show_test`	Logical. Include the stored omnibus statistics in the subtitle. The model name is always shown.

Details

For a categorical predictor and a numeric continuous/count outcome, repeated observations are averaged per cluster and predictor level for display. These are descriptive means, not model predictions or confidence intervals. Ordinal, binary and nominal responses are shown as counts of observed responses without averaging categories. Numeric predictors are shown as scatterplots. Other predictors remain in the fitted model whose tests are printed in the subtitle.

Value

A ggplot object. It can be saved with `save_paper_figure()` or combined with other plots using **patchwork**.

See Also

`fit_recommended()`

Examples

```
fit <- fit_recommended(iris, "Sepal.Length", "Species",
                       outcome_type = "continuous", verbose = FALSE)
plot_fit(fit)
```

`questionnaire_items` *The items of one questionnaire*

Description

The items of one questionnaire

Usage

```
questionnaire_items(instrument, notes = TRUE)
```

Arguments

`instrument`	Instrument key, e.g. `"tia"`. See `list_questionnaires()`.
`notes`	Logical. If `TRUE` (default), also emit the instrument's scoring notes – how the overall score is formed, what its range means, and where its published form is known to vary between administrations.

Value

Invisibly, a data frame with one row per item: number, code, wording, subscale, and whether the scoring key reverses it.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself. A `prefix` maps columns by what their names say – an item code or label, or one item number per column running 1 to the number of items – never by sort order, and stops when the names do not identify the items. Note that the `attrakdiff` key is blocked by dimension with the negative pole first; data stored as answered on the official AttrakDiff sheet belong to `attrakdiff_official`.

See Also

`check_questionnaire()`, `score_questionnaire()`

Examples

```
questionnaire_items("ueq_s")
```

`recommend_test` *Recommend a principled analysis for one outcome*

Description

Works out, from the data alone, which statistical model is appropriate for a given outcome and set of predictors, and – crucially – *why*. The decision follows a transparent three-question tree:

1. **What is the outcome's measurement scale?** (via `classify_outcome()`: continuous, ordinal, binary, count, nominal.) This fixes the model family. A count is checked for over-dispersion (`performance::check_overdispersion()`, Pearson χ^2/df); an over-dispersed count is modelled as negative binomial (`glmmTMB::glmmTMB(family = nbinom2)` when clustered, `MASS::glm.nb()` otherwise).
2. **Are the observations independent or clustered?** A `cluster` column in which some cluster contributes several rows makes the design clustered, whatever `design` says: the cluster then needs a random effect, i.e. a *mixed* model. For a categorical predictor that varies within clusters and whose levels are replicated within a cluster, or a numeric covariate with enough repeated observations, by-cluster random slopes are added (Barr et al., 2013); `fit_recommended()` simplifies the structure when the fit is singular.
3. **For a continuous outcome, do the parametric assumptions hold?** A Shapiro–Wilk test on the residuals of the model that would be fitted (all predictors, their interactions and the random effects) and, for between-subjects designs, a Brown–Forsythe test (Levene's test centred on the median) across all design cells decide between a parametric model and a rank-based alternative.

Usage

```
recommend_test(
  data,
  outcome,
  predictors = NULL,
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L,
  factors = NULL,
  random_slopes = NULL
)

recommend_analysis(
  data,
  outcome,
  predictors = NULL,
```

```
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L,
  factors = NULL,
  random_slopes = NULL
)
```

Arguments

`data`	The data frame.
`outcome`	The dependent variable (column name as string).
`predictors`	Optional character vector of predictor (independent variable) column names.
`cluster`	Optional column name identifying the subject/cluster for repeated-measures or otherwise non-independent data (the random-effect grouping factor).
`design`	One of `"auto"` (default; clustered when some `cluster` has several rows), `"between"`, or `"within"`. Repeated rows per cluster are modelled with a random effect even with `"between"` – ignoring them would be pseudo-replication.
`outcome_type`	One of `"auto"` (default; use `classify_outcome()`) or an explicit `"continuous"`, `"ordinal"`, `"binary"`, `"count"`, `"nominal"` to override the automatic classification.
`ordinal_max_levels`	Passed to `classify_outcome()`. Default 7.
`factors`	Optional character vector naming the numeric predictors that are categorical (condition codes). `NULL` (default) decides from the data: a numeric predictor is categorical only when it looks like codes – at most 10 consecutive whole numbers starting at 0 or 1, each observed at least twice (and for at least two clusters). Factor, character and logical predictors are always categorical; `factors = character(0)` keeps every numeric predictor numeric.
`random_slopes`	Optional character vector of predictors to include as by-cluster random slopes. `NULL` (default) includes replicated within-cluster factors and numeric covariates that vary within clusters with at least three observations in most clusters. Every named predictor must vary within `cluster`. `character(0)` requests random intercepts only. Unidentifiable or nonconverged structures may still be simplified by `fit_recommended()`.

Details

The recommendation ranges over ordinary ANOVA (Type III sums of squares for factorial designs, via **afex**), heteroscedasticity-robust (HC3) ANOVA, Welch tests, rank-based methods (Kruskal–Wallis + Dunn, Wilcoxon, the Aligned Rank Transform, nparLD), generalized linear models, cumulative link models, multinomial models, and their mixed-model counterparts – linear mixed models (LMM), generalized linear mixed models (GLMM, `lme4`/`glmmTMB`), cumulative link mixed models (CLMM, `ordinal`) and multinomial mixed models (`mclogit::mblogit`). Only models that can actually be fitted to the design are recommended: continuous covariates take the (mixed) regression route, because ART and nparLD accept only categorical predictors; ART is used only for all-factor full-factorial models; nparLD only for a single within-subject factor.

Value

An object of class "colley_recommendation" (a list) with components including outcome_type, clustered, design ("between", "within", "mixed", or "between (clustered)" for between-cluster predictors with repeated rows per cluster), n_obs (complete rows), categorical (which predictors are categorical), within (which predictors vary within clusters), random (the random-effect structure), family, assumptions (normality and homogeneity results with their test statistics), dispersion (for counts), recommendation (human-readable label), model_function (the R function to call, e.g. "ordinal::clmm"), reporter (the matching colleyRstats reporter), fit_call (a ready-to-edit call as a string), alternatives, rationale, methods_text (an APA-style sentence) and methods_tex (the same, escaped for LaTeX). A print method summarises it.

Naming

recommend_test() is the canonical name. recommend_analysis() is an alias for the same function object, kept for backward compatibility.

References

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language, 68*(3), 255–278.

Examples

```
set.seed(1)
d <- data.frame(
  id    = factor(rep(1:20, each = 3)),
  cond  = factor(rep(c("A", "B", "C"), times = 20)),
  score = rnorm(60),
  rating = factor(sample(1:5, 60, TRUE), ordered = TRUE)
)
# Ordinal outcome measured repeatedly within subject -> CLMM
recommend_test(d, outcome = "rating", predictors = "cond", cluster = "id")
# Continuous, between-subjects -> ANOVA or its rank-based fallback
recommend_test(d, outcome = "score", predictors = "cond")
```

---

remove_outliers_REI *Flag suspicious survey responses via the Response Entropy Index (REI)*

Description

Computes each respondent's Response Entropy Index (Tawa, 2021) and flags unusually low or high values. Note that no rows are removed; entries are only flagged via the Suspicious column.

Usage

```
remove_outliers_REI(df, header = FALSE, variables = "", range = c(1, 5))
```

Arguments

df	Data frame containing the data.
header	Which columns enter the computation. TRUE: only the columns named in variables. FALSE (the default, kept for compatibility): *every* column of df – remove ID, timestamp and other non-item columns first. A warning names any column that does not look like a response: with header = FALSE a non-numeric column, and in either mode a column with values outside range.
variables	Which variables to consider when header = TRUE: either a single character string with names separated by commas ("var1,var2") or a character vector (c("var1", "var2")). Names are matched exactly (not as regular expressions), so survey-export names such as "G01Q01[SQ001]" work; names not found in df are ignored with a warning.
range	Numeric vector of length 2 specifying the range of the Likert scale (used to sanity-check the responses). Defaults to c(1, 5).

Details

The REI is the Shannon entropy (base 10) of a respondent's distribution of answers over the response options, $REI_i = -\sum_k p_{ki} \log_{10} p_{ki}$, where p_{ki} is the proportion of the respondent's *answered* items that received option k. Low values indicate overly consistent answering (e.g. straightlining), high values overly scattered answering; either can indicate careless responding. Because the index ignores item content, compute it on the items as answered, before reverse-coding (Tawa, 2021).

Missing responses are left out: proportions are taken over the items a respondent answered, so the same answer pattern yields the same REI however many items were skipped (dividing by the total number of items, as versions before 0.3.0 did, lowered the REI of anyone who skipped items). A respondent who answered nothing gets NA. Responses outside the declared Likert range trigger a warning (they often indicate mis-coded data, or a non-item column) but are still included in the REI computation.

Flags are relative to the sample: each REI is converted to a percentile of a normal distribution with the sample's mean and standard deviation. "Maybe" marks the outer 10% on either side (the preliminary guideline of Tawa, 2021), "Yes" the outer 5%. When all respondents have the same REI the percentiles are undefined: Percentile is NA, no row is flagged, and a warning says so.

Value

A data frame with the calculated REI, the item columns used, Percentile, and a Suspicious flag ("No", "Maybe", "Yes"; NA for a respondent without answers).

References

Tawa, J. (2021). The Response Entropy Index: Comparative assessment of performance and cultural bias across indices of careless responding. *Survey Research Methods, 15*(3), 299–325. doi:10.18148/srm/2021.v15i3.7832

Examples

```
df <- data.frame(
  id = 1:6,
  q1 = c(1, 5, 3, 3, 2, 4), q2 = c(1, 1, 4, 3, 2, 5),
  q3 = c(1, 4, 3, 3, 5, 4), q4 = c(1, 2, 4, 3, 1, 5)
)
# select the item columns; the ID column is no response
result <- remove_outliers_REI(df, TRUE, c("q1", "q2", "q3", "q4"), c(1, 5))
result
```

---

replace_values *Replace values across a data frame*

Description

Replace all occurrences of given values in all columns of a data frame. Factor levels are preserved (and extended by the replacement values), and numeric/logical columns are only touched where a value actually matches, so unrelated entries keep their exact binary representation.

Usage

```
replace_values(data, to_replace, replace_with)
```

Arguments

data	The input data frame to be modified.
to_replace	A vector of values to be replaced within the data frame. This must be the same length as replace_with.
replace_with	A vector of corresponding replacement values. This must be the same length as to_replace.

Value

Modified data frame with specified values replaced.

Examples

```
data <- data.frame(
  q1 = c("neg2", "neg1", "0"),
  q2 = c("1", "neg2", "neg1")
)

replace_values(
  data,
  to_replace = c("neg2", "neg1"),
  replace_with = c("-2", "-1")
)
```

reportART	*Generate the Latex-text based on the ARTool (see https://github.com/mjskay/ARTool). The ART result must be piped into an anova(). Only significant main and interaction effects are reported. P-values are rounded for the third digit. Each effect is accompanied by its partial eta squared, derived from the F statistic and its degrees of freedom with effectsize::F_to_eta2(), and a two-sided 95% confidence interval. A term is reported as an interaction when its name contains ":"; each factor name is rendered as a* \Name *macro only when that is a safe new LaTeX command, and as escaped text otherwise. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).*

Description

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: \newcommand{\F}[3]{$F({#1},{#2})={#3}$} \newcommand{\p}{\textit{p=}} \newcommand{\pminor}{\text

Usage

```
reportART(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_art(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

model	the model of the art
dv	the name of the dependent variable that should be reported
write_to_clipboard	whether to write to the clipboard
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

`report_art()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportART()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE)) {
  set.seed(123)

  main_df <- data.frame(
    tlx_mental = stats::rnorm(80),
    Video      = factor(rep(c("A", "B"), each = 40)),
    gesture    = factor(rep(c("G1", "G2"), times = 40)),
    eHMI       = factor(rep(c("On", "Off"), times = 40)),
    UserID     = factor(rep(1:20, each = 4))
  )

  art_model <- ARTool::art(
    tlx_mental ~ Video * gesture * eHMI +
      Error(UserID / (gesture * eHMI)),
    data = main_df
  )

  model_anova <- stats::anova(art_model)
  reportART(model_anova, dv = "mental demand")
}
```

---

reportArtCon *Report significant ART contrasts (art.con) as LaTeX text*

Description

Companion to `reportDunnTest()` for aligned-rank-transform (ART) models. It extracts the significant pairwise comparisons produced by `ARTool::art.con()` (an **emmeans** contrast grid), describes the groups involved from the raw data, and prints LaTeX-formatted sentences such as "An ART-C post-hoc test (Holm-adjusted) found that ...".

Usage

```
reportArtCon(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
```

```
  paired = FALSE,
  id = NULL,
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed")
)

report_art_con(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed")
)
```

Arguments

ac	the contrast object returned by ARTool::art.con() (or its summary())
data	the raw data frame used to fit the model
iv	independent variable (the contrasted factor)
dv	dependent variable
paired	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to FALSE. When TRUE, id is required.
id	the subject/pairing column, used only when paired = TRUE. Replicate trials per subject and condition are averaged before pairing.
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them
descriptives	which descriptives to print beside each level: "auto" (default, median and IQR for this rank-based test), "mean", "median" or "trimmed" (20% trimmed mean and 20% winsorized SD).

Details

The p-values are taken as-is from the contrast object, i.e. they are already adjusted by whatever adjust was passed to art.con() (e.g. "holm"), and that correction – read from the contrast summary – is named in the sentence. With adjust = "none" the p-values are emitted as \p{}/\pminor{} rather than as p_{adj}. The effect size is the rank-biserial correlation computed from the raw data. ART is most often used for within-subjects designs; pass paired = TRUE together with id (the subject column) to obtain the paired rank-biserial effect size.

Which level is "higher" follows the sign of the contrast estimate (first minus second level, on the aligned-rank scale). Until 0.3.0 it followed the raw means, which can disagree with the test. By default (descriptives = "auto") the levels are described by their median and IQR, which match a rank-based test; "mean" restores *M*/*SD* and warns where the means order two levels against the contrast.

The two levels of each contrast are read from the contrast coefficients, so level names that emmeans rewrites in its labels – numbers ("mode1 - mode2") or names containing `-`, `+`, `*` or `/` ("Both - (Hand-only)") – are handled.

Attention: `ac` must be a pairwise contrast over a single factor `iv` (e.g. `art.con(model, ~ interaction_mode, adjust = "holm")`).

Required commands in LaTeX (all part of `latex_preamble()`): `\padj`, `\padjminor`, `\p`, `\pminor`, `\mdn`, `\iqr`, `\m`, `\sd` and `\newcommand{\rankbiserial}[1]{$r_{rb}$ = #1$}`.

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_art_con()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportArtCon()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
  reportArtCon(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportArtConTable` *Report ART contrasts (art.con) as a LaTeX table. Customizable with sensible defaults. Companion to* `reportDunnTestTable()`.

Description

The table lists the significant contrasts with their test statistic – a column headed *t*, or *z* when emmeans reports z ratios – its degrees of freedom (fractional Kenward-Roger or Satterthwaite dfs are shown to two decimals, not rounded to integers; the column is omitted for z ratios), the p-value and the rank-biserial correlation. The p-value column is headed "p-adjusted" and the caption names the correction read from the contrast summary; with `adjust = "none"` the header is "p" and the caption says the p-values are not adjusted. A positive statistic means the first-named level is higher on the aligned-rank scale.

Usage

```
reportArtConTable(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)

report_art_con_table(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)
```

Arguments

`ac`	the contrast object returned by `ARTool::art.con()` (or its `summary()`)
`data`	the raw data frame used to fit the model
`iv`	independent variable (the contrasted factor)
`dv`	dependent variable

paired	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to FALSE. When TRUE, id is required.
id	the subject/pairing column, used only when paired = TRUE. Replicate trials per subject and condition are averaged before pairing.
orderByP	whether to order by the p value
numberDigitsForPValue	the number of digits to show
latexSize	which size for the text
orderText	whether to order the comparisons alphabetically; ignored when orderByP = TRUE
style	table rule style: "hline" (default) or "booktabs" (\toprule/\midrule/\bottomrule; needs \usepackage{booktabs}).
sink_to	optional path of a .tex file to write the table to, so a manuscript can \input{} it

Value

Invisibly returns the rendered LaTeX table as a string (or NULL when xtable is unavailable); the table is also printed.

Naming

report_art_con_table() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportArtConTable() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
  reportArtConTable(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportCLMM` *Report a cumulative link (mixed) model in LaTeX/APA style*

Description

Reporter for ordinal proportional-odds models fitted with **ordinal**: cumulative link mixed models (`ordinal::clmm`) and their fixed-effects counterpart (`ordinal::clm`). Each model term is first tested with a Type III Wald χ^2 test (`emmeans::joint_tests()`, independent of the contrast coding); each location coefficient is then reported – as an odds ratio for the logit link (the multiplicative change in the odds of being in a higher outcome category), as the raw coefficient with the link named otherwise – with its confidence interval, z statistic and p-value, labelled as the contrast it is (see `reportGLMM()`).

Usage

```
reportCLMM(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL,
  omnibus = TRUE
)

report_clmm(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL,
  omnibus = TRUE
)
```

Arguments

`model`	A fitted `ordinal::clmm` or `ordinal::clm` model.
`dv`	Name of the (ordinal) dependent variable, used in the sentence text.
`exponentiate`	`"auto"` (default; odds ratios for the logit link, raw coefficients for probit, cloglog and other links) or `TRUE`/`FALSE` to force it. `FALSE` reports raw log-odds.
`conf_level`	Confidence level for the intervals. Default 0.95.
`write_to_clipboard`	Whether to copy the sentences to the clipboard.
`sink_to`	Optional path of a `.tex` file to write the sentences to.
`omnibus`	Logical. Report the Type III omnibus test of every model term before the coefficients. Default `TRUE`; needs **emmeans**.

Details

Only the location coefficients are reported: the thresholds (cut-points, including the `threshold.1`/`spacing` parameters of equidistant thresholds), scale effects and nominal effects are not predictor effects on the location, so unlike `reportGLMM()` this reporter has no `include_intercept` argument.

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_clmm()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportCLMM()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ordinal", quietly = TRUE) &&
  requireNamespace("parameters", quietly = TRUE)) {
  m <- ordinal::clmm(rating ~ temp + contact + (1 | judge), data = ordinal::wine)
  reportCLMM(m, dv = "wine rating")
}
```

---

reportDunnTest *Report dunnTest as text*

Description

Reports the significant comparisons of an `FSA::dunnTest()` result as sentences such as "A Dunn post-hoc test (Holm-adjusted) found that ... for the Species virginica was significantly higher (Mdn=..., IQR=...) than for setosa (Mdn=..., IQR=...; p_adj<.001, r_rb=...)". The multiplicity correction is read from the test object; with `method = "none"` the p-values are uncorrected and are emitted as `\p{}`/`\pminor{}`, not as p_{adj}.

Usage

```
reportDunnTest(
  d,
  data,
  iv = "testiv",
  dv = "testdv",
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed")
)
```

```
report_dunn_test(
  d,
  data,
  iv = "testiv",
  dv = "testdv",
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed")
)
```

Arguments

d	the dunn test object
data	the data frame
iv	independent variable
dv	dependent variable
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them
descriptives	which descriptives to print beside each level: "auto" (default, median and IQR for this rank-based test), "mean", "median" or "trimmed" (20% trimmed mean and 20% winsorized SD).

Details

Which level is "higher" follows the sign of the test's Z (FSA labels a comparison "A - B"; $Z > 0$ means A has the higher mean rank). Until 0.3.0 it followed the raw means, which can point the other way: with a few large outliers a level can have the larger mean but the significantly lower mean rank. As a consistency check, the mean ranks are recomputed from data and a warning is given when they disagree with Z (i.e. when data is not the data the test was computed on).

By default (descriptives = "auto") each level is described by its median and IQR, the location measure that matches a rank-based test; "mean" restores *M*/*SD* and warns where the means order two levels against the test.

Required commands in LaTeX (all part of latex_preamble()): \padj, \padjminor, \p, \pminor, \mdn, \iqr, \m, \sd and \newcommand{\rankbiserial}[1]{r_{rb} = #1$}.

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

report_dunn_test() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportDunnTest() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)

  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTest(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

`reportDunnTestTable` *report Dunn test as a table. Customizable with sensible defaults.*

Description

The table lists the significant comparisons with their Z, p-value and rank-biserial correlation. FSA's Z for "A - B" is positive when A has the higher mean rank, which the caption states. The p-value column is headed "p-adjusted" and the caption names the correction read from the test object; a test run with `method = "none"` gets the header "p" and a caption saying the p-values are not adjusted. P-values are never rounded across .05 or .01.

Usage

```
reportDunnTestTable(
  d = NULL,
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)

report_dunn_test_table(
  d = NULL,
```

```
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)
```

Arguments

d	the dunn test object
data	the data frame
iv	independent variable
dv	dependent variable
orderByP	whether to order by the p value
numberDigitsForPValue	the number of digits to show
latexSize	which size for the text
orderText	whether to order the comparisons alphabetically; ignored when orderByP = TRUE
style	table rule style: "hline" (default, classic \hline rules) or "booktabs" (journal-standard \toprule/\midrule/\bottomrule; needs \usepackage{booktabs}).
sink_to	optional path of a .tex file to write the table to, so a manuscript can \input{} it

Value

Invisibly returns the rendered LaTeX table as a string (or NULL when xtable is unavailable); the table is also printed.

Naming

report_dunn_test_table() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportDunnTestTable() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)
```

```
  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTestTable(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

`reportggstatsplot` *Report statistical details for ggstatsplot.*

Description

Writes the omnibus test of a `ggbetweenstats()`/`ggwithinstats()` plot as a sentence, naming the test statsExpressions ran and the effect size it produced. Rank-based tests are reported with their χ^2, V or W statistic, t tests (including Yuen's trimmed-means test) as *t(df)*, and ANOVA-type tests as *F(df1, df2)*. A plot made with `type = "bayes"` carries a Bayes factor instead of a p-value; it is reported as BF_{10} together with the evidence category of Jeffreys (1961), as classified by `effectsize::interpret_bf()`.

Usage

```
reportggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_ggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

p	the object returned by ggwithinstats or ggbetweenstats
iv	the independent variable
dv	the dependent variable
write_to_clipboard	whether to write to the clipboard
sink_to	optional path of a .tex file to write the sentence to, so a manuscript can \input{} it

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

report_ggstatsplot() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportggstatsplot() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot
plt <- ggbetweenstats(mtcars, am, mpg)

reportggstatsplot(plt, iv = "am", dv = "mpg")
```

---

reportggstatsplotPostHoc

Report significant post-hoc pairwise comparisons

Description

This function extracts significant pairwise comparisons from a ggstatsplot object, describes the groups involved from the raw data, and prints LaTeX-formatted sentences reporting the results.

Usage

```
reportggstatsplotPostHoc(
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed"),
  subject = NULL
)

report_ggstatsplot_posthoc(
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL,
  descriptives = c("auto", "mean", "median", "trimmed"),
  subject = NULL
)
```

Arguments

data	A data frame containing the raw data used to generate the plot.
p	A ggstatsplot object (e.g., returned by ggbetweenstats) containing the pairwise comparison statistics.
iv	Character string. The column name of the independent variable (grouping variable).
dv	Character string. The column name of the dependent variable.
label_mappings	Optional named list or vector. Used to rename factor levels in the output text (e.g., list("old_name" = "New Label")).
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them. It is written in every case, including when nothing is significant, so a manuscript never keeps a stale result.
descriptives	which descriptives to print beside each level: "auto" (default; the ones matching the test, see Details), "mean" (*M*, *SD*), "median" (*Mdn*, *IQR*) or "trimmed" (20% trimmed mean, 20% winsorized SD).
subject	optional column name identifying participants in a within-subjects (ggwithinstats()) plot. When given, the descriptives and any direction statistic are computed only from participants observed in every level of iv, one row per participant and level.

Details

Each sentence names the post-hoc test that produced the comparison and the multiplicity correction applied to its p-value, both read from the pairwise table `ggstatsplot` attaches to the plot (e.g. "A Games-Howell post-hoc test (Holm-adjusted) found that ..."). Which test that is depends on the `type` of the plot and on whether it is between- or within-subjects – Games-Howell, Dunn, Durbin-Conover, Student's t or Yuen's trimmed means – so it is worth reporting rather than assuming. When the plot carries no such information (an older `ggstatsplot`, or a hand-built table) the sentence falls back to a plain "A post-hoc test ...".

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Direction and descriptives

Which level "was significantly higher" is decided by the quantity the test compares, computed from `data`: the means for Student's t and Games-Howell, the mean ranks of the pooled ranking for Dunn, the within-participant rank sums for Durbin-Conover, and the 20% trimmed means for Yuen's test (taken from the sign of the `estimate` column of the pairwise table when it is present). Until 0.3.0 the direction always came from the raw means, which contradicts a rank-based test whenever outliers or skew pull a mean – the situation in which such a test is chosen.

The descriptives printed beside each level match the test by default (`descriptives = "auto"`): `\m{}`/`\sd{}` for the mean-based tests, `\mdn{}`/`\iqr{}` for the rank-based ones, and the 20% trimmed mean M_t with the 20% winsorized standard deviation SD_w for Yuen's test. Any of these can be forced; a warning is given when the forced descriptives order two levels against the test's direction.

For a within-subjects plot pass `subject`: `ggwithinstats()` tests only the participants observed in every level, and with `subject` the descriptives (and the Durbin-Conover rank sums) are computed from exactly those participants. Without it, all rows are described, and a Durbin-Conover table pairs observations by row order within each level – what `ggwithinstats()` itself does without `subject.id` – with a warning.

LaTeX Requirements

To easily copy and paste the results to your manuscript, the commands of `latex_preamble()` (or the shipped `colleyRstats.sty`) must be defined: `\m`, `\sd`, `\mdn`, `\iqr`, `\padj`, `\padjminor`, `\p` and `\pminor`.

`\p`/`\pminor` are used only when the plot reports `p.adjust.method = "None"`: those p-values are uncorrected and must not be labelled p_{adj}.

Naming

`report_ggstatsplot_posthoc()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportggstatsplotPostHoc()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot (a factor with three levels, so there are pairwise tests)
plt <- ggbetweenstats(mtcars, cyl, mpg)

# Report stats
reportggstatsplotPostHoc(
  data = mtcars,
  p = plt,
  iv = "cyl",
  dv = "mpg",
  label_mappings = list("4" = "four cylinders")
)
```

---

reportGLMM *Report a (generalized) linear mixed model in LaTeX/APA style*

Description

Turns a fitted mixed model into ready-to-paste manuscript sentences: first a Type III omnibus test per model term (main effects and interactions), then one sentence per fixed-effect coefficient with its estimate (or odds / incidence-rate ratio), confidence interval, test statistic and p-value. Works with linear mixed models (`lme4::lmer`, `lmerTest::lmer`), generalized linear mixed models (`lme4::glmer`, `glmmTMB::glmmTMB`) and, for convenience, ordinary `lm`, `glm` and `MASS::glm.nb` fits.

Usage

```
reportGLMM(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  include_intercept = FALSE,
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL,
  omnibus = TRUE
)

report_glmm(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  include_intercept = FALSE,
  conf_level = 0.95,
```

```
  write_to_clipboard = FALSE,
  sink_to = NULL,
  omnibus = TRUE
)
```

Arguments

model	A fitted model (lmer, glmer, glmmTMB, lm, glm, or glm.nb).
dv	Name of the dependent variable, used in the sentence text.
exponentiate	"auto" (default; exponentiate for logit-link binomial and log-link count families), or TRUE/FALSE to force it.
include_intercept	Whether to also report the intercept. Default FALSE.
conf_level	Confidence level for the intervals. Default 0.95.
write_to_clipboard	Whether to copy the sentences to the clipboard.
sink_to	Optional path of a .tex file to write the sentences to, so a manuscript can \input{} them.
omnibus	Logical. Report the Type III omnibus test of every model term before the coefficients. Default TRUE; needs **lmerTest** (linear mixed models) or **emmeans** (all other models).

Details

Omnibus tests. With treatment (dummy) coding – R's default – a coefficient such as aa2 in y ~ a * b is the a2 - a1 difference *at the reference level of b*, not "the effect of a", and a factor with three or more levels has no single coefficient at all. The omnibus tests answer the questions a results section asks: for linear mixed models, **lmerTest**'s Type III F-tests with Satterthwaite's degrees of freedom; for all other models, emmeans::joint_tests() (Type III Wald F for lm, Wald χ^2 otherwise). Both are independent of the contrast coding. The coefficient sentences that follow are labelled as what they are, e.g. "the contrast a2 vs. a1 of a (at b = b1)", "the slope of age (at cond = A)".

Degrees of freedom. Coefficient t-tests of linear mixed models use Satterthwaite's degrees of freedom (**lmerTest**; the asymptotic normal approximation when it is not installed), and the sentence says so – never the residual degrees of freedom.

Effect labels. Coefficients are exponentiated automatically only where the ratio has a name: odds ratios (OR) for a logit link, risk ratios (RR) for a binomial log link, incidence-rate ratios (IRR) for a log-link count family (Poisson, negative binomial). Other links (probit, cloglog, identity) report the raw coefficient and name the link. Only the conditional-mean (location) coefficients are reported: zero-inflation, dispersion and scale parameters are not predictor effects on the mean.

The reported statistics rely on the **parameters** package. The LaTeX output uses the \p/\pminor and \F macros from latex_preamble().

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

report_glmm() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportGLMM() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("lme4", quietly = TRUE) &&
  requireNamespace("parameters", quietly = TRUE)) {
  m <- lme4::lmer(Reaction ~ Days + (1 | Subject), data = lme4::sleepstudy)
  reportGLMM(m, dv = "reaction time")
}
```

---

reportMeanAndSD — *Report the mean and standard deviation of a dependent variable for all levels of an independent variable rounded to the 2nd digit.*

Description

Each level yields one line such as A: \m{4.21}, \sd{1.03}, with the level name escaped for LaTeX. Written to sink_to, the lines are joined into one compilable sentence ("A: ...; B: ...."). Until 0.3.0 every line was a LaTeX comment (it began with a percent sign), so an \input{} of the file typeset nothing; as_comment = TRUE restores that form for notes kept next to a manuscript.

Usage

```
reportMeanAndSD(
  data,
  iv = "testiv",
  dv = "testdv",
  sink_to = NULL,
  as_comment = FALSE
)

report_mean_sd(
  data,
  iv = "testiv",
  dv = "testdv",
  sink_to = NULL,
  as_comment = FALSE
)
```

Arguments

data	the data frame
iv	the independent variable
dv	the dependent variable
sink_to	optional path of a .tex file to write the lines to, so a manuscript can \input{} them
as_comment	if TRUE, every line is prefixed with a percent sign (a LaTeX comment) and written one per line, as before 0.3.0. Defaults to FALSE: real text.

Details

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: \newcommand{\m}{\textit{M=}} \newcommand{\sd}{\textit{SD=}}

Value

Invisibly returns the formatted lines (one per level) as a character vector; the text is also emitted via message().

Naming

report_mean_sd() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportMeanAndSD() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
example_data <- data.frame(Condition = rep(c("A", "B", "C"),
each = 10), TLX1 = stats::rnorm(30))

reportMeanAndSD(example_data, iv = "Condition", dv = "TLX1")
```

---

reportNparLD *Report the model produced by nparLD. The model provided must be the model generated by the command 'nparLD' nparLD (see https://CRAN.R-project.org/package=nparLD).*

Description

Only significant main and interaction effects of the ANOVA-type statistic (ATS) are reported, as $F(df, \infty)$ with the estimated (fractional) numerator degrees of freedom. P-values are rounded for the third digit. No effect size is attached: the relative treatment effects 'nparLD' estimates describe factor-level combinations, not the terms of the ATS table. A term is reported as an interaction when its name contains `":"`. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).

Usage

```
reportNparLD(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_nparld(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	the model
`dv`	the dependent variable
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}[3]{$F({#1},{#2})={#3}$}` `\newcommand{\p}{\textit{p=}}` `\newcommand{\pminor}{\text`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_nparld()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportNparLD()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("nparLD", quietly = TRUE)) {
  # Small toy data set for nparLD
  set.seed(123)
  example_data <- data.frame(
    Subject = factor(rep(1:10, each = 3)),
    Time    = factor(rep(c("T1", "T2", "T3"), times = 10)),
    # a rising trend over time, so there is something to report
    TLX1    = rep(c(45, 52, 61), times = 10) + stats::rnorm(30, sd = 4)
  )

  # Fit nparLD model
  model <- nparLD::nparLD(TLX1 ~ Time, data = example_data, subject = "Subject")

  # Report the nparLD result
  reportNparLD(model, dv = "TLX1")
}
```

---

reportNPAV *Generate the Latex-text based on the NPAV by Lüpsen (see https://www.uni-koeln.de/~{}luepsen/R/). Only significant main and interaction effects are reported. P-values are rounded for the third digit and partial eta squared values, with a two-sided 95% confidence interval, are provided when possible. A term is reported as an interaction when its name contains ":". Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).*

Description

Deprecated: `reportNPAV()` will be removed in a future release. Use `reportART()` with ARTool instead.

Usage

```
reportNPAV(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

model	the model of the np.anova
dv	the name of the dependent variable that should be reported

write_to_clipboard
: whether to write to the clipboard

sink_to
: optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}[3]{$F({#1},{#2})={#3}$} \newcommand{\p}{\textit{p=}} \newcommand{\pminor}{\text`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
model <- data.frame(
  Df = c(1, 1, 10),
  `F value` = c(6.12, 5.01, NA),
  `Pr(>F)` = c(0.033, 0.045, NA),
  check.names = FALSE
)
rownames(model) <- c("Video", "gesture:eHMI", "Residuals")
reportNPAV(model, dv = "mental workload")
```

---

report_all *Analyze and report several dependent variables at once*

Description

Runs `analyze_and_report()` for each dependent variable (e.g., all questionnaire scales of a study) and additionally returns a summary table of the omnibus tests with Holm-adjusted p-values across the dependent variables, plus – when **patchwork** is installed – a combined figure.

Usage

```
report_all(
  data,
  dvs,
  iv,
  design = c("between", "within"),
  labels = NULL,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_dir = NULL,
  subject = NULL
)
```

Arguments

data	the data frame
dvs	character vector of dependent variable column names
iv	the independent variable (column name as string)
design	"between" (default) or "within"
labels	optional named character vector mapping a dv name to its axis label, e.g. c(tlx_mental = "Mental Demand")
xlabels	optional labels for the x-axis, passed to every plot
plotType	either "box", "violin", or "boxviolin" (default)
sink_dir	optional directory; each dv's sentences are written to <sink_dir>/<dv>.tex so a manuscript can \input{} them
subject	the participant-ID column (a string); required for design = "within". See analyze_and_report().

Details

Each dependent variable is analysed on its own complete cases (see analyze_and_report()), so in a within-subjects design a participant missing one rating is left out of that dependent variable only. The p.holm column corrects the omnibus p-values for the number of dependent variables; the per-dv sentences report the uncorrected ones.

Value

Invisibly returns a list with components results (named list of analyze_and_report() results), summary (data frame with one row per dv: method, statistic, p.value, and Holm-adjusted p.holm), and combined_plot (a patchwork figure, or NULL when patchwork is not installed).

Examples

```
out <- report_all(mtcars, dvs = c("mpg", "disp"), iv = "cyl")
out$summary
```

---

reshape_data *Reshape Excel Data Based on Custom Markers and Include Custom ID Column*

Description

This function takes an Excel file with data in a wide format and transforms it to a long format. It includes a customizable "ID" column in the first position and repeats it for each slice. The function identifies sections of columns between markers that start with a user-defined string (default is "videoinfo") and appends those sections under the first section, aligning by column index. Columns before the first marker (e.g. a survey export's metadata or demographics) are not a section: like the ID column, they are repeated for every slice. The marker columns themselves are dropped.

Usage

```
reshape_data(
  input_filepath,
  sheetName = "Results",
  marker = "videoinfo",
  id_col = "ID",
  output_filepath
)
```

Arguments

input_filepath	String, the file path of the input Excel file.
sheetName	String, the name of the sheet to read from the Excel file. Default is "Results".
marker	String, the string that identifies the start of a new section of columns. Default is "videoinfo".
id_col	String, the name of the column to use as the ID column. Default is "ID".
output_filepath	String, the file path for the output Excel file.

Details

Relevant if you receive data in wide-format but cannot use built-in functionality due to naming (e.g., in LimeSurvey)

Value

None, writes the reshaped data to an Excel file specified by output_filepath.

Examples

```
if (requireNamespace("writexl", quietly = TRUE) &&
  requireNamespace("readxl", quietly = TRUE)) {
  tmp_in  <- tempfile(fileext = ".xlsx")
  tmp_out <- tempfile(fileext = ".xlsx")

  # Two marker-delimited sections of equal width; each section is stacked
  # under the first one, keyed by the ID column.
  toy <- data.frame(
    ID = c(1, 2),
    videoinfo1 = c("marker", "marker"),
    rating = c(10, 11),
    videoinfo2 = c("marker", "marker"),
    rating2 = c(20, 21),
    stringsAsFactors = FALSE
  )

  writexl::write_xlsx(toy, tmp_in)

  reshape_data(
```

```
    input_filepath = tmp_in,
    marker = "videoinfo",
    id_col = "ID",
    output_filepath = tmp_out
  )

  out <- readxl::read_excel(tmp_out)
  print(out)
}
```

`reverse_code` *Reverse-code responses*

Description

Flips a response scale so that a negatively worded item points the same way as the rest of its subscale: `min + max - x`.

Usage

```
reverse_code(x, min, max)
```

Arguments

`x`	Numeric responses.
`min, max`	The endpoints of the response scale the item was answered on – the *possible* range, not the observed one. Taking them from the data is the classic reverse-coding bug: if nobody picked 1, the flip is off by a point for every respondent.

Value

`x`, reverse-coded, with `NA` preserved.

Examples

```
reverse_code(c(1, 3, 5, NA), min = 1, max = 5)
```

`rFromNPAV` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements.

Description

Computes $r = |z|/\sqrt{N}$ with $z = \Phi^{-1}(p/2)$ for a two-sided p-value, or $z = \Phi^{-1}(p)$ for a one-sided one (`alternative = "less"` or `"greater"`).

Usage

```
rFromNPAV(pvalue, N, alternative = "two.sided", df1 = NULL)
```

Arguments

`pvalue`	p value
`N`	number of measurements in the experiment
`alternative`	`"two.sided"` (default), `"less"` or `"greater"`: the alternative of the test that produced `pvalue`.
`df1`	optional numerator degrees of freedom of the F test that produced `pvalue`; a value above 1 triggers a warning (see Details).

Details

The conversion is defined for *focused* tests with a single degree of freedom (Rosenthal, 1994). The p-value of an F test with one numerator degree of freedom equals the two-sided p-value of the matching t test, so the default is right for such effects. An omnibus F test with more than one numerator degree of freedom – a main effect of a three-level factor, say – has no single direction and no z equivalent: its p-value converts to a number, but not to the effect size r of anything. Pass `df1` to be warned in that case, and report partial eta squared for such effects instead (as `reportNPAV()` and `reportART()` do).

Necessary LaTeX command: `\newcommand{\effectsize}{\textit{r=}}`

Value

Invisibly returns a list with components:

- `r`: effect size as a numeric scalar.
- `z`: corresponding z-statistic.
- `text`: LaTeX-formatted character string that is also sent to the console.

References

Rosenthal, R. (1994). Parametric measures of effect size. In H. Cooper & L. V. Hedges (Eds.), *The handbook of research synthesis* (pp. 231–244). Russell Sage Foundation.

Examples

```
rFromNPAV(0.02, N = 180)
```

`rFromWilcox` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements.

Description

Computes $r = |z|/\sqrt{N}$, recovering z from the test's p-value. A two-sided p-value splits its probability over both tails, so $z = \Phi^{-1}(p/2)$; a one-sided test (`alternative = "less"` or `"greater"`, read from the test object) has it all in one tail, so $z = \Phi^{-1}(p)$. Halving a one-sided p-value, as this function did before 0.3.0, overstates $|z|$ and therefore r.

Usage

```
rFromWilcox(wilcoxModel, N)
```

Arguments

`wilcoxModel`	the Wilcox model (an `htest` object from `stats::wilcox.test()`); its `alternative` is taken into account.
`N`	number of measurements in the experiment

Details

r is returned as a magnitude (non-negative); read the direction of the effect from the data. With an exact p-value (small samples without ties) the recovered z is the normal deviate matching that p-value rather than the test's normal-approximation statistic.

Value

Invisibly returns a list with components:

- `r`: effect size as a numeric scalar.
- `z`: corresponding z-statistic.
- `text`: character string that is also sent to the console.

References

Rosenthal, R. (1994). Parametric measures of effect size. In H. Cooper & L. V. Hedges (Eds.), *The handbook of research synthesis* (pp. 231–244). Russell Sage Foundation.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
rFromWilcox(w, N = nrow(d))
```

`rFromWilcoxAdjusted` *Effect size r from a multiplicity-inflated Wilcoxon p-value (deprecated)*

Description

[Deprecated]

This function is deprecated because the quantity it returns is not an effect size. It multiplies the p-value by `adjustFactor` before converting it to r, which shrinks r towards zero as the number of comparisons grows ($r = 0.34$ becomes 0.21 with six comparisons and 0 with forty) although the effect itself is unchanged. Multiplicity corrections belong to the p-values, which decide significance; an effect size describes the magnitude of one comparison and is reported unadjusted. Adjust the p-values with `stats::p.adjust()` and compute r with `rFromWilcox()`. The old value is still returned for compatibility.

Usage

```
rFromWilcoxAdjusted(wilcoxModel, N, adjustFactor)
```

Arguments

`wilcoxModel`	the Wilcox model; its `alternative` is taken into account as in `rFromWilcox()`.
`N`	number of measurements in the experiment
`adjustFactor`	the factor the p-value is multiplied by (the number of comparisons).

Value

Invisibly returns a list with components:

- `r`: the shrunken "effect size" as a numeric scalar.
- `z`: the z-statistic of the inflated p-value.
- `text`: character string that is also sent to the console.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
# Instead of rFromWilcoxAdjusted(w, N = nrow(d), adjustFactor = 2):
stats::p.adjust(w$p.value, method = "bonferroni", n = 2)
rFromWilcox(w, N = nrow(d))
```

`save_paper_figure` *Save a plot with publication-ready defaults*

Description

Saves a ggplot with sizes matching common two-column conference/journal layouts (e.g., ACM): a single-column figure is 3.33 in wide, a full-width figure 7 in. On Windows and Linux, PDFs are rendered with `grDevices::cairo_pdf` so that fonts are embedded and unicode glyphs survive; on macOS the default pdf device is used instead, because R's cairo on macOS is known to crash some setups (e.g., GitHub Actions runners) and the macOS device handles fonts well on its own.

Usage

```
save_paper_figure(
  plot = ggplot2::last_plot(),
  filename,
  columns = 1,
  width = NULL,
  height = NULL,
  base_size = NULL,
  dpi = 300,
  device = NULL
)
```

Arguments

`plot`	The plot to save (defaults to the last plot displayed).
`filename`	Output path; the extension selects the device (`.pdf` is recommended for LaTeX).
`columns`	1 for a single-column figure, 2 for a full-width figure. Ignored when `width` is given.
`width`	Figure width in inches; overrides `columns`.
`height`	Figure height in inches. Defaults to 2/3 of the width.

base_size	Base font size in points for the saved figure, applied via colley_theme() sizing on top of whatever theme the plot carries. The default NULL derives it from width, so a 3.33 in figure gets about 7 pt and a 7 in figure about 9 pt – close to the body text of a typical two-column paper, which is what makes a figure legible at 100\ rather than only when zoomed. Pass a number to choose it yourself, or NA to leave the plot's own text sizes untouched.
dpi	Resolution for raster output. Default 300.
device	Graphics device passed to ggplot2::ggsave(). The default NULL selects it automatically as described above; pass e.g. grDevices::cairo_pdf explicitly to override.

Value

Invisibly returns filename.

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl), mpg)) +
  ggplot2::geom_boxplot()
save_paper_figure(p, file.path(tempdir(), "cyl-mpg.pdf"), columns = 1)
```

---

score_questionnaire *Score a standardised questionnaire*

Description

Turns the raw item columns of a published questionnaire into its subscale scores, applying that instrument's own scoring key: reverse-coding, any recoding it prescribes (centring a semantic differential to $-3..+3$, zero-basing the SUS), the subscale structure, and the published weights or multipliers. Supported instruments are listed by list_questionnaires(); register your own with define_questionnaire().

Usage

```
score_questionnaire(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
  reverse_items = NULL,
  min_valid = 1,
  append = FALSE,
  prefix_out = NULL,
  verbose = TRUE,
  unreverse_items = NULL,
  impute = c("default", "prorate", "midpoint")
)
```

Arguments

data	A data frame with one row per respondent (or per respondent and condition) and one column per item.
instrument	Instrument key, e.g. "sus", "nasa_tlx", "ueq_s", "tia", "ssq". See list_questionnaires().
items	Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (c(mental = "tlx_md", effort = "tlx_ef", ...)), which is order-proof and the safer choice for a survey export you did not lay out yourself.
prefix	Optional column-name prefix selecting the item columns, e.g. "sus_". Must select exactly as many columns as the instrument has items. The columns are then matched to items *by their names*, never by sort order: either what follows the prefix is the item's code, label or (for a single-item subscale) subscale name, compared without case or punctuation (tlx_mental, tlx_Mental_Demand, tia_rc1, ipq_SP2); or it carries one item number per column, and those numbers are exactly 1..n (sus_2, SUS[2], Q5_2; in SUS_2_1 the number that varies across the columns is the item). Anything else – numbering from 0, numbers that do not run 1..n, names that only partly match – is an error asking for a named items mapping, rather than a guess.
scale	Optional two-element vector giving the response range your survey used, e.g. c(1, 21) for the 21-point NASA-TLX sheet or c(0, 4) for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale. Without scale, a coding that fits the range but looks shifted against it draws a warning (see Details); passing scale explicitly confirms the coding and silences it.
reverse_items	Optional items to reverse-code in addition to the scoring key's own, given as item numbers or item codes – for a survey that printed a pair the other way round. For backward compatibility it *toggles*: naming an item the key already reverses un-reverses it, and because passing an instrument's own reverse set (e.g. c(2, 4, 6, 8, 10) for the SUS) is a common mistake that would do exactly that, it raises a warning. Use unreverse_items to undo a key reversal on purpose.
min_valid	Minimum proportion of a subscale's items that must be answered for a score to be produced. The default 1 scores only complete subscales and returns NA otherwise – no silent imputation. Relax it (e.g. 0.8) to score partially complete responses; how the missing items are then completed is set by impute.
append	Logical. If TRUE, return data with the score columns added; if FALSE (default), return only the scores.
prefix_out	Optional string prefixed to every score column, useful when the same instrument is scored more than once per row (pre/post, or one block per condition).
verbose	Logical. If TRUE (default), emit the one-time mapping message described above.
unreverse_items	Optional items the scoring key reverses that should *not* be reversed, because the export already stored them reverse-coded. Naming an item the key does not reverse is an error.

`impute` How the missing items of a row that clears `min_valid` are completed. `"default"` applies the instrument's published rule: for the SUS, each missing item counts as the scale's centre point, as Brooke (1996) instructs respondents who cannot answer an item; for every other instrument, a subscale is the mean of the items present (a summed score is scaled up proportionally, so it stays on its published range). `"prorate"` uses the mean of the items present everywhere, and `"midpoint"` the scale centre everywhere. Irrelevant with the default `min_valid = 1`.

Details

Item columns must hold numbers. Factors are read by their labels when the labels are numbers, and by level order only when the factor is ordered. Text that is not a number (`"Strongly agree"`, `"5 - Strongly agree"`, a decimal comma) is an error naming the values, because converting it would silently turn responses into missing values; blank cells are missing. A fractional response on an instrument answered in whole scale points draws a warning.

The scoring message reports the observed response range next to the assumed one. Without an explicit `scale`, three codings that fit the assumed range but look shifted draw a warning: NASA-TLX responses that never exceed 21 on the 0–100 scale (the 21-point sheet); and, for multi-item whole-point instruments with at least five respondents, no response at 0 on a zero-based scale (a 1-based export, e.g. IPQ 1–7) or no response at the top of a one-based scale (a 0-based export, e.g. SUS 0–4).

Value

A data frame with one row per row of `data` and one column per subscale, plus the instrument's overall score where it defines one. The instrument definition and the resolved column mapping are attached as the `"instrument"` and `"mapping"` attributes.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself. A `prefix` maps columns by what their names say – an item code or label, or one item number per column running 1 to the number of items – never by sort order, and stops when the names do not identify the items. Note that the `attrakdiff` key is blocked by dimension with the negative pole first; data stored as answered on the official AttrakDiff sheet belong to `attrakdiff_official`.

See Also

check_questionnaire() to verify the mapping, score_reliability() for internal consistency, questionnaire_items() for the item list, define_questionnaire() to add an instrument.

Examples

```
set.seed(42)
sus_raw <- as.data.frame(matrix(sample(1:5, 10 * 6, TRUE), nrow = 6))
names(sus_raw) <- paste0("sus_", 1:10)

score_questionnaire(sus_raw, "sus", prefix = "sus_")

# A 21-point NASA-TLX sheet, scored onto the conventional 0-100
tlx <- data.frame(
  mental = c(14, 3), physical = c(2, 1), temporal = c(11, 4),
  performance = c(6, 2), effort = c(13, 5), frustration = c(9, 1)
)
score_questionnaire(tlx, "nasa_tlx", scale = c(1, 21))
```

---

score_reliability *Internal consistency of a questionnaire's subscales*

Description

Cronbach's alpha per subscale – and for the whole scale, where the instrument defines an overall score (the SUS, the SSQ Total, RTLX, the UEQ-S Overall) – computed on the same recoded item matrix that score_questionnaire() aggregates. Reverse-coded items are therefore already flipped, and a negative alpha means a genuine problem rather than a forgotten reversal.

Usage

```
score_reliability(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
  reverse_items = NULL,
  verbose = TRUE,
  unreverse_items = NULL
)
```

Arguments

data	A data frame with one row per respondent (or per respondent and condition) and one column per item.
instrument	Instrument key, e.g. "sus", "nasa_tlx", "ueq_s", "tia", "ssq". See list_questionnaires().

`items` Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (`c(mental = "tlx_md", effort = "tlx_ef", ...)`), which is order-proof and the safer choice for a survey export you did not lay out yourself.

`prefix` Optional column-name prefix selecting the item columns, e.g. `"sus_"`. Must select exactly as many columns as the instrument has items. The columns are then matched to items *by their names*, never by sort order: either what follows the prefix is the item's code, label or (for a single-item subscale) subscale name, compared without case or punctuation (`tlx_mental`, `tlx_Mental_Demand`, `tia_rc1`, `ipq_SP2`); or it carries one item number per column, and those numbers are exactly `1..n` (`sus_2`, `SUS[2]`, `Q5_2`; in `SUS_2_1` the number that varies across the columns is the item). Anything else – numbering from 0, numbers that do not run `1..n`, names that only partly match – is an error asking for a named `items` mapping, rather than a guess.

`scale` Optional two-element vector giving the response range your survey used, e.g. `c(1, 21)` for the 21-point NASA-TLX sheet or `c(0, 4)` for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale. Without `scale`, a coding that fits the range but looks shifted against it draws a warning (see Details); passing `scale` explicitly confirms the coding and silences it.

`reverse_items` Optional items to reverse-code in addition to the scoring key's own, given as item numbers or item codes – for a survey that printed a pair the other way round. For backward compatibility it *toggles*: naming an item the key already reverses un-reverses it, and because passing an instrument's own reverse set (e.g. `c(2, 4, 6, 8, 10)` for the SUS) is a common mistake that would do exactly that, it raises a warning. Use `unreverse_items` to undo a key reversal on purpose.

`verbose` Logical. If `TRUE` (default), emit the one-time mapping message described above.

`unreverse_items`
Optional items the scoring key reverses that should *not* be reversed, because the export already stored them reverse-coded. Naming an item the key does not reverse is an error.

Details

McDonald's omega (total) is added when **psych** is installed. It is computed from an unrotated one-factor solution (`psych::fa(nfactors = 1, rotate = "none")`) as $1 - \sum u^2 / \sum R$, which is the `omega.tot` that `psych::omega(nfactors = 1)` reports, without that function's dependence on **GPArotation** and without its automatic flipping of negatively loading items – which would hide exactly the reverse-coding error this function is meant to expose. Omega needs at least three items; for two-item scales it is not identified.

For two-item scales (the SUS Learnability scale, three TiA subscales) the Spearman-Brown coefficient $2r/(1 + r)$ is reported as well, which Eisinga, te Grotenhuis & Pelzer (2013) recommend over alpha: alpha underestimates the reliability of a two-item scale whenever the two item variances differ.

Value

A data frame with one row per subscale, plus a final row named after the overall score where the instrument defines one: the number of items, the number of complete cases it was computed on, `alpha`, `omega` (omega total), `spearman_brown` (two-item scales only), the mean inter-item correlation, and a `note` saying why a coefficient is missing (a single item, two items, too few complete responses, **psych** not installed, or the reason the factor solution failed). Subscales of a single item yield `NA` – internal consistency is not defined for them.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself. A `prefix` maps columns by what their names say – an item code or label, or one item number per column running 1 to the number of items – never by sort order, and stops when the names do not identify the items. Note that the `attrakdiff` key is blocked by dimension with the negative pole first; data stored as answered on the official AttrakDiff sheet belong to `attrakdiff_official`.

References

Eisinga, R., te Grotenhuis, M., & Pelzer, B. (2013). The reliability of a two-item scale: Pearson, Cronbach, or Spearman-Brown? *International Journal of Public Health, 58*(4), 637–642. doi:10.1007/s0003801204163

McDonald, R. P. (1999). *Test theory: A unified treatment*. Erlbaum.

See Also

`score_questionnaire()`

Examples

```
set.seed(3)
trait <- rnorm(60)
d <- as.data.frame(lapply(1:10, function(i) {
  round(pmin(pmax(3 + trait + rnorm(60, sd = 0.6), 1), 5))
}))
names(d) <- paste0("sus_", 1:10)
# Items 2, 4, 6, 8, 10 are negatively worded on the real SUS, so flip them
```

```
d[paste0("sus_", c(2, 4, 6, 8, 10))] <- 6 - d[paste0("sus_", c(2, 4, 6, 8, 10))]
score_reliability(d, "sus", prefix = "sus_")
```

`stat_sum_df` *Generating the sum and adding a crossbar.*

Description

Generating the sum and adding a crossbar.

Usage

```
stat_sum_df(fun, geom = "crossbar", ...)
```

Arguments

`fun`	function
`geom`	geom to be shown
`...`	Additional arguments passed to stat_summary

Value

A `ggplot2` layer that can be added to a ggplot object.

Examples

```
# Simple summary function: use the mean as y, ymin, and ymax
mean_fun <- function(x) {
  m <- mean(x, na.rm = TRUE)
  data.frame(y = m, ymin = m, ymax = m)
}

ggplot2::ggplot(mtcars, ggplot2::aes(x = factor(cyl), y = mpg)) +
  stat_sum_df(mean_fun)
```

summarize_sickness *Summarise a motion-sickness time course*

Description

Repeated single-item sickness ratings (FMS, MISC) are collected once a minute and then, almost always, reduced to a handful of per-participant numbers before they are analysed. This produces those numbers: peak, mean, final rating, the area under the rating curve, and when a threshold was first crossed.

Usage

```
summarize_sickness(data, value, id, time = NULL, by = NULL, threshold = NULL)
```

Arguments

`data`	A data frame in long format: one row per rating.
`value`	Column holding the rating.
`id`	Column identifying the participant (and, with `by`, the condition).
`time`	Optional column holding the time of the rating, in whatever unit the study used. Needed for a meaningful area under the curve and for time-to-threshold; without it, the rating index is used and `auc` is reported in rating-steps.
`by`	Optional further grouping columns, e.g. the experimental condition.
`threshold`	Optional rating at or above which a participant counts as affected, e.g. `6` for MISC (nausea) or `10` for FMS.

Value

A data frame with one row per participant (and `by` group): `n` ratings, `peak`, `mean`, `final`, `auc` (trapezoidal), `auc_rate` (the AUC divided by the observed duration, i.e. the time-weighted mean rating), and – with `threshold` – `reached` and `time_to_threshold`.

See Also

`score_questionnaire()` for the single-measurement instruments

Examples

```
d <- data.frame(
  pid = rep(c("p1", "p2"), each = 5),
  minute = rep(0:4, 2),
  fms = c(0, 1, 3, 6, 8, 0, 0, 1, 1, 2)
)
summarize_sickness(d, value = "fms", id = "pid", time = "minute", threshold = 5)
```

use_colleyrstats_sty *Write colleyRstats.sty into a project (for Overleaf)*

Description

Copies the **colleyRstats** LaTeX macro package next to your manuscript so the report output compiles with a single `\usepackage{colleyRstats}` – no need to paste `latex_preamble()` into the preamble. Upload the written `colleyRstats.sty` to your Overleaf project (or keep it in the same folder as `main.tex`).

Usage

```
use_colleyrstats_sty(dir = ".", overwrite = FALSE)
```

Arguments

`dir`	Directory to write `colleyRstats.sty` into. Default the current working directory.
`overwrite`	Overwrite an existing file? Default `FALSE`.

Value

Invisibly, the path to the written `.sty` file.

Examples

```
use_colleyrstats_sty(tempdir(), overwrite = TRUE)
```

use_study_project *Scaffold a reproducible study analysis*

Description

Writes a complete, runnable analysis project: a **targets** pipeline that recomputes only what changed, R scripts split along the stages every user study goes through (read, clean, score, model, plot), a Quarto report, and a directory the generated LaTeX lands in so a manuscript can `\input{}` the numbers instead of having them re-typed.

Usage

```
use_study_project(
  path,
  name = NULL,
  questionnaires = c("nasa_tlx", "sus"),
  renv = FALSE,
  git = TRUE,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

path	Directory to create the project in. Created if it does not exist.
name	Project name, used in the README and the report title. Defaults to the directory name.
questionnaires	Character vector of instrument keys the study uses, e.g. `c("nasa_tlx", "sus")`. The scoring script is generated with one call per instrument. See `list_questionnaires()`.
renv	Logical. Initialise **renv** in the project, pinning the package versions this analysis was run with – which is what makes the project still run in three years, and what makes it a usable open-science artifact. Default `FALSE`: initialising discovers every package the scripts use and installs it into a project library, which can mean network downloads, so it is opt-in. With `TRUE`, `renv::init()` runs in a separate R process (via **callr** when installed, otherwise `Rscript`), because activating a project rewrites the calling session's library paths, environment variables (`R_LIBS_USER`, `PATH`, `RENV_PATHS_*`), repository options and sandbox; your current session is left exactly as it was. Run `renv::init()` yourself later to opt in after the fact.
git	Logical. Write a `.gitignore` suited to an R analysis project. Default `TRUE`.
overwrite	Logical. Replace files that already exist. Default `FALSE`, so running this on a live project adds missing pieces without touching your work.
quiet	Logical. Suppress the per-file messages.

Details

The generated pipeline is wired to this package: scoring goes through `score_questionnaire()`, model choice and fitting through `fit_recommended()`, sentences and tables through the `report_*` functions, and figures through `save_paper_figure()`. It runs as written against the example data it ships with, so the first thing you do in a new project is see a green pipeline, then replace the example data with yours.

Value

Invisibly, the normalised project path.

Why a pipeline rather than a script

A study analysis is re-run many times – after a data fix, after a reviewer asks for one more contrast, after a co-author changes a factor label. With a script, that means re-running everything and hoping nothing stale is left in the workspace; **targets** tracks which step depends on what and recomputes only the affected ones, which also means the pipeline is a machine-checkable record of how each number was produced.

See Also

`score_questionnaire()`, `fit_recommended()`, `save_paper_figure()`, `emit_overleaf()`

Examples

```
project <- file.path(tempdir(), "driving-study")
use_study_project(project, questionnaires = c("nasa_tlx", "sus"), renv = FALSE)
list.files(project, recursive = TRUE)
```

Index

%!in% *(not_in)*, 52

add_pareto_emoa_column, 3
add_pareto_emoa_column(), *5*
add_pareto_moocore_column, 5
add_pareto_moocore_column(), *4*
analyze_and_report, 6
analyze_and_report(), *27*, *32*, *82, 83*
animate_mobo2, 8
ARTool::art.con(), *62, 63*, *65*
assumption_methods_text, 9
assumption_methods_text(), *6*, *13*, *15*

check_assumptions_anova *(checkAssumptionsForAnova)*, 11
check_homogeneity_by_group, 13
check_homogeneity_by_group(), *10*
check_normality_by_group, 14
check_normality_by_group(), *9–12*, *40*, *43*
check_questionnaire, 15
check_questionnaire(), *17*, *23*, *25*, *55*, *92, 93*, *95*
checkAssumptionsForAnova, 11
cite_methods, 17
cite_methods(), *28*
classify_outcome, 18
classify_outcome(), *31*, *56, 57*
colley_theme, 21
colley_theme(), *20*, *90*
colleyRstats_setup, 19

debug_contr_error, 22
define_questionnaire, 23
define_questionnaire(), *50*, *90*, *93*
define_result_macro, 25

effectsize::F_to_eta2(), *61*
effectsize::interpret_bf(), *72*
emit_name_macros, 26
emit_overleaf, 27
emit_overleaf(), *100*
expand_latex_macros, 28

figure_base_size, 29
fit_recommended, 30
fit_recommended(), *19*, *54*, *56, 57*, *99, 100*
FSA::dunnTest(), *68*

generateEffectPlot, 33
generateMoboPlot, 35
generateMoboPlot2, 38
generateMoboPlot2(), *8, 9*
ggbetweenstatsWithPriorNormalityCheck, 40
ggbetweenstatsWithPriorNormalityCheck(), *42*
ggbetweenstatsWithPriorNormalityCheckAsterisk, 42
ggplot2::ggsave(), *90*
ggplot2::mean_cl_boot(), *34*
ggwithinstatsWithPriorNormalityCheck, 43
ggwithinstatsWithPriorNormalityCheck(), *46*
ggwithinstatsWithPriorNormalityCheckAsterisk, 45

latex_escape, 49
latex_escape(), *28*
latex_preamble, 49
latex_preamble(), *29*, *64*, *69*, *75*, *77*, *98*
latexify_report, 47
list_questionnaires, 50
list_questionnaires(), *16*, *25*, *55*, *90, 91*, *93*, *99*

n_fun, 53
na.zero, 50
normalize, 51
not_empty, 52

not_in, 52
nparLD, *79*

pathPrep, 53
plot_between_stats *(*ggbetweenstatsWithPriorNormalityCheck*)*, 40
plot_between_stats_asterisk *(*ggbetweenstatsWithPriorNormalityCheckAsterisk*)*, 42
plot_effect *(*generateEffectPlot*)*, 33
plot_fit, 54
plot_fit(), *32*
plot_mobo *(*generateMoboPlot*)*, 35
plot_mobo2 *(*generateMoboPlot2*)*, 38
plot_within_stats *(*ggwithinstatsWithPriorNormalityCheck*)*, 43
plot_within_stats_asterisk *(*ggwithinstatsWithPriorNormalityCheckAsterisk*)*, 45

questionnaire_items, 55
questionnaire_items(), *17*, *50*, *93*

recommend_analysis *(*recommend_test*)*, 56
recommend_test, 56
recommend_test(), *19*, *30–32*
remove_outliers_REI, 58
replace_values, 60
report_all, 82
report_all(), *27*
report_art *(*reportART*)*, 61
report_art_con *(*reportArtCon*)*, 62
report_art_con_table *(*reportArtConTable*)*, 64
report_clmm *(*reportCLMM*)*, 67
report_dunn_test *(*reportDunnTest*)*, 68
report_dunn_test_table *(*reportDunnTestTable*)*, 70
report_ggstatsplot *(*reportggstatsplot*)*, 72
report_ggstatsplot_posthoc *(*reportggstatsplotPostHoc*)*, 73
report_glmm *(*reportGLMM*)*, 76
report_mean_sd *(*reportMeanAndSD*)*, 78
report_nparld *(*reportNparLD*)*, 79
reportART, 61
reportART(), *86*
reportArtCon, 62
reportArtConTable, 64
reportCLMM, 67
reportDunnTest, 68
reportDunnTest(), *62*
reportDunnTestTable, 70
reportDunnTestTable(), *64*
reportggstatsplot, 72
reportggstatsplot(), *6*
reportggstatsplotPostHoc, 73
reportggstatsplotPostHoc(), *6*, *41*, *44*
reportGLMM, 76
reportGLMM(), *67*, *68*
reportMeanAndSD, 78
reportNparLD, 79
reportNPAV, 81
reportNPAV(), *86*
reshape_data, 83
reverse_code, 85
rFromNPAV, 86
rFromWilcox, 87
rFromWilcox(), *88*
rFromWilcoxAdjusted, 88
rstatix::anova_test(), *12*
rstatix::levene_test(), *13*

save_paper_figure, 89
save_paper_figure(), *20*, *22*, *28*, *29*, *54*, *99*, *100*
score_questionnaire, 90
score_questionnaire(), *15–17*, *23*, *25*, *50*, *55*, *93*, *95*, *97*, *99*, *100*
score_reliability, 93
score_reliability(), *23*, *93*
see::scale_colour_see(), *34*
stat_sum_df, 96
stats::lm(), *11*
stats::p.adjust(), *14*, *88*
stats::wilcox.test(), *87*
summarize_sickness, 97

use_colleyrstats_sty, 98
use_colleyrstats_sty(), *49*
use_study_project, 98