Package: rwf (via r-universe)

November 21, 2026

Title Statistical reporting and visualization for common methods

Version 0.1.0

Author Dimitrios Zacharatos [aut, cre]

Maintainer Dimitrios Zacharatos <dzacharatos@yahoo.com>

Description Utilities for psychological data analysis, including automated statistical reporting and visualization for common inferential and modeling workflows. Helps users run analyses faster with less repetitive code and more standardized output.

Depends R (>= 3.5), ggplot2

Imports MASS, future, future.apply, gtools, irr, openxlsx, pROC, plyr, psych, reshape2, xgboost, nortest, Rmisc, ggfortify, ggrepel, car, sjstats, ggpubr, gridExtra, scales, pwr, mirt, nlme, NLP, openNLP, tm, spelling, lavaan, ggExtra, purrr, emmeans, ez, htmlwidgets, mixlm, rstudioapi, igraph, stopwords, gtable, stringr, rlang

Suggests caret, coin, DescTools, DiagrammeR, e1071, effectsize, forecast, GPArotation, lme4, lsr, Matrix, openNLPdata, rcompanion, Rcsdp, rJava, rmarkdown, rstatix, testthat (>= 3.0.0), thurstonianIRT, withr

URL https://sedzinfo.github.io/rwf/

BugReports https://github.com/sedzinfo/rwf/issues

Roxygen list(markdown = FALSE)

License GPL (>= 3)

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Config/pak/sysreqs
cmake libglpk-dev make default-jdk libicu-dev libuv1-dev libxml2-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-11-21 16:40:10 UTC

RemoteUrl https://github.com/sedzinfo/rwf

RemoteRef HEAD

RemoteSha 01076616bfc50a0a1c533c45c3c7cc9378b1132f

RemoteSubdir rwf

Contents

c_bind . . . 6
call_to_string . . . 7
cdf . . . 8
cdff . . . 10
cfa_icc_index . . . 11
change_data_type . . . 12
check_heywood . . . 13
clear_stopwords . . . 14
clear_text . . . 15
combine_files . . . 15
comparison_combinations . . . 16
compute_ability . . . 17
compute_adjustment . . . 18
compute_aggregate . . . 19
compute_alpha_diagnostics . . . 20
compute_aov_es . . . 21
compute_cohens_d . . . 23
compute_confidence_inteval . . . 25
compute_crosstable . . . 25
compute_descriptives . . . 26
compute_dissatenuation . . . 27
compute_dummy_comparisons . . . 28
compute_efa_cfa . . . 29
compute_frequencies . . . 31
compute_friedman_test . . . 32
compute_icc_thurstonian . . . 34
compute_info_1pl . . . 35
compute_info_2pl . . . 36
compute_info_3pl . . . 36
compute_kruskal_wallis_test . . . 37
compute_kurtosis . . . 39
compute_map . . . 40
compute_mean_sd_alpha . . . 41
compute_moving_average . . . 41
compute_one_way_test . . . 42
compute_posthoc . . . 45
compute_power_r . . . 46
compute_power_r_matrix . . . 47

compute_raw_alpha . . . 48
compute_residual_stats . . . 48
compute_scores . . . 49
compute_se_theta . . . 50
compute_shrout . . . 50
compute_skewness . . . 51
compute_solve . . . 52
compute_standard . . . 53
compute_standard_error . . . 55
compute_text_similarity . . . 55
compute_tversky_index . . . 56
compute_unidimensional_ability . . . 57
compute_unidimensional_theta . . . 58
compute_wilcoxon_effect_size . . . 59
compute_y_logistic . . . 60
confusion . . . 61
confusion_matrix_percent . . . 62
convert_excel_unix_timestamp . . . 63
data_frame_index . . . 63
decompose_datetime . . . 64
deg2rad . . . 65
detach_package . . . 66
df_admission . . . 66
df_automotive_data . . . 67
df_blood_pressure . . . 68
df_co2 . . . 69
df_crop_yield . . . 70
df_difficile . . . 70
df_insurance . . . 71
df_ocean . . . 72
df_personality . . . 75
df_responses . . . 77
df_responses_state . . . 82
df_sexual_comp . . . 82
df_titanic . . . 83
display_upper_lower_triangle . . . 84
dotnames . . . 85
drop_levels . . . 86
dummy_arrange . . . 86
environment_options . . . 87
excel_confusion_matrix . . . 88
excel_critical_value . . . 89
excel_generic_format . . . 91
excel_matrix . . . 93
extract_components . . . 95
extract_tirt_params . . . 96
fixed . . . 97
flatten_list . . . 98

generate_comparisons_matrix . . . 98
generate_correlation_matrix . . . 99
generate_data . . . 100
generate_factor . . . 101
generate_matrix_A . . . 102
generate_matrix_lambda_hat . . . 102
generate_missing . . . 103
generate_multiple_response_vector . . . 103
generate_string . . . 104
generate_unique_comparisons_index . . . 105
get_fwd . . . 105
get_script_directory . . . 106
icc_cfa . . . 107
increase_index . . . 107
install_all_packages . . . 108
install_load . . . 108
k_fold . . . 109
k_sample . . . 110
key_to_cfa_model . . . 111
matrix_triangle . . . 111
min_max_index . . . 112
model_loadings . . . 113
name_triplet_pairs . . . 114
off_diagonal_index . . . 115
outlier_summary . . . 116
output_compare_model_logistic . . . 116
output_separator . . . 117
padNA . . . 118
plot_acf . . . 119
plot_boxplot . . . 120
plot_cfa . . . 120
plot_cfa_gg . . . 121
plot_confusion . . . 122
plot_confusion_extended . . . 123
plot_corrplot . . . 124
plot_crosstable . . . 125
plot_duplicate_y_axis . . . 126
plot_hinvert_title_grob . . . 127
plot_histogram . . . 127
plot_icc_thurstonian . . . 128
plot_interaction . . . 129
plot_irt_onefactor . . . 130
plot_loadings . . . 131
plot_logistic_model . . . 132
plot_mosaic . . . 133
plot_mtmm . . . 134
plot_multiplot . . . 135
plot_normality_diagnostics . . . 136

plot_oneway . . . 137
plot_oneway_diagnostics . . . 139
plot_outlier . . . 140
plot_qq . . . 141
plot_response_frequencies . . . 141
plot_roc . . . 142
plot_scatterplot . . . 143
plot_scree . . . 145
plot_separability . . . 145
plot_trees_xgboost . . . 146
plot_ts . . . 147
proportion_accurate . . . 148
questions_by_keys . . . 149
questions_dimensions_dataframe . . . 150
rad2deg . . . 151
rank_df_to_binary . . . 151
rank_to_binary . . . 152
rank3_to_triplets . . . 153
rbind_all . . . 153
recode_scale_dummy . . . 154
remove_nc . . . 155
remove_outliers . . . 156
remove_user_packages . . . 157
replace_na_with_previous . . . 157
report_alpha . . . 158
report_cfa . . . 160
report_choric_serial . . . 160
report_correlation . . . 162
report_dataframe . . . 163
report_efa . . . 164
report_factorial_anova . . . 165
report_hlr . . . 169
report_irt . . . 170
report_lda . . . 170
report_logistic . . . 171
report_manova . . . 173
report_normality_tests . . . 175
report_oneway . . . 176
report_pdf . . . 179
report_regression . . . 180
report_ttests . . . 182
report_wtests . . . 185
report_xgboost . . . 187
response_dimension . . . 189
response_frequency . . . 190
result_confusion_performance . . . 191
round_dataframe . . . 192
score_tirt . . . 193

score_tirt_pattern . . . 195
simulate_cfa_fit . . . 196
simulate_correlation_from_sample . . . 198
stat_word_char . . . 199
str_aes . . . 200
str_count . . . 201
str_mgsub . . . 201
str_pad . . . 202
str_proper . . . 203
str_replace . . . 203
str_replace_all . . . 204
str_split . . . 205
str_split_df . . . 205
str_split_fixed . . . 206
str_squish . . . 207
str_sub . . . 208
str_trim_df . . . 208
str_wrap . . . 209
swap . . . 210
symmetric_matrix . . . 210
tag_pos . . . 211
ts_smoothing . . . 212
write_txt . . . 213

Index **215**

`c_bind` *Column-bind data frames or vectors of unequal lengths*

Description

Combines any number of data frames or vectors side by side, padding shorter inputs with `NA` rows so all columns reach the same length. Each input's columns are prefixed with the object's name to avoid duplicate column names. Vectors are coerced to single-column data frames before binding.

Usage

```
c_bind(..., first = TRUE)
```

Arguments

`...`	Data frames or vectors to column-bind. Names are taken from the unevaluated expressions passed (e.g. variable names).
`first`	Logical. When `TRUE` (default) `NA` padding rows are appended at the bottom of shorter inputs; when `FALSE` they are prepended at the top.

Value

A data frame with one column per column across all inputs, padded with NA rows to the length of the longest input. Column names follow the pattern <object_name> for single-column inputs and <object_name>_<original_colname> for multi-column inputs.

Author(s)

Ananda Mahto

Examples

```
c_bind(rnorm(10), rnorm(11), rnorm(12), rnorm(13))
```

---

call_to_string *Convert a model call to a compact string*

Description

Extracts the call from a model object and returns it as a single whitespace-free string. Tries model$call first, falling back to model$Call if the first is NULL.

Usage

```
call_to_string(model)
```

Arguments

model	A model object with a call or Call element (e.g. from lm, glm, coxph).

Value

A character scalar with the model call, whitespace removed.

Examples

```
df <- generate_correlation_matrix()
model <- lm(df$X1 ~ df$X2)
call_to_string(model)
```

cdf *Check dataframe*

Description

Produces a column-level diagnostic summary of a dataframe, reporting missing values, data types, range statistics, and optionally unique values and factor levels. Returns a named list with a per-column table and a whole-dataframe summary. Can optionally export results to an `.xlsx` file.

Usage

```
cdf(
  df,
  name_length = (getOption("width")/3),
  digits = 2,
  nuniques = 0,
  parralel = FALSE,
  file = NULL
)
```

Arguments

`df`	A `data.frame` to inspect. Accepts any column types: numeric, integer, character, factor, logical, `Date`, `POSIXct`.
`name_length`	Integer. Maximum number of characters shown for column names and for the MIN and MAX values in the printed output; longer strings are truncated. The default is one third of the console width, `getOption("width") / 3`.
`digits`	Integer. Number of decimal places used when rounding MEAN, MEDIAN, and SD for numeric columns. Defaults to `2`.
`nuniques`	Integer. If `> 0`, appends a UNIQUES column to the output, plus a LEVELS column when at least one column is a factor. Columns with more distinct entries than `nuniques` are summarised as `"N Uniques"` / `"N Levels"`. Set to `0` to skip (faster). Defaults to `0`.
`parralel`	Logical. If `TRUE`, uses `future.apply` with a `multisession` plan across all available cores. Recommended for wide dataframes (> 100 columns) or very large `n`. Defaults to `FALSE`.
`file`	Character or `NULL`. If a string is provided, exports results to `<file>.xlsx` with two sheets: `variables` and `summary`. Any existing file with the same name is overwritten. Defaults to `NULL`.

Value

A named `list` with two elements:

`$summary` A single-row `data.frame` with whole-dataframe counts: COLLUMNS, ROWS, TOTAL, EMPTY, null, NAN, na, INF, FIN, FACTOR.

`$check` A per-column `data.frame` with the following fields:

NAMES Column name (truncated to `name_length`).

EMPTY Count of `""` empty strings.

null Count of `NULL` values (always 0 for dataframe columns).

na Count of `NA` values.

NOT_NA Count of non-`NA` values.

NAN Count of `NaN` values.

INF Count of `Inf` and `-Inf` values.

FIN Count of finite values.

RANGE Number of distinct values.

MEAN Arithmetic mean, rounded to `digits`. `NA` for non-numeric columns.

MEDIAN Median, rounded to `digits`. `NA` for non-numeric columns.

SD Standard deviation, rounded to `digits`. `NA` for non-numeric columns.

MIN Minimum value or first label in sorted order.

MAX Maximum value or last label in sorted order.

MODE Storage mode as returned by `mode()`.

TYPE Type as returned by `typeof()`.

CLASS Class as returned by `class()`; several classes are joined with `", "` (e.g. `"ordered, factor"`).

FACTOR Logical; `TRUE` if the column is a factor.

Note

MEAN, MEDIAN, and SD are `NA` for non-numeric columns. MIN and MAX for non-double columns are derived from `sort()` on character representations — natural sort ordering is not guaranteed for mixed alphanumeric strings.

Examples

```
# cdf(df = mtcars, parralel = TRUE)
cdf(df = change_data_type(mtcars, "factor"), nuniques = 3)
cdf(df = data.frame(t(mtcars)), file = file.path(tempdir(), "mtcars"), nuniques = 10)
cdf(df = mtcars)
cdf(df = generate_missing(mtcars))
cdf(df = infert, nuniques = 10)
cdf(df = infert)
df <- data.frame(infert,
  date = seq(as.Date("2010-1-1"),
    as.Date("2020-1-1"),
    length.out = nrow(infert)
  )
)
cdf(df = df)
```

cdff *Check dataframe (optimised)*

Description

A faster equivalent of cdf. Produces the same column-level diagnostic summary (except for the cases listed in Note) but computes every statistic, including UNIQUES and LEVELS, in a single pass over each column, builds the result without rbind calls, and removes the gtools and plyr dependencies. Recommended for large dataframes (> 100k rows #' or > 50 columns).

Usage

```
cdff(
  df,
  name_length = (getOption("width")/3),
  digits = 2,
  nuniques = 0,
  parralel = FALSE,
  file = NULL
)
```

Arguments

df	A data.frame to inspect. Accepts any column types: numeric, integer, character, factor, logical, Date, POSIXct.
name_length	Integer. Maximum number of characters shown for column names and for the MIN and MAX values in the printed output; longer strings are truncated. The default is one third of the console width, getOption("width") / 3.
digits	Integer. Number of decimal places used when rounding MEAN, MEDIAN, and SD for numeric columns. Defaults to 2.
nuniques	Integer. If > 0, appends a UNIQUES column to the output, plus a LEVELS column when at least one column is a factor. Columns with more distinct entries than nuniques are summarised as "N Uniques" / "N Levels". Set to 0 to skip (faster). Defaults to 0.
parralel	Logical. If TRUE, uses future.apply with a multisession plan across all available cores. Recommended for wide dataframes (> 100 columns) or very large n. Defaults to FALSE.
file	Character or NULL. If a string is provided, exports results to <file>.xlsx with two sheets: variables and summary. Any existing file with the same name is overwritten. Defaults to NULL.

Value

Identical structure to cdf: a named list with elements $summary and $check. See cdf for full field descriptions.

Note

MIN and MAX are only reported where the values have a natural order: numeric columns (double or integer) give the smallest and largest value, ordered factors the lowest and highest level that occurs, and dates or date-times the earliest and latest value. Unordered factors, character and logical columns have no natural order, so their MIN and MAX are NA (use nuniques to list their values instead), as are columns with only missing values. This differs from cdf, which reports the first and last label in sorted order for non-double columns.

MEAN, MEDIAN and SD are NA (rather than NaN) for numeric columns without any observed value. UNIQUES lists the values in their natural order (numbers by value, factor levels in level order, dates by time), leaving out NA and NaN; RANGE still counts them.

Examples

```
# cdff(df = mtcars, parralel = TRUE)
cdff(df = change_data_type(mtcars, "factor"), nuniques = 3)
cdff(df = data.frame(t(mtcars)), file = file.path(tempdir(), "mtcars"), nuniques = 10)
cdff(df = mtcars)
cdff(df = generate_missing(mtcars))
cdff(df = infert, nuniques = 10)
cdff(df = infert)
df <- data.frame(infert,
  date = seq(as.Date("2010-1-1"),
    as.Date("2020-1-1"),
    length.out = nrow(infert)
  )
)
cdff(df = df)
```

---

cfa_icc_index *index of items to convert from lavaan to thurstonian order for analysis*

Description

index of items to convert from lavaan to thurstonian order for analysis

Usage

```
cfa_icc_index(nitems, nfactors = 3)
```

Arguments

nitems	number of items in the questionnaire
nfactors	number of factors

Examples

```
cfa_icc_index(nitems=18,nfactors=3)
```

change_data_type *Convert column data types in a data frame*

Description

Converts all or selected columns in a data frame to a specified data type. Whitespace (tabs, carriage returns, newlines) is trimmed automatically when converting to `"character"` or `"numeric"`.

Usage

```
change_data_type(df, type)
```

Arguments

`df`	A data frame whose columns will be converted.
`type`	Character string specifying the conversion to apply:
	`"character"` Converts all columns to character, trimming leading and trailing whitespace.
	`"numeric"` Converts all columns to numeric (via character with whitespace trimming). Non-numeric strings become `NA`.
	`"factor"` Converts all columns to factor.
	`"factor_character"` Converts only factor columns to character. All other columns are left unchanged.
	`"character_factor"` Converts only character columns to factor. All other columns are left unchanged.

Value

A data frame with the same dimensions as `df` with column types converted as specified.

Examples

```
cdf(df = change_data_type(df = mtcars, "character"))
cdf(df = change_data_type(df = mtcars, "numeric"))
cdf(df = change_data_type(df = mtcars, "factor"))
df <- change_data_type(df = mtcars, "factor")
cdf(df = change_data_type(df = df, "factor_character"))
cdf(df = change_data_type(df = df, "character_factor"))
```

check_heywood *Check for Heywood Cases and Related SEM Estimation Problems*

Description

Screens a fitted lavaan model for common warning signs such as negative variances, impossible standardized values, unusually large standard errors, and convergence failure.

In simple terms: this is a quick model health check. It tells you whether your solution contains suspicious estimates that often indicate misspecification, weak identification, or numerical instability.

Usage

```
check_heywood(fit_model, verbose = TRUE)
```

Arguments

`fit_model`	A fitted lavaan model object, for example the result of a call to `lavaan::cfa()`, `lavaan::sem()` or related wrappers.
`verbose`	Logical. If `TRUE` (default), prints diagnostic sections and a summary to the console. If `FALSE`, only returns results.

Details

The function checks:

- Negative variances (`~~` with `lhs == rhs` and estimate < 0).
- Negative residual variances in Thurstonian style parameters (`~*~` with estimate < 0).
- Standardized loadings outside [-1, 1].
- Standardized correlations outside [-1, 1].
- Extremely large standard errors (`se > 10`).
- Non-convergence.

A Heywood case usually refers to impossible estimates like negative variances or standardized loadings greater than 1 in absolute value.

Value

An invisible list with:

- `has_issues`: Logical, `TRUE` if any issue was detected.
- `issues`: Named list of detected issue tables / messages.
- `converged`: `lavaan::lavInspect(fit_model, "converged")`, the logical convergence flag of the fit.

Examples

```
library(lavaan)

# Example model
HS.model <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
'

fit <- cfa(HS.model, data = HolzingerSwineford1939)

# Verbose diagnostic output
chk <- check_heywood(fit, verbose = TRUE)

# Programmatic use
check_heywood(fit, verbose = TRUE)
```

`clear_stopwords` *Remove stopwods*

Description

Remove stopwods

Usage

```
clear_stopwords(text, stopwords = stopwords::stopwords("english"))
```

Arguments

`text`	character vector
`stopwords`	character words to remove

Examples

```
text1<-"word_one word_two word_three"
text2<-"word_three word_four word_six"
text3<-"All the Lorem Ipsum generators on the Internet tend to repeat predefined
chunks as necessary, making this the first true generator on the Internet."
text4<-"It uses a dictionary of over 200 Latin words, combined with a handful of
model sentence structures, to generate Lorem Ipsum which looks reasonable."
text5<-"The generated Lorem Ipsum is therefore always free from repetition,
injected humour, or non-characteristic words etc."
stopwords<-stopwords::stopwords("english")
text<-c(text1,text2,text3,text4,text5)
clear_stopwords(text,stopwords=stopwords)
```

clear_text *Clear text*

Description

Clear text

Usage

```
clear_text(text)
```

Arguments

text	character vector

Examples

```
text1<-"word_one word_two word_three"
text2<-"word_three word_four word_six"
text3<-"All the Lorem Ipsum generators on the Internet tend to repeat predefined
chunks as necessary, making this the first true generator on the Internet."
text4<-"It uses a dictionary of over 200 Latin words, combined with a handful of
model sentence structures, to generate Lorem Ipsum which looks reasonable."
text5<-"The generated Lorem Ipsum is therefore always free from repetition,
injected humour, or non-characteristic words etc."
text<-c(text1,text2,text3,text4,text5)
clear_text(text)
```

combine_files *Combine all files in a directory into a single file*

Description

Lists the files in `input_dir` that match `pattern`, reads each one and writes their contents, in alphabetical order, to `output_file`. Each file's contents are preceded by a `# FILE: <path>` header block so the origin of every section is visible. If `output_file` lies inside `input_dir`, it is excluded so the file does not include itself. If no file matches, an empty `output_file` is written.

Usage

```
combine_files(
  input_dir = "working_functions",
  output_file = "all_functions.R",
  pattern = "\\.[Rr]$",
  recursive = TRUE
)
```

Arguments

input_dir	Character string. Directory to read files from. Default "working_functions".
output_file	Character string. Path of the combined file to write. An existing file is overwritten. Default "all_functions.R".
pattern	Regular expression that selects the files, passed to base::list.files(). Default "\\.[Rr]$" selects R scripts. Use NULL to include every file.
recursive	Logical. If TRUE (default), files in subdirectories of input_dir are included; if FALSE, only files directly in input_dir.

Value

Invisibly returns a character vector with the paths of the combined files. Called for its side effect of writing output_file.

Examples

```
# input_dir <- file.path(tempdir(), "combine_files_example")
# dir.create(file.path(input_dir, "sub"), recursive = TRUE, showWarnings = FALSE)
# writeLines("a <- 1", file.path(input_dir, "a.R"))
# writeLines("b <- 2", file.path(input_dir, "sub", "b.R"))
# output_file <- file.path(tempdir(), "combined.R")
# combine_files(input_dir = input_dir, output_file = output_file)
# cat(readLines(output_file), sep = "\n")
# combine_files(input_dir = input_dir, output_file = output_file, recursive = FALSE)
```

---

comparison_combinations

All pairwise column name combinations

Description

Generates a data frame of all pairwise combinations of column names from a data frame. Useful for programmatically specifying variable pairs to pass to functions like compute_crosstable or plot_crosstable.

Usage

```
comparison_combinations(df, all_orders = TRUE)
```

Arguments

df	A data frame whose column names will be combined.
all_orders	Logical. When TRUE (default) both orderings of each pair are included (e.g. (X1, X2) and (X2, X1)), producing $n(n-1)$ rows for n columns. When FALSE only unique unordered pairs are returned, producing $n(n-1)/2$ rows.

Value

A data frame with two character columns `X1` and `X2`, each row representing one variable pair.

Examples

```
comparison_combinations(generate_correlation_matrix(n = 10)[, 1:4])
```

`compute_ability` *Compute subject ability for thurstonian models*

Description

Computes person ability for binary thurstonian coded items for a single dimension

Usage

```
compute_ability(
  response,
  eta,
  gamma,
  lambda,
  psi,
  plot = FALSE,
  map = compute_map(eta = eta, mean = 0, sd = 1)
)
```

Arguments

`response`	item responses
`eta`	eta or ability
`gamma`	gamma or threshold
`lambda`	lambda or loading
`psi`	psi or error
`plot`	if TRUE plots icc curves using the plot_icc_thurstonian function
`map`	vector from compute_map

Examples

```
gamma<-c(0.556,-1.253,-1.729,0.618,0.937,0.295,-0.672,-1.127,-0.446,0.632,1.147,0.498)
psi<-c(2.172,1.883,2.055,1.869,2.231,2.100,1.762,1.803,1.565,1.892,1.794,1.686)
lambda<-c(1.082,1.082,-1.297,-1.297,0.802,0.802,1.083,1.083)
gamma<-gamma[response_dimension(c(1:12),3,c(1,2))]
psi<-psi[response_dimension(c(1:12),3,c(1,2))]
eta<-seq(-6,6,by=0.1)
response1<-c(0,0,0,0,0,0,0,0)
response2<-c(1,1,1,1,1,1,1,1)
```

```
response3<-c(1,0,1,0,1,0,1,0)
response4<-c(0,1,0,1,0,1,0,1)
map<-compute_map(eta=eta,mean=0,sd=1)
compute_ability(response1,eta,gamma,lambda,psi,map=map,plot=FALSE)
compute_ability(response2,eta,gamma,lambda,psi,map=map,plot=FALSE)
compute_ability(response3,eta,gamma,lambda,psi,map=map,plot=FALSE)
compute_ability(response4,eta,gamma,lambda,psi,map=map,plot=FALSE)
```

`compute_adjustment` *Compute multiple comparison alpha adjustments*

Description

Calculates Bonferroni and Šidák corrected alpha thresholds for a given family-wise alpha level and number of tests.

Usage

```
compute_adjustment(a, ntests)
```

Arguments

`a`	Numeric. The desired family-wise alpha level (e.g. `0.05`).
`ntests`	Integer. The number of tests (comparisons) being performed.

Value

A named list with two elements:

sidak Šidák corrected alpha: $1 - (1 - \alpha)^{1/k}$.

bonferroni Bonferroni corrected alpha: α/k.

Examples

```
compute_adjustment(0.05, 100)
```

`compute_aggregate` *Aggregate descriptive statistics by group*

Description

Computes a comprehensive set of descriptive statistics for all numeric columns in a data frame, stratified by one or more grouping variables. Unlike `compute_descriptives`, this function operates on every numeric column simultaneously and returns results in a long format where each row corresponds to one statistic for one group combination. Results can be exported to an Excel file.

Usage

```
compute_aggregate(df, iv, file = NULL)
```

Arguments

`df`	A data frame containing the variables of interest. All numeric columns that are not listed in `iv` are summarised.
`iv`	Integer vector of column indices identifying the grouping variables used to stratify the output.
`file`	Character string naming the output Excel file (without extension). When `NULL` (default) no file is written.

Value

A data frame in long format with one row per statistic per group combination. The first column is `statistic` (see below), followed by the grouping variable columns, and then one column per numeric variable in `df`. The `statistic` column takes the following values:

mean Arithmetic mean.

SD Standard deviation.

median Median.

mad Median absolute deviation.

trimmed mean 50% trimmed mean.

N Number of non-missing observations.

min Minimum observed value.

max Maximum observed value.

range Difference between maximum and minimum.

skewness Skewness; deviations from 0 indicate departure from symmetry.

kurtosis Excess kurtosis; deviations from 0 indicate departure from normality.

IQR Interquartile range (Q0.75 - Q0.25).

SE Standard error of the mean.

Examples

```
file = file.path(tempdir(), "descriptives")
compute_aggregate(df = mtcars, iv = 9)
compute_aggregate(df = mtcars, iv = 9:10)
compute_aggregate(df = mtcars, iv = 11)
compute_aggregate(df = mtcars, iv = 11, file = file)
```

`compute_alpha_diagnostics` *Item-total correlations and alpha-if-item-removed diagnostics*

Description

Computes item-level reliability diagnostics for a unidimensional scale. For each item the function returns the corrected and uncorrected item-total correlations and Cronbach's alpha recalculated with that item excluded. Use this alongside `compute_raw_alpha` to identify items that reduce scale reliability.

Usage

```
compute_alpha_diagnostics(df)
```

Arguments

`df`	A data frame whose columns are the items of a single scale. All columns must be numeric.

Value

A data frame with one row per item and the following columns:

item Item name.

alpha.if.item.removed Cronbach's alpha computed on the remaining items after excluding this item.

item.total.correlation Pearson correlation between this item and the sum of all items (uncorrected).

item.total.correlation.r.drop Pearson correlation between this item and the sum of all other items (corrected item-total correlation, also known as r-drop). Values below 0.30 typically indicate a problematic item.

Examples

```
set.seed(12345)
df <- data.frame(matrix(.5, ncol = 6, nrow = 6))
correlation_martix <- as.matrix(df)
diag(correlation_martix) <- 1
df <- round(generate_correlation_matrix(correlation_martix, nrows = 1000), 0) + 5
psych::alpha(df)
compute_alpha_diagnostics(df = df)
```

compute_aov_es *Effect Sizes for ANOVA Models*

Description

Builds the ANOVA table of a between-subjects aov model with Type I, II or III sums of squares and adds six effect sizes for every term:

- etasq: eta-squared (η^2)
- partial_etasq: partial eta-squared (η_p^2)
- omegasq: omega-squared (ω^2)
- partial_omegasq: partial omega-squared (ω_p^2)
- epsilonsq: epsilon-squared (ϵ^2)
- cohens_f: Cohen's f, computed from η_p^2

In simple terms, this shows for every factor and interaction of an ANOVA how much of the variance in the outcome it explains.

Usage

```
compute_aov_es(model, ss = "I")
```

Arguments

model	An aov model of a between-subjects design (without an Error() term).
ss	Type of sums of squares: "I" (sequential, the default), "II" or "III".

Details

SS_{total} is the sum of the sums of squares of all the terms and the residuals in the table, MS_{error} the residual mean square and N the number of observations used by the model.

The non-partial measures (etasq, omegasq, epsilonsq) divide by the total variance, so they depend on the other factors in the design. The partial measures (partial_etasq, partial_omegasq, cohens_f) divide by the variance of the term and the error only, which makes them easier to compare across designs. In a one-way design the two are the same. etasq and partial_etasq are biased upwards in small samples; omegasq, partial_omegasq and epsilonsq correct for that and are negative when $F < 1$. Negative values are returned as they are, as in sjstats::anova_stats; effectsize reports them as 0.

Types of sums of squares:

- "I": sequential; each term is adjusted for the terms before it, so the result depends on the order of the terms in the formula.
- "II": each term is adjusted for all the other terms that do not contain it (main effects are not adjusted for their interactions).

- `"III"`: each term is adjusted for all the other terms. It needs sum-to-zero contrasts, set with `options(contrasts = c("contr.sum", "contr.poly"))` before the model is fitted; with the default treatment contrasts the Type III sums of squares of main effects are not meaningful.

In a balanced design the three types give the same results. In an unbalanced design Type II and III sums of squares do not add up to the total variance of the outcome; SS_{total} is then the sum of the table, as in `effectsize`.

Rules of thumb for `etasq`, `omegasq`, `epsilonsq` and `cohens_f`. Small, medium and large are the benchmarks of Cohen (1988); tiny, very large and huge convert the Cohen's d benchmarks of Sawilowsky (2009) with $\eta^2 = d^2/(d^2 + 4)$ and $f = d/2$:

- tiny: η^2 < 0.01, f < 0.10 (d < 0.2)
- small: η^2 0.01 to < 0.06, f 0.10 to < 0.25 (d = 0.2)
- medium: η^2 0.06 to < 0.14, f 0.25 to < 0.40 (d = 0.5)
- large: η^2 0.14 to < 0.26, f 0.40 to < 0.60 (d = 0.8)
- very large: η^2 0.26 to < 0.50, f 0.60 to < 1.00 (d = 1.2)
- huge: η^2 >= 0.50, f >= 1.00 (d = 2.0)

These are rough guides; what counts as a meaningful effect depends on the field.

The effect sizes match the results of `effectsize::eta_squared`, `effectsize::omega_squared` and `effectsize::epsilon_squared`. They also match `effectsize::cohens_f` (apart from the truncation of negative values) and `sjstats::anova_stats`.

Value

A data frame with one row per term, one for the residuals and, with `ss = "III"`, one for the intercept:

- `call`: model formula
- `ss`: type of sums of squares
- `comparisons`: term
- `Df`, `Sum Sq`, `Mean Sq`, `F value`, `Pr(>F)`: the ANOVA table
- `etasq`: SS_{effect}/SS_{total}
- `partial_etasq`: $SS_{effect}/(SS_{effect} + SS_{error})$
- `omegasq`: $(SS_{effect} - df_{effect}MS_{error})/(SS_{total} + MS_{error})$
- `partial_omegasq`: $df_{effect}(MS_{effect} - MS_{error})/(SS_{effect} + (N - df_{effect})MS_{error})$
- `epsilonsq`: $(SS_{effect} - df_{effect}MS_{error})/SS_{total}$
- `cohens_f`: $\sqrt{\eta_p^2/(1 - \eta_p^2)}$

The effect sizes are `NA` for the residuals and the intercept.

Source

The computation of `partial_omegasq` is adapted from the answer of Stephen Martin (2014) to the question "Omega squared for measure of effect in R?" on Cross Validated, https://stats.stackexchange.com/a/126520, licensed CC BY-SA 3.0. The Type II and III sums of squares come from `car::Anova` (Fox & Weisberg, 2019).

References

Ben-Shachar, M. S., Lüdecke, D., & Makowski, D. (2020). effectsize: Estimation of effect size indices and standardized parameters. Journal of Open Source Software, 5(56), 2815. doi:10.21105/joss.02815

Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Erlbaum Associates.

Fox, J., & Weisberg, S. (2019). An R companion to applied regression (3rd ed.). Sage. https://www.john-fox.ca/Companion/

Hays, W. L. (1963). Statistics for psychologists. Holt, Rinehart and Winston.

Kelley, T. L. (1935). An unbiased correlation ratio measure. Proceedings of the National Academy of Sciences, 21(9), 554-559. doi:10.1073/pnas.21.9.554

Martin, S. (2014, December 3). Answer to "Omega squared for measure of effect in R?" Cross Validated. https://stats.stackexchange.com/a/126520

Olejnik, S., & Algina, J. (2003). Generalized eta and omega squared statistics: Measures of effect size for some common research designs. Psychological Methods, 8(4), 434-447. doi:10.1037/1082-989X.8.4.434

Sawilowsky, S. S. (2009). New effect size rules of thumb. Journal of Modern Applied Statistical Methods, 8(2), 597-599. doi:10.22237/jmasm/1257035100

Examples

```
one_way_between <- aov(uptake ~ Treatment, data = CO2)
compute_aov_es(model = one_way_between, ss = "I")
sjstats::anova_stats(one_way_between, digits = 10)
effectsize::omega_squared(one_way_between, partial = FALSE)

# Type III sums of squares need sum-to-zero contrasts
old_contrasts <- options(contrasts = c("contr.sum", "contr.poly"))
factorial_between <- aov(uptake ~ Treatment * Type, data = CO2)
compute_aov_es(model = factorial_between, ss = "I")
compute_aov_es(model = factorial_between, ss = "II")
compute_aov_es(model = factorial_between, ss = "III")
sjstats::anova_stats(car::Anova(factorial_between, type = 3), digits = 10)
effectsize::omega_squared(car::Anova(factorial_between, type = 3), partial = TRUE)
options(old_contrasts)
```

`compute_cohens_d` *Compute Cohen's D Effect Size*

Description

Computes Cohen's d effect size for a two-group comparison as abs(mean1 - mean2) / sd_pooled, with sd_pooled = sqrt((sd1^2 + sd2^2) / 2). This is the same formula used inline by report_ttests, extracted here as a standalone, reusable function so it can be called on its own.

Usage

```
compute_cohens_d(formula, data)
```

Arguments

formula	a formula of the form dv ~ group, same as used by stats::t.test's formula method. The grouping variable must have exactly two levels. Note: as with stats::t.test's formula method, paired tests are not supported here; use stats::t.test(x, y, paired = TRUE) directly for that case.
data	data frame containing the variables in formula, same as stats::t.test's data argument.

Value

A single numeric value, Cohen's d effect size.

Note

effect size

- Very small 0.01 Sawilowsky (2009)
- Small 0.20 Cohen (1988)
- Medium 0.50 Cohen (1988)
- Large 0.80 Cohen (1988)
- Very large 1.20 Sawilowsky (2009)
- Huge 12.0 Sawilowsky (2009)

Examples

```
compute_cohens_d(
  formula = bp_before ~ agegrp,
  data = df_blood_pressure[df_blood_pressure$agegrp %in% c("30-45", "46-59"), ]
)
effectsize::cohens_d(
  bp_before ~ agegrp,
  data = df_blood_pressure[df_blood_pressure$agegrp %in% c("30-45", "46-59"), ],
  pooled_sd = TRUE
)
```

compute_confidence_inteval *Compute confidence interval*

Description

Compute confidence interval

Usage

```
compute_confidence_inteval(vector)
```

Arguments

vector	vector

Examples

```
set.seed(1)
vector <- rnorm(1000)
compute_confidence_inteval(vector)
```

compute_crosstable *Pairwise cross-tabulation of categorical variables*

Description

Computes contingency tables (frequency counts and percentages) for pairs of categorical variables. Variable pairs can be supplied explicitly via `combinations`, or all unique pairs within a set of columns can be generated automatically via `factor_index`.

Usage

```
compute_crosstable(df, factor_index = NULL, combinations = NULL)
```

Arguments

df	A data frame containing the variables to cross-tabulate.
factor_index	Integer vector of column indices. When provided and `combinations` is `NULL`, all unique pairwise combinations of the selected columns are computed (self-pairs and duplicate pairs are excluded).
combinations	A data frame with two character columns named `index1` and `index2`, each row specifying one variable pair to cross-tabulate. This argument takes precedence over `factor_index`.

Value

A data frame with one row for each combination of levels of each pair of variables, with the following columns:

f1 Name of the first variable.

f2 Name of the second variable.

l1 Level of the first variable.

l2 Level of the second variable.

Frequency Observed count for the `l1` × `l2` cell.

Percent Cell count as a percentage of all observations in that variable pair (`Frequency / total * 100`).

Variable pairs with zero total observations are silently dropped.

Examples

```
combinations <- data.frame(index1 = c("vs", "am", "gear"),
                           index2 = c("cyl", "cyl", "cyl"))
compute_crosstable(df = mtcars, combinations = combinations)
combinations <- data.frame(index1 = c("vs", "am"),
                           index2 = c("cyl", "cyl"))
compute_crosstable(df = mtcars, combinations = combinations)
compute_crosstable(df = mtcars, factor_index = 8:10)
```

`compute_descriptives` *Descriptive statistics*

Description

Computes a comprehensive set of descriptive statistics for one or more continuous variables, optionally stratified by one or more grouping (independent) variables. When `iv` is supplied the statistics are computed separately for each combination of dependent and independent variable levels. Results can be exported to an Excel file.

Usage

```
compute_descriptives(df, dv, iv = NULL, file = NULL)
```

Arguments

`df`	A data frame containing the variables of interest.
`dv`	Integer vector of column indices identifying the dependent (continuous) variables to summarise.
`iv`	Integer vector of column indices identifying the independent (grouping) variables used to stratify the output. When `NULL` (default) the statistics are computed on the full sample.
`file`	Character string naming the output Excel file (without extension). When `NULL` (default) no file is written.

Value

A data frame with one row per variable (and per group level when `iv` is supplied) containing the following columns:

factor Name of the grouping variable (`iv` only).

levels Level of the grouping variable (`iv` only).

variable Name of the dependent variable.

n Sample size (non-missing observations).

mean Arithmetic mean.

sd Standard deviation.

median Median.

trimmed 10% trimmed mean.

mad Median absolute deviation.

min Minimum observed value.

max Maximum observed value.

range Difference between maximum and minimum.

skew Skewness; deviations from 0 indicate departure from symmetry.

kurtosis Excess kurtosis; deviations from 0 indicate departure from normality.

se Standard error of the mean.

IQR Interquartile range (Q0.75 - Q0.25).

Q0.1, Q0.25, Q0.5, Q0.75, Q0.9 Percentiles at 10, 25, 50, 75, and 90.

Examples

```
file = file.path(tempdir(), "descriptives_factor")
compute_descriptives(df = mtcars, dv = 1:5)
compute_descriptives(df = mtcars, dv = 1:2, iv = 9:10)
compute_descriptives(df = mtcars, dv = 1:2)
compute_descriptives(df = mtcars, dv = 1:2, iv = 9:10, file = file)
```

---

compute_dissatenuation

Compute the disattenuation correction for measurement error

Description

Estimates the true correlation between two variables by correcting the observed correlation for attenuation due to measurement error in both variables.

Usage

```
compute_dissatenuation(variable1, error1, variable2, error2)
```

Arguments

variable1	Numeric vector. True scores for the first variable.
error1	Numeric vector. Measurement error for variable1. Must be the same length as variable1.
variable2	Numeric vector. True scores for the second variable.
error2	Numeric vector. Measurement error for variable2. Must be the same length as variable2.

Details

The observed correlation is computed from the error-contaminated scores (variable + error). Reliability for each variable is estimated as the ratio of true score variance to total observed variance. The disattenuated correlation is then:

$$\rho = \frac{r_{obs}}{\sqrt{R_1 \cdot R_2}}$$

where R_1 and R_2 are the reliability estimates.

Value

A numeric scalar. The disattenuated (corrected) correlation between variable1 and variable2.

Examples

```
set.seed(1)
compute_dissatenuation(rnorm(10), rnorm(10), rnorm(10), rnorm(10))
```

---

compute_dummy_comparisons

Compute number of dummy comparisons

Description

Compute number of dummy comparisons

Usage

```
compute_dummy_comparisons(items)
```

Arguments

items	number of items per block

Examples

```
compute_dummy_comparisons(1)
compute_dummy_comparisons(2)
compute_dummy_comparisons(3)
compute_dummy_comparisons(4)
compute_dummy_comparisons(5)
compute_dummy_comparisons(6)
```

`compute_efa_cfa` *Exploratory and confirmatory factor analysis for each dimension*

Description

For every dimension of a questionnaire, fits three exploratory factor analysis (EFA) models with `psych::fa` and two one-factor confirmatory factor analysis (CFA) models with `lavaan::cfa`, and collects their loadings, fit indices, modification indices, residuals, factor scores and score correlations. All results are also written to an Excel workbook.

Usage

```
compute_efa_cfa(
  dimensions,
  df_response,
  filename = "efa_cfa_models.xlsx",
  mini = 1,
  maxi = 5
)
```

Arguments

`dimensions`	Data frame with one row per item and the columns `dimension` (the dimension the item belongs to), `subquestion_text` (the name of the item's column in `df_response`) and `key` (`1` for an item that is reverse scored, `-1` for an item scored as it is; see Details).
`df_response`	Data frame of numeric item responses with one column per item, named as in `dimensions$subquestion_text`. Missing responses must be `NA`; recode missing-data codes (such as 0) to `NA` first.
`filename`	Path of the Excel file to write. Its folder is created if it does not exist. Default `"efa_cfa_models.xlsx"` in the working directory.
`mini, maxi`	Lowest and highest possible response. They are used to reverse score items (`mini + maxi - x`). Defaults 1 and 5.

Details

The models fitted for each dimension are

1. Model 1: one-factor EFA (maximum likelihood).
2. Model 2: two-factor EFA (maximum likelihood, oblimin rotation).
3. Model 3: three-factor EFA (maximum likelihood, oblimin rotation).
4. Model 4: one-factor CFA (`missing = "ML"`, with means).
5. Model 5: Model 4 plus a random intercept factor `RI`, with all its loadings fixed to 1 and unrelated to the dimension's factor.

An EFA model that is not identified for the number of items (negative degrees of freedom), or that fails, is reported as `NA`: the one-factor model needs at least 3 items, the two-factor model 5 and the three-factor model 6. The CFA models need at least 4 items, because with 3 items the random intercept model has more parameters than the data can identify (lavaan fits it without a warning, so the function skips it).

The EFA and CFA models use the responses as they are, so items worded in the opposite direction have negative loadings. The mean scores and the correlation sheets use reverse scored items: `psych::reverse.code` is called with `key * -1`, so the items with `key = 1` are reversed.

Warnings from lavaan (for example about respondents without any answer) are shown as messages and do not stop the CFA. If a CFA model stops with an error or does not converge, the CFA columns of that dimension are `NA` and the function continues with the next dimension. In the lavaan models, a dimension name that is not a syntactically valid R name (for example one with spaces) is changed with `make.names`.

The function stops with an error if `dimensions` lacks one of its columns, if `key` has values other than 1 and -1, or if an item is not a numeric column of `df_response`. It warns about responses outside `mini` to `maxi` and about items without variance (no responses, or the same response from everyone), because the EFA and CFA of their dimension fail.

The function needs the rwf functions `model_loadings`, `c_bind`, `change_data_type`, `matrix_triangle`, `excel_critical_value` and `excel_matrix`.

Value

A list with

efa_cfa_models One row per item with n, dimension, variable and F1 (Model 1 loadings); S2F1, S2F2 (Model 2 structure matrix, the correlations of the items with the rotated factors); P2F1, P2F2 (Model 2 pattern matrix, the regression weights of the rotated factors); S3F1 to S3F3 and P3F1 to P3F3 (the same for Model 3); index with f1_v, f2_v, f3_v (EFA fit of Models 1 to 3: RMSEA with its lower and upper limits and confidence level, RMS, CFI, TLI); unstandardized and standardized (Model 4 loadings); unstandardized_RI, standardized_RI, RI_u and RI_s (Model 5 loadings and random intercept loadings); index_cfa with fit_CFA and fit_CFA_RI (CFA fit of Models 4 and 5: RMSEA with its limits, SRMR, CFI, TLI). A dimension with fewer than 7 items gets extra rows (without a variable) for the EFA fit indices.

cfa_scores Factor scores of Model 4 (`lavaan::lavPredict`), one column per dimension whose CFA ran and one row per row of `df_response` (`NA` for respondents without any answer in that dimension).

plots_scree An empty list (the scree plot code is switched off).

cfa_coefficients Model 4 loadings (lambda), residual variances (theta) and intercepts, standardized and unstandardized, one row per item.

The Excel file has the sheets critical (efa_cfa_models, with loadings beyond 0.32 or below -0.32 highlighted), CFA_Modification_Indexes (Model 4 modification indices of 10 or more), EFA1F_Residuals (Model 1 residual correlations beyond 0.1 in absolute value), CFA_Coefficients, CFA_RI_Coefficients, used_items (correlations of the items with the dimension mean scores), used_items_item_correlation (item pairs sorted by the size of their correlation), used_items_mean_score and used_items_cfa_score (correlations between the dimension mean scores and between the CFA scores). A sheet without any content (for example when no CFA ran) is not written.

Note

Most of the run time is spent writing the Excel file, because `excel_critical_value` adds conditional formatting one cell at a time.

Examples

```
items<-c(paste0("E",1:10),paste0("N",1:10),paste0("A",1:10),
         paste0("C",1:10),paste0("O",1:10))
reverse_items<-c("E2","E4","E6","E8","E10","N2","N4","A1","A3","A5",
                 "A7","C2","C4","C6","C8","O2","O4","O6")
df<-df_ocean[,items]
df[df==0]<-NA # 0 is the missing-data code of df_ocean
dimensions<-data.frame(dimension=substr(items,1,1),
                       subquestion_text=items,
                       key=ifelse(items%in%reverse_items,1,-1))
result<-compute_efa_cfa(dimensions=dimensions,
                        df_response=df[1:2000,],
                        filename=file.path(tempdir(),"efa_cfa_ocean.xlsx"))
head(result$efa_cfa_models)
round(cor(result$cfa_scores,use="pairwise.complete.obs"),2)
```

---

compute_frequencies *Frequency table for categorical variables*

Description

Computes frequency counts, proportions, and percentages for every column in a data frame. All columns are processed together and the results are stacked into a single long-format table. Missing values are excluded from the frequency counts via `table()`. Results can be exported to an Excel file.

Usage

```
compute_frequencies(df, file = NULL)
```

Arguments

df	A data frame with the categorical variables to tabulate as columns. All columns are processed regardless of class.
file	Character string naming the output Excel file (without extension). When NULL (default) no file is written.

Value

A data frame in long format with one row per observed level per variable, containing the following columns:

variable Name of the column from df.

Observation Observed level (category label).

Frequency Count of observations at that level.

Proportion Relative frequency (Frequency / total for that variable).

Percent Proportion multiplied by 100.

Examples

```
file = file.path(tempdir(), "descriptives")
set.seed(123)
df <- generate_missing(generate_factor(nrows = 10, ncols = 10), missing = 5)
compute_frequencies(df = df)
compute_frequencies(df = generate_factor())
compute_frequencies(df = generate_factor(), file = file)
```

---

compute_friedman_test *Friedman Test with Effect Size*

Description

Runs the Friedman rank-sum test for a complete block design (repeated measures) and returns the test statistic, p-value, and Kendall's W (kendall_w) as the effect size.

In simple terms, this tests whether the same subjects (blocks) respond differently across conditions (groups), and quantifies how consistently the subjects rank the conditions in the same order.

Usage

```
compute_friedman_test(formula, df, ci = FALSE, conf.level = 0.95, nboot = 1000)
```

Arguments

formula	A formula in the form y ~ group | block, where group is the repeated condition and block identifies the subject.
df	A data frame in long format containing the variables in formula, with one row per block and group.
ci	If TRUE, adds a percentile bootstrap confidence interval for kendall_w.
conf.level	Confidence level of the interval.
nboot	Number of bootstrap resamples.

Details

The observations are ranked within each block. With n blocks, k groups, R_j the rank sum of group j and t the sizes of the groups of tied values within blocks,

$$Q = \frac{12 \sum_{j=1}^{k} \left(R_j - n(k+1)/2\right)^2}{nk(k+1) - \sum(t^3 - t)/(k-1)}$$

Under the null hypothesis Q approximately follows a chi-squared distribution with $k - 1$ degrees of freedom.

kendall_w is in [0, 1]. 0 means the blocks rank the groups in no consistent order; 1 means every block ranks the groups in the same order.

As in rstatix::friedman_effsize, the rules of thumb for kendall_w are the Cohen (1988) benchmarks for correlations:

- tiny: < 0.1
- small: 0.1 to < 0.3
- medium: 0.3 to < 0.5
- large: >= 0.5

These are rough guides; what counts as a meaningful effect depends on the field.

Rows with a missing group or block are removed. Blocks with a missing value in any group are removed, as in the default method of stats::friedman.test; n reports how many blocks remain. A block with more than one observation for the same group is an error.

The confidence interval resamples blocks with replacement nboot times, recomputes kendall_w in each resample and takes the $(1 - conf.level)/2$ and $1 - (1 - conf.level)/2$ quantiles. Use set.seed for reproducible intervals.

Value

A one-row data frame with:

- formula: model formula used
- method: test name
- kendall_w: Kendall's coefficient of concordance, $Q/(n(k-1))$
- kendall_w_lower, kendall_w_upper: confidence interval of kendall_w (only if ci = TRUE)

- `Q`: Friedman chi-squared statistic, corrected for ties
- `df`: degrees of freedom ($k - 1$)
- `n`: number of complete blocks used
- `p`: p-value

Source

`Q` is computed with the same tie-corrected formula as `stats::friedman.test` (R Core Team, R package `stats`, licensed GPL-2 | GPL-3).

References

Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Erlbaum Associates.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 32(200), 675-701. doi:10.1080/01621459.1937.10503522

Kendall, M. G., & Babington Smith, B. (1939). The problem of m rankings. The Annals of Mathematical Statistics, 10(3), 275-287. doi:10.1214/aoms/1177732186

R Core Team (2026). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. doi:10.32614/R.manuals

Examples

```
form <- formula(uptake ~ conc | Plant)
friedman.test(formula = form, data = df_co2)
rstatix::friedman_effsize(df_co2, form, ci = TRUE, conf.level = 0.95,
                          ci.type = "perc", nboot = 100)
effectsize::kendalls_w(form, data = df_co2)
compute_friedman_test(formula = form, df = df_co2)
set.seed(1)
compute_friedman_test(formula = form, df = df_co2, ci = TRUE)
```

---

compute_icc_thurstonian

Compute item characteristic curves for thurstonian models

Description

Computes icc curves for binary thurstonian coded items for a single dimension

Usage

```
compute_icc_thurstonian(eta, gamma, lambda, psi, plot = FALSE)
```

Arguments

eta	eta or ability
gamma	gamma or threshold
lambda	lambda or loading
psi	psi or error
plot	if TRUE plots icc curves using the plot_icc_thurstonian function

Examples

```
gamma<-c(0.556,-1.253,-1.729,0.618,0.937,0.295,-0.672,-1.127,-0.446,0.632,1.147,0.498)
psi<-c(2.172,1.883,2.055,1.869,2.231,2.100,1.762,1.803,1.565,1.892,1.794,1.686)
lambda<-c(1.082,1.082,-1.297,-1.297,0.802,0.802,1.083,1.083)
gamma<-gamma[response_dimension(c(1:12),3,c(1,2))]
psi<-psi[response_dimension(c(1:12),3,c(1,2))]
eta<-seq(-6,6,by=0.01)
compute_icc_thurstonian(eta=eta,gamma=gamma,lambda=lambda,psi=psi,plot=FALSE)
```

---

compute_info_1pl *Compute item information for 1PL model*

Description

Compute item information for 1PL model

Usage

```
compute_info_1pl(b, theta)
```

Arguments

b	numeric difficulty parameter
theta	numeric theta

Examples

```
compute_info_1pl(b=1,theta=-3)
compute_info_1pl(b=1,theta=-2)
compute_info_1pl(b=1,theta=-1)
compute_info_1pl(b=1,theta=0)
compute_info_1pl(b=1,theta=1)
compute_info_1pl(b=1,theta=2)
compute_info_1pl(b=1,theta=3)
ti<-compute_info_1pl(b=1,theta=seq(-6,6,by=.01)) # test information
plot(ti,x=seq(-6,6,by=.01))
```

compute_info_2pl *Compute item information for 2PL model*

Description

Compute item information for 2PL model

Usage

```
compute_info_2pl(a, b, theta)
```

Arguments

a	numeric discrimination parameter
b	numeric difficulty parameter
theta	numeric theta

Examples

```
compute_info_2pl(a=1.5,b=1,theta=-3)
compute_info_2pl(a=1.5,b=1,theta=-2)
compute_info_2pl(a=1.5,b=1,theta=-1)
compute_info_2pl(a=1.5,b=1,theta=0)
compute_info_2pl(a=1.5,b=1,theta=1)
compute_info_2pl(a=1.5,b=1,theta=2)
compute_info_2pl(a=1.5,b=1,theta=3)
ti<-compute_info_2pl(a=1,b=-2,theta=seq(-6,6,by=.01)) # test information
plot(ti,x=seq(-6,6,by=.01))
ti<-compute_info_2pl(a=2,b=0,theta=seq(-6,6,by=.01)) # test information
plot(ti,x=seq(-6,6,by=.01))
ti<-compute_info_2pl(a=3,b=2,theta=seq(-6,6,by=.01)) # test information
plot(ti,x=seq(-6,6,by=.01))
```

compute_info_3pl *Compute item information for 3PL model*

Description

Compute item information for 3PL model

Usage

```
compute_info_3pl(a, b, g, theta)
```

Arguments

a	numeric discrimination parameter
b	numeric difficulty parameter
g	numeric guessing parameter
theta	numeric theta

Examples

```
compute_info_3pl(a=1.5,b=1,g=.2,theta=-3)
compute_info_3pl(a=1.5,b=1,g=.2,theta=-2)
compute_info_3pl(a=1.5,b=1,g=.2,theta=-1)
compute_info_3pl(a=1.5,b=1,g=.2,theta=0)
compute_info_3pl(a=1.5,b=1,g=.2,theta=1)
compute_info_3pl(a=1.5,b=1,g=.2,theta=2)
compute_info_3pl(a=1.5,b=1,g=.2,theta=3)
ti<-compute_info_3pl(a=1.5,b=1,g=.2,theta=seq(-6,6,by=.01)) # test information
plot(ti,x=seq(-6,6,by=.01))
```

---

compute_kruskal_wallis_test

Kruskal-Wallis Test with Effect Sizes

Description

Runs a one-way Kruskal-Wallis rank-sum test and returns the test statistic, p-value, and two effect sizes:

- `etasq`: eta-squared for Kruskal-Wallis (η^2_H)
- `epsilonsq`: epsilon-squared (ϵ^2)

In simple terms, this tests whether groups differ in their distributions, and quantifies how large that group effect is.

Usage

```
compute_kruskal_wallis_test(
  formula,
  df,
  ci = FALSE,
  conf.level = 0.95,
  nboot = 1000
)
```

Arguments

formula	A one-way formula in the form y ~ group.
df	A data frame containing the variables in formula.
ci	If TRUE, adds percentile bootstrap confidence intervals for both etasq and epsilonsq.
conf.level	Confidence level of the intervals.
nboot	Number of bootstrap resamples.

Details

epsilonsq is in [0, 1]. etasq is at most 1 and is negative when $H < k - 1$, that is when the groups differ less than expected by chance. Multiplying by 100 gives an approximate percentage-style interpretation of explained rank variance.

Rules of thumb for etasq and epsilonsq. Small, medium and large are the η^2 benchmarks of Cohen (1988); tiny, very large and huge convert the Cohen's d benchmarks of Sawilowsky (2009) with $\eta^2 = d^2/(d^2 + 4)$:

- tiny: < 0.01 (d < 0.2)
- small: 0.01 to < 0.06 (d = 0.2)
- medium: 0.06 to < 0.14 (d = 0.5)
- large: 0.14 to < 0.26 (d = 0.8)
- very large: 0.26 to < 0.50 (d = 1.2)
- huge: >= 0.50 (d = 2.0)

These are rough guides; what counts as a meaningful effect depends on the field. epsilonsq is biased upwards by about $(k - 1)/(n - 1)$ in small samples.

Rows with a missing value in the outcome or the grouping variable are removed before the test, as in stats::kruskal.test.

The confidence intervals resample rows with replacement nboot times, recompute both effect sizes in each resample and take the $(1 - conf.level)/2$ and $1 - (1 - conf.level)/2$ quantiles. Use set.seed for reproducible intervals.

Value

A one-row data frame with:

- formula: model formula used
- method: test name
- etasq: Kruskal-Wallis eta-squared, $(H - k + 1)/(n - k)$
- etasq_lower, etasq_upper: confidence interval of etasq (only if ci = TRUE)
- epsilonsq: epsilon-squared, $H/(n - 1)$
- epsilonsq_lower, epsilonsq_upper: confidence interval of epsilonsq (only if ci = TRUE)
- H: Kruskal-Wallis chi-squared statistic
- df: degrees of freedom $(k - 1)$
- p: p-value

Source

The computation of H, including the correction for ties, is adapted from stats::kruskal.test (R Core Team, R package stats, licensed GPL-2 | GPL-3). etasq and epsilonsq use the same formulas as rstatix::kruskal_effsize and effectsize::rank_epsilon_squared.

References

Ben-Shachar, M. S., Lüdecke, D., & Makowski, D. (2020). effectsize: Estimation of effect size indices and standardized parameters. Journal of Open Source Software, 5(56), 2815. doi:10.21105/joss.02815

Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Erlbaum Associates.

Kassambara, A. (2026). rstatix: Pipe-friendly framework for basic statistical tests (R package version 1.1.0). doi:10.32614/CRAN.package.rstatix

R Core Team (2026). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. doi:10.32614/R.manuals

Sawilowsky, S. S. (2009). New effect size rules of thumb. Journal of Modern Applied Statistical Methods, 8(2), 597-599. doi:10.22237/jmasm/1257035100

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

Examples

```
form <- formula(bp_before ~ agegrp)
kruskal.test(formula = form, data = df_blood_pressure)
rcompanion::epsilonSquared(
  x = df_blood_pressure$bp_before,
  g = df_blood_pressure$agegrp,
  group = "row",
  ci = TRUE,
  conf = 0.95,
  type = "perc",
  R = 1000,
  digits = 3
)
rstatix::kruskal_effsize(df_blood_pressure, form, ci = TRUE, conf.level = 0.95,
                         ci.type = "perc", nboot = 100)
compute_kruskal_wallis_test(formula = form, df = df_blood_pressure)
set.seed(1)
compute_kruskal_wallis_test(formula = form, df = df_blood_pressure, ci = TRUE)
```

---

compute_kurtosis *Compute kurtosis of a numeric vector*

Description

Calculates the excess kurtosis of a numeric vector using the b_2 formula consistent with MINITAB and BMDP. Missing values are removed before computation.

Usage

```
compute_kurtosis(vector)
```

Arguments

vector	Numeric vector.

Value

A numeric scalar. A value of 0 indicates a normal distribution; positive values indicate heavier tails (leptokurtic); negative values indicate lighter tails (platykurtic).

Note

Formula used: $b_2 = m_4/s^4 - 3 = (g_2 + 3)(1 - 1/n)^2 - 3$. Used in MINITAB and BMDP. Results match `e1071::kurtosis()` with `type = 2`.

Examples

```
set.seed(1)
vector <- rnorm(1000)
compute_kurtosis(vector)
e1071::kurtosis(vector)
```

---

compute_map *Simulate prior distribution*

Description

Simulate prior distribution

Usage

```
compute_map(eta, mean = 0, sd = 1)
```

Arguments

eta	vector
mean	numeric
sd	numeric

Examples

```
eta<-seq(-6,6,by=0.1)
compute_map(eta=eta,mean=0,sd=1)
```

compute_mean_sd_alpha *Mean and SD of scale scores*

Description

Computes the mean and standard deviation of respondent scale scores. When `divisor` is `NULL` the scale score is the row mean across items. When `divisor` is supplied the scale score is the row sum divided by `divisor`, which is useful when scores need to be expressed on a custom metric (e.g. dividing a sum score by the maximum possible score to obtain a proportion).

Usage

```
compute_mean_sd_alpha(df, divisor = NULL)
```

Arguments

`df`	A data frame whose columns are the items of a single scale. All columns must be numeric.
`divisor`	Numeric scalar used to divide the row sum before computing the mean and SD. When `NULL` (default) row means are used instead.

Value

A one-row data frame with columns `MEAN` and `SD` containing the mean and standard deviation of the scale scores across respondents.

Examples

```
set.seed(12345)
df <- data.frame(matrix(.5, ncol = 6, nrow = 6))
correlation_martix <- as.matrix(df)
diag(correlation_martix) <- 1
df <- round(generate_correlation_matrix(correlation_martix, nrows = 1000), 0) + 5
compute_mean_sd_alpha(df)
compute_mean_sd_alpha(df, divisor = 100)
```

compute_moving_average *Centered moving average*

Description

Smooths every numeric column of a data frame by replacing each value with the mean of a symmetric window of `2w + 1` observations (the `w` rows before, the row itself, and the `w` rows after). At the edges of the series the window is clipped to the available rows, so the average is computed over fewer observations.

Usage

```
compute_moving_average(df, w)
```

Arguments

df	A data frame whose columns will be smoothed. All columns should be numeric.
w	Integer half-window size. The total window width is 2w + 1. For example, w = 5 averages 11 consecutive rows centred on each observation.

Value

A data frame with the same dimensions and column names as df where each value has been replaced by its centred moving average.

Examples

```
compute_moving_average(df = mtcars, w = 5)
```

---

compute_one_way_test *One-Way ANOVA with Effect Sizes and Power*

Description

Runs a one-way analysis of variance for two or more independent groups, assuming equal variances (Fisher's F test) or not (Welch's F test), and returns the sums of squares, mean squares, F statistic, p-value, observed power, and five effect sizes:

- etasq: eta-squared (η^2)
- partial.etasq: partial eta-squared (η_p^2)
- omegasq: omega-squared (ω^2)
- partial.omegasq: partial omega-squared (ω_p^2)
- cohens.f: Cohen's f

In simple terms, this tests whether the group means differ, and quantifies how much of the variance in the outcome the groups explain.

Usage

```
compute_one_way_test(formula, df, var.equal = TRUE)
```

Arguments

formula	A one-way formula in the form y ~ group.
df	A data frame containing the variables in formula.
var.equal	If TRUE, assumes equal variances (Fisher's F test). If FALSE, uses Welch's F test.

Details

In a one-way design `partial.etasq` equals `etasq` and `partial.omegasq` equals `omegasq`. `etasq` and `partial.etasq` are in [0, 1] and are biased upwards in small samples, by about $(k-1)/(N-1)$ when there is no effect. `omegasq` and `partial.omegasq` remove most of that bias and are negative when $F < 1$. Multiplying `etasq` or `omegasq` by 100 gives the percentage of variance explained by the groups.

Rules of thumb for `etasq`, `omegasq` and `cohens.f`. Small, medium and large are the benchmarks of Cohen (1988); tiny, very large and huge convert the Cohen's d benchmarks of Sawilowsky (2009) with $\eta^2 = d^2/(d^2 + 4)$ and $f = d/2$:

- tiny: η^2 < 0.01, f < 0.10 (d < 0.2)
- small: η^2 0.01 to < 0.06, f 0.10 to < 0.25 (d = 0.2)
- medium: η^2 0.06 to < 0.14, f 0.25 to < 0.40 (d = 0.5)
- large: η^2 0.14 to < 0.26, f 0.40 to < 0.60 (d = 0.8)
- very large: η^2 0.26 to < 0.50, f 0.60 to < 1.00 (d = 1.2)
- huge: η^2 >= 0.50, f >= 1.00 (d = 2.0)

These are rough guides; what counts as a meaningful effect depends on the field.

`power` uses the noncentral F distribution with noncentrality parameter $\lambda = f^2 N$ (Cohen, 1988), evaluated at the observed `cohens.f`. Observed power is a function of the p-value and adds no information to it (Hoenig & Heisey, 2001); use power analysis with an expected effect size to plan a study, for example with `pwr::pwr.anova.test`.

Welch's test (Welch, 1951) has no sums of squares. With `var.equal = FALSE`, `ms_effect` and `ms_error` are the numerator and denominator of the Welch F statistic and `ss_effect` and `ss_error` are $MS \times df$. The resulting `etasq`, `omegasq`, `partial.omegasq` and `cohens.f` equal the conversions of the Welch F statistic in `effectsize::F_to_eta2`, `effectsize::F_to_omega2` and `effectsize::F_to_f`. Treat the Welch effect sizes as approximations.

Rows with a missing value in the outcome or the grouping variable are removed before the test, as in `stats::oneway.test`.

Value

A one-row data frame with:

- `formula`: model formula used
- `method`: "Assuming homoscedasticity" (Fisher) or "Assuming heteroscedasticity" (Welch)
- `ss_effect`, `ss_error`: sums of squares between and within groups
- `ms_effect`, `ms_error`: mean squares, SS/df
- `etasq`: eta-squared, SS_{effect}/SS_{total}
- `partial.etasq`: partial eta-squared, $SS_{effect}/(SS_{effect} + SS_{error})$
- `omegasq`: omega-squared, $(SS_{effect} - df_{effect} MS_{error})/(SS_{total} + MS_{error})$
- `partial.omegasq`: partial omega-squared, $df_{effect}(MS_{effect} - MS_{error})/(df_{effect} MS_{effect} + (df_{error} + 1) MS_{error})$
- `cohens.f`: Cohen's f, $\sqrt{\eta^2/(1 - \eta^2)}$

- `power`: observed power of the F test at $\alpha = 0.05$
- `statistic`: F statistic, MS_{effect}/MS_{error}
- `df_effect`: degrees of freedom of the effect ($k - 1$)
- `df_error`: degrees of freedom of the error ($N - k$, or the Welch degrees of freedom)
- p: p-value

Source

The group statistics and the Welch test (the weights, the Welch F statistic and its degrees of freedom) are adapted from `stats::oneway.test` (R Core Team, R package `stats`, licensed GPL-2 | GPL-3). The sums of squares, effect sizes and power are added in rwf.

References

Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Erlbaum Associates.

Hoenig, J. M., & Heisey, D. M. (2001). The abuse of power: The pervasive fallacy of power calculations for data analysis. The American Statistician, 55(1), 19-24. doi:10.1198/000313001300339897

Olejnik, S., & Algina, J. (2003). Generalized eta and omega squared statistics: Measures of effect size for some common research designs. Psychological Methods, 8(4), 434-447. doi:10.1037/1082-989X.8.4.434

R Core Team (2026). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. doi:10.32614/R.manuals

Sawilowsky, S. S. (2009). New effect size rules of thumb. Journal of Modern Applied Statistical Methods, 8(2), 597-599. doi:10.22237/jmasm/1257035100

Welch, B. L. (1951). On the comparison of several mean values: An alternative approach. Biometrika, 38(3/4), 330-336. doi:10.2307/2332579

Examples

```
form <- formula(bp_before ~ agegrp)
oneway.test(formula = form, data = df_blood_pressure, var.equal = TRUE)
oneway.test(formula = form, data = df_blood_pressure, var.equal = FALSE)
car::Anova(aov(form, data = df_blood_pressure), type = 2)
lsr::etaSquared(aov(form, data = df_blood_pressure), type = 3, anova = TRUE)
effectsize::omega_squared(aov(form, data = df_blood_pressure), partial = FALSE)
sjstats::anova_stats(lm(form, data = df_blood_pressure), digits = 22)
compute_one_way_test(formula = form, df = df_blood_pressure, var.equal = TRUE)
compute_one_way_test(formula = form, df = df_blood_pressure, var.equal = FALSE)
```

`compute_posthoc` *Tukey and Games-Howell Post Hoc Tests*

Description

Compares every pair of group means after a one-way ANOVA with two post hoc tests:

- `tukey`: the Tukey-Kramer test, which pools the variances of all groups and assumes they are equal
- `games.howell`: the Games-Howell test, which uses the variances of the two groups being compared and does not assume equal variances

In simple terms, after an ANOVA shows that the group means differ, this shows which pairs of groups differ, while keeping the chance of any false positive across all the pairs at 5%.

Usage

```
compute_posthoc(y, x)
```

Arguments

`y`	A numeric vector with the outcome.
`x`	A vector with the group of each value of `y`, a factor or a vector that can be converted to one.

Details

For groups i and j with means $\bar{y}$, variances s^2 and sizes n, out of k groups and N observations:

Tukey-Kramer (Tukey, 1953; Kramer, 1956) uses the pooled error variance $MS_{error} = \sum(n_j - 1)s_j^2/(N-k)$:

$$t_{ij} = \frac{|\bar{y}_i - \bar{y}_j|}{\sqrt{MS_{error}(1/n_i + 1/n_j)}}, \qquad df = N - k$$

Games-Howell (Games & Howell, 1976) uses the two group variances and the Welch-Satterthwaite degrees of freedom for each pair:

$$t_{ij} = \frac{|\bar{y}_i - \bar{y}_j|}{\sqrt{s_i^2/n_i + s_j^2/n_j}}, \qquad df_{ij} = \frac{(s_i^2/n_i + s_j^2/n_j)^2}{\frac{(s_i^2/n_i)^2}{n_i - 1} + \frac{(s_j^2/n_j)^2}{n_j - 1}}$$

In both tests the p-value comes from the studentized range distribution, $p = P(q_{k,df} \geq \sqrt{2}\, t_{ij})$, which already accounts for the number of groups. Do not adjust these p-values again.

Use Tukey after Fisher's F test (`compute_one_way_test(var.equal = TRUE)`) and Games-Howell after Welch's F test (`var.equal = FALSE`). Games-Howell is the safer choice when the group variances or sizes differ.

`t` is an absolute value; the direction of a difference comes from the group means. The p-values match `stats::TukeyHSD` and `rstatix::games_howell_test`.

Missing values are not removed: a missing value in `y` makes every result `NA`. Remove incomplete cases before calling the function, as `report_oneway` does.

Value

A list with:

- `input`: list with the `x` and `y` supplied
- `output`: list with two matrices, `tukey` and `games.howell`, each with one row per pair of groups (named `"group1:group2"` in the order of the factor levels) and the columns:
  - `t`: absolute t statistic of the pair
  - `df`: degrees of freedom
  - `p`: p-value, adjusted for all the pairwise comparisons

Source

Adapted from `posthocTGH()` in the `userfriendlyscience` package by Gjalt-Jorn Peters (Open University of the Netherlands) and Jeff Baggett (University of Wisconsin - La Crosse), licensed GPL (>= 3), https://github.com/Matherion/userfriendlyscience. `posthocTGH()` was in turn based on `games_howell.R`, a script hosted on the course page of Robert Cribbie (York University) at `http://www.psych.yorku.ca/cribbie/6130/games_howell.R`, which is no longer available.

References

Games, P. A., & Howell, J. F. (1976). Pairwise multiple comparison procedures with unequal n's and/or variances: A Monte Carlo study. Journal of Educational Statistics, 1(2), 113-125. doi:10.3102/10769986001002113

Kramer, C. Y. (1956). Extension of multiple range tests to group means with unequal numbers of replications. Biometrics, 12(3), 307-310. doi:10.2307/3001469

Peters, G.-J. Y. (2018). userfriendlyscience: Quantitative analysis made accessible (R package version 0.7.2). https://github.com/Matherion/userfriendlyscience

Tukey, J. W. (1953). The problem of multiple comparisons. Unpublished manuscript, Princeton University.

Examples

```
TukeyHSD(aov(bp_before ~ agegrp, data = df_blood_pressure))
rstatix::games_howell_test(df_blood_pressure, bp_before ~ agegrp)
compute_posthoc(y = df_blood_pressure$bp_before, x = df_blood_pressure$agegrp)
compute_posthoc(y = df_blood_pressure$bp_after, x = df_blood_pressure$agegrp)
```

---

compute_power_r *Compute r power curve*

Description

Compute r power curve

Usage

```
compute_power_r(
  n = 100,
  r = NULL,
  sig.level = 0.05,
  alternative = c("two.sided", "less", "greater"),
  title = "",
  base_size = 10
)
```

Arguments

n	number of observations
r	correlation coefficient
sig.level	alpha (type I error probability)
alternative	a character string specifying the alternative hypothesis, which must be one of "two.sided" (default), "greater" or "less"
title	plot title
base_size	base font size

Examples

```
compute_power_r(n=100,r=.5,sig.level=.05,alternative=c("two.sided"))
```

---

compute_power_r_matrix

Compute correlation matrix

Description

Compute correlation matrix

Usage

```
compute_power_r_matrix(m, ...)
```

Arguments

m	correlation matrix
...	arguments passed to compute_power_r

Examples

```
compute_power_r_matrix(m=stats::cor(mtcars,use="pairwise.complete.obs"),n=100)
```

compute_raw_alpha *Cronbach's alpha (raw)*

Description

Computes Cronbach's alpha directly from the covariance matrix of a unidimensional set of items using the formula $\alpha = \frac{k}{k-1}\left(1 - \frac{\sum \text{diag}(\Sigma)}{\sum \Sigma}\right)$, where k is the number of items and Σ is the covariance matrix. Pairwise complete observations are used so that missing data on individual items does not discard entire rows.

Usage

```
compute_raw_alpha(df)
```

Arguments

df	A data frame whose columns are the items of a single scale. All columns must be numeric.

Value

A single numeric value: Cronbach's alpha for the scale. Values above 0.70 are generally considered acceptable for research purposes.

Examples

```
set.seed(12345)
df <- data.frame(matrix(.5, ncol = 6, nrow = 6))
correlation_martix <- as.matrix(df)
diag(correlation_martix) <- 1
df <- round(generate_correlation_matrix(correlation_martix, nrows = 1000), 0) + 5
psych::alpha(df)
compute_raw_alpha(df = df)
```

compute_residual_stats *Residuals for matrices*

Description

Root Mean Squared Residual Number of absolute residuals > 0.05 Proportion of absolute residuals > 0.05. It can either accept a psych EFA model or it can compare two correlation or covariance matrices

Usage

```
compute_residual_stats(model, data = NULL)
```

Arguments

model	psych EFA model. It has to be a correlation or covariance matrix if data is not NULL
data	correlation or covariance matrix

Examples

```
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
compute_residual_stats(model)
```

---

compute_scores *Compute subject ability for thurstonian models*

Description

Computes person ability for binary thurstonian coded items for a single dimension

Usage

```
compute_scores(mydata, ...)
```

Arguments

mydata	item responses
...	arguments passed to compute_ability

Examples

```
gamma<-c(0.556,-1.253,-1.729,0.618,0.937,0.295,
        -0.672,-1.127,-0.446,0.632,1.147,0.498)
psi<-c(2.172,1.883,2.055,1.869,2.231,2.100,
       1.762,1.803,1.565,1.892,1.794,1.686)
lambda<-c(1.082,1.082,-1.297,-1.297,0.802,0.802,1.083,1.083)
gamma<-gamma[response_dimension(c(1:12),3,c(1,2))]
psi<-psi[response_dimension(c(1:12),3,c(1,2))]
eta<-seq(-6,6,by=0.1)
map<-compute_map(eta=eta,mean=0,sd=1)
response_df<-data.frame(matrix(nrow=0,ncol=8))
response_df[1,]<-c(0,0,0,0,0,0,0,0)
response_df[2,]<-c(1,1,1,1,1,1,1,1)
response_df[3,]<-c(1,0,1,0,1,0,1,0)
response_df[4,]<-c(0,1,0,1,0,1,0,1)
compute_scores(response_df,eta,gamma,lambda,psi,map=map,plot=FALSE)
```

compute_se_theta *Compute the SE of theta*

Description

Compute the SE of theta

Usage

```
compute_se_theta(info)
```

Arguments

info	numeric information

Examples

```
compute_se_theta(1)
ti<-compute_info_2pl(a=10,b=0,theta=seq(-3,3,by=.01)) # test information
plot(compute_se_theta(ti),x=seq(-3,3,by=.01))
```

compute_shrout *Shrout-Fleiss reliability coefficients*

Description

Computes five reliability coefficients from the Shrout and Fleiss (1979) framework using variance components extracted from a person × item × time mixed model (see extract_components). The coefficients cover between-person and within-person reliability under both fixed and random time designs, and are appropriate for longitudinal or repeated-measures measurement studies.

Usage

```
compute_shrout(sperson, spersonitem, stime, spersontime, serror, m, k)
```

Arguments

sperson	Variance component of the person main effect.
spersonitem	Variance component of the person × item interaction.
stime	Variance component of the time main effect.
spersontime	Variance component of the person × time interaction.
serror	Variance component of residual error.
m	Number of items (reports) averaged over.
k	Number of time points averaged over.

Value

A data frame with one row per reliability coefficient containing columns `measure`, `result`, and `description`:

r1f Between-person reliability of a single measure at fixed time points.

r1r Between-person reliability of a single measure at random time points (different people, different days).

rkf Between-person reliability of scores averaged over `m` items and `k` fixed time points.

rkr Between-person reliability of scores averaged over `k` random time points.

rc Within-person reliability of change across time points.

References

Shrout, P. E., & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin, 86*(2), 420–428.

Examples

```
design <- expand.grid(time = 1:3, item = 1:2, person = 1:10)
design <- change_data_type(design, type = "factor")
design$response <- rnorm(30, 0, 0.1)
model <- mixlm::lm(response ~ r(time) * r(person) + r(item) * r(person), data = design)
result <- extract_components(model)
vc <- result$components
compute_shrout(
  sperson = vc[2, 3], spersonitem = vc[5, 3], stime = vc[1, 3],
  spersontime = vc[4, 3], serror = vc[6, 3], 3, 3
)
```

---

compute_skewness *Compute skewness of a numeric vector*

Description

Calculates the skewness of a numeric vector using the b_1 formula consistent with MINITAB and BMDP. Missing values are removed before computation.

Usage

```
compute_skewness(vector)
```

Arguments

`vector`	Numeric vector.

Value

A numeric scalar. Positive values indicate right skew, negative values indicate left skew.

Note

Formula used: $b_1 = m_3/s^3 = g_1((n-1)/n)^{3/2}$. Used in MINITAB and BMDP. Results match `e1071::skewness()` with `type = 2`.

Examples

```
set.seed(1)
vector <- rnorm(1000)
compute_skewness(vector)
e1071::skewness(vector)
```

---

compute_solve *Solve Linear Systems or Invert a Matrix (Gauss-Jordan)*

Description

Solves matrix equations of the form A X = B using Gauss-Jordan elimination with partial pivoting.

In simple terms: this function takes a square matrix A and finds X so that multiplying A by X gives B.

If B is not provided, it uses the identity matrix and returns the inverse of A.

Usage

```
compute_solve(a, b)
```

Arguments

a	Numeric square matrix A.
b	Optional numeric vector or matrix B. Must have the same number of rows as A. If omitted, B is set to the identity matrix.

Details

Mathematical meaning:

$$AX = B$$

where A is known, B is known, and X is unknown.

Algebra meaning: each column of X is the solution to one linear system with the same A.

Computational method: the function builds the augmented matrix [A | B], then applies row operations until the left side becomes the identity matrix. At that point, the right side is X.

Partial pivoting is used for better numerical stability: in each column, it swaps in the row with the largest absolute pivot.

The function stops with an error if A is singular (non-invertible).

Value

If b is a matrix, returns matrix X solving A X = B.

If b is a vector, returns vector x solving A x = b.

If b is missing, returns the inverse of A.

Examples

```
# Example 1: solve A x = b
A <- matrix(c(2, 1,
1, 3), nrow = 2, byrow = TRUE)
b <- c(1, 2)
x <- compute_solve(A, b)
x
# Check: A %% x should equal b
A %% x

# Example 2: solve A X = B (multiple right-hand sides)
B <- cbind(c(1, 2), c(0, 1))
X <- compute_solve(A, B)
X
# Check: A %% X should equal B
A %% X

# Example 3: inverse of A (when b is omitted)
A_inv <- compute_solve(A)
A_inv
# Check: A %% A_inv should be identity
A %% A_inv
```

---

compute_standard *Compute standard scores from a numeric vector*

Description

Transforms a numeric vector into one of several standard score formats, including z-scores, T-scores, stens, stanines, percentiles, and others. Can operate on raw scores or pre-standardised z-scores.

Usage

```
compute_standard(vector, mean = 0, sd = 1, type = "z", input = "non_standard")
```

Arguments

vector	Numeric vector of raw scores or z-scores (see input).
mean	Numeric. Population mean used for "uz" and density types. Default is 0.
sd	Numeric. Population standard deviation used for "uz" and density types. Default is 1.

type Character. The output score type. One of:

"z" Z-scores (mean=0, sd=1).

"uz" Unstandardise: convert z-scores back to raw scores using supplied mean and sd.

"sten" Sten scores (1–10, mean=5.5, sd=2).

"t" T-scores (mean=50, sd=10).

"stanine" Stanine scores (1–9, mean=5, sd=2).

"center" Mean-centred scores.

"center_reversed" Reversed mean-centred scores.

"percent" Percentage of the maximum observed value.

"percentile" Cumulative normal percentile (0–100).

"scale_zero_one" Min-max scaled scores (0–1).

"normal_density" Normal density values.

"cumulative_density" Cumulative sum of the input vector.

"all" Returns a data frame with all score types, sorted by z-score.

input Character. Whether vector contains raw scores ("non_standard") or already-standardised z-scores ("standard"). Default is "non_standard".

Value

A numeric vector of transformed scores, or a data frame when type = "all".

Examples

```
vector <- c(rnorm(10), NA, rnorm(10))
compute_standard(vector, type = "z")
compute_standard(vector, mean = 0, sd = 1, type = "uz")
compute_standard(vector, type = "sten")
compute_standard(vector, type = "t")
compute_standard(vector, type = "stanine")
compute_standard(vector, type = "center")
compute_standard(vector, type = "center_reversed")
compute_standard(vector, type = "percent")
compute_standard(vector, type = "scale_zero_one")
ndf <- compute_standard(seq(-6, 6, .01), mean = 0, sd = 1, type = "normal_density")
plot(ndf)
cdf <- compute_standard(ndf, mean = 0, sd = 1, type = "cumulative_density")
plot(cdf)
compute_standard(vector, type = "all")
compute_standard(seq(-6, 6, .1), type = "all", input = "standard")
```

compute_standard_error

Compute the standard error of the mean

Description

Compute the standard error of the mean

Usage

```
compute_standard_error(vector)
```

Arguments

vector	Numeric vector. Missing values are removed before computation.

Value

A numeric scalar. The standard error of the mean.

Examples

```
set.seed(1)
vector <- rnorm(1000)
compute_standard_error(vector)
```

compute_text_similarity

Text similarity measures

Description

Text similarity measures

Usage

```
compute_text_similarity(text1, text2)
```

Arguments

text1	character vector
text2	character vector

Examples

```
text1<-"word_one word_two word_three"
text2<-"word_three word_four word_six"
text3<-"All the Lorem Ipsum generators on the Internet tend to repeat predefined
chunks as necessary, making this the first true generator on the Internet."
text4<-"It uses a dictionary of over 200 Latin words, combined with a handful of
model sentence structures, to generate Lorem Ipsum which looks reasonable."
text5<-"The generated Lorem Ipsum is therefore always free from repetition,
injected humour, or non-characteristic words etc."
text<-c(text1,text2,text3,text4,text5)
text<-unlist(strsplit(text,split=" "))
text1<-unlist(strsplit(text1,split=" "))
text2<-unlist(strsplit(text2,split=" "))
text3<-unlist(strsplit(text3,split=" "))
text4<-unlist(strsplit(text4,split=" "))
text5<-unlist(strsplit(text5,split=" "))
compute_text_similarity(text1,text1)
compute_text_similarity(text1,text2)
compute_text_similarity(text1,text3)
compute_text_similarity(text1,text4)
```

---

compute_tversky_index *Compute the Tversky index*

Description

Computes the Tversky index between two sets, a generalisation of the Jaccard and Sørensen–Dice similarity coefficients. The index measures the overlap between `x` and `y` relative to their differences, weighted by `alpha` and `beta`.

The Tversky index is defined as:

$$T(x, y) = \frac{|x \cap y|}{|x \cap y| + \alpha|x \setminus y| + \beta|y \setminus x|}$$

Special cases:

- `alpha` = `beta` = `0.5` — Sørensen–Dice coefficient
- `alpha` = `beta` = `1.0` — Jaccard index

Usage

```
compute_tversky_index(x, y, alpha = 0.5, beta = 0.5)
```

Arguments

x	A vector. Coerced to character before comparison.
y	A vector. Coerced to character before comparison.
alpha	Non-negative numeric. Weight applied to elements in x but not in y. Defaults to 0.5.
beta	Non-negative numeric. Weight applied to elements in y but not in x. Defaults to 0.5.

Value

A single numeric value in the range [0, 1], where 0 indicates no overlap and 1 indicates identical sets.

Note

Both x and y are treated as *sets* — duplicate elements within each vector are ignored. Inputs are coerced to character before comparison, so 1L and "1" are treated as equal.

Examples

```
x <- c("a", "b", "c", "d")
y <- c("b", "c", "d", "e")

# default (Sorensen-Dice)
compute_tversky_index(x, y)

# Jaccard index
compute_tversky_index(x, y, alpha = 1, beta = 1)

# asymmetric: penalise x-only elements more heavily
compute_tversky_index(x, y, alpha = 0.9, beta = 0.1)

# identical sets → 1
compute_tversky_index(x, x)

# disjoint sets → 0
compute_tversky_index(c("a", "b"), c("c", "d"))
```

---

compute_unidimensional_ability

Compute theta for unidimensional models

Description

Compute theta for unidimensional models

Usage

```
compute_unidimensional_ability(
  a,
  b,
  g = NULL,
  d = 1.702,
  u,
  lim_theta = c(-6, 6)
)
```

Arguments

a	numeric vector discrimination parameters
b	numeric vector difficulty parameters
g	numeric vector guessing parameters
d	numeric scaling constant usually it is a value that approximating 1.749
u	numeric vector responses
lim_theta	vector minimum and maximum value of theta

Examples

```
a<-c(0.39,0.45,0.52,0.3,0.35,0.43,0.42,0.44,0.34,0.42)
b<-c(-1.96,-1.9,-1.38,-0.58,0.48,-0.81,-0.35,1.59,1.33,2.93)
u<-c(1,1,1,1,0,0,1,0,1,0)
# SHOULD RETURN 0.48402574251176
compute_unidimensional_ability(a=a,b=b,u=u,d=1.7,g=NULL)
a<-c(1.27,0.9,0.94,0.95,0.55,0.6,0.44,0.4)
b<-c(-0.54,0.18,0.21,1.26,1.73,-0.87,1.72,2.67)
u<-c(1,1,1,1,0,0,0,0)
# SHOULD RETURN 1.04621621510192
compute_unidimensional_ability(a=a,b=b,u=u,d=1.7,g=NULL)
a<-c(0.41,0.32,0.33,1.2,0.63,0.62,0.7,0.61,0.38,0.53,0.6,1.16)
b<-c(-1.4,-1.3,-1.17,0.2,0.71,0.86,-0.12,0.12,2.06,1.38,1.18,-0.33)
u<-c(1,0,1,1,0,0,0,1,1,0,1,0)
# SHOULD RETURN 0.0860506282671103
compute_unidimensional_ability(a=a,b=b,u=u,d=1.7,g=NULL)
```

---

compute_unidimensional_theta

Compute theta for unidimensional models

Description

Compute theta for unidimensional models

Usage

```
compute_unidimensional_theta(a, b = 0, g = 0, i = 1, d = 1.702, theta = 0)
```

Arguments

a	numeric discrimination parameter
b	numeric difficulty parameter
g	numeric guessing parameter
i	numeric inattentiveness parameter
d	numeric scaling constant usually a value 1.749 or 1.702
theta	numeric or vector theta

Note

when scaling constant=1 it has no effect in equation
when inattentiveness=1 and guessing=0 function computes a 2PL score
when inattentiveness=1 and guessing!=0 function computes a 3PL score
when inattentiveness!=1 and guessing!=0 function computes a 4PL score

Examples

```
compute_unidimensional_theta(a=10,b=0)
x<-seq(-3,3,by=.01)
plot(compute_unidimensional_theta(a=5,b=0,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=5,b=-1,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=5,b=1,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=0,b=0,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=1,b=0,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=0,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=.1,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=.5,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=0,i=1,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=0,i=.9,theta=x),x=x, ylim = c(0, 1))
plot(compute_unidimensional_theta(a=10,b=0,g=0,i=.5,theta=x),x=x, ylim = c(0, 1))
```

---

compute_wilcoxon_effect_size

Compute Wilcoxon Effect Size

Description

Computes the Wilcoxon rank-sum / signed-rank effect size r = abs(Z) / sqrt(N), with Z derived from the p-value of stats::wilcox.test (Z = qnorm(p / 2, lower.tail = FALSE)). This avoids a dependency on rstatix / coin, using only stats::wilcox.test under the hood. The effect size magnitude is computed from the two-sided p-value regardless of the alternative used for the underlying hypothesis test.

Usage

```
compute_wilcoxon_effect_size(
  formula,
  data,
  mu = 0,
  exact = NULL,
  correct = TRUE,
  ...
)
```

Arguments

formula	a formula of the form dv ~ group, same as used by stats::wilcox.test's formula method. Note: as with stats::wilcox.test's formula method, paired tests are not supported here; use stats::wilcox.test(x, y, paired = TRUE) directly for that case.
data	data frame containing the variables in formula, same as stats::wilcox.test's data argument.
mu	a number specifying an optional shift, same as stats::wilcox.test.
exact	logical, as in stats::wilcox.test, indicating whether an exact p-value should be computed.
correct	logical, as in stats::wilcox.test, indicating whether to apply the continuity correction.
...	additional arguments passed to stats::wilcox.test.

Value

A single numeric value, the Wilcoxon effect size (r).

Examples

```
compute_wilcoxon_effect_size(
  formula = bp_before ~ agegrp,
  data = df_blood_pressure[df_blood_pressure$agegrp %in% c("30-45", "46-59"), ]
)
rstatix::wilcox_effsize(bp_before ~ agegrp,
  data = df_blood_pressure[df_blood_pressure$agegrp %in% c("30-45", "46-59"), ]
)
```

---

compute_y_logistic *Compute y for logistic function*

Description

This function requires x range to produce a vector with y values

Usage

```
compute_y_logistic(intercept, coefficient, x)
```

Arguments

intercept	Numeric
coefficient	Numeric
x	Numeric

Examples

```
x <- -10:10
compute_y_logistic(0, 1, x)
compute_y_logistic(0, 1, 1)
plot(x, compute_y_logistic(0, 1, x), type = "l")
grid()
abline(b = 0, a = .5)
```

---

confusion *Create a confusion matrix from observed and predicted vectors*

Description

Generates a confusion matrix from observed and predicted values.

Usage

```
confusion(observed, predicted)
```

Arguments

observed	Vector of observed variables. These are the true class labels.
predicted	Vector of predicted variables. These are the predicted class labels.

Details

This function creates a confusion matrix by comparing the observed (true) class labels with the predicted class labels. The confusion matrix is a table that is often used to describe the performance of a classification model. The function performs the following steps: 1. Identifies the unique class labels from both the observed and predicted vectors. 2. Sorts the class labels in a mixed order (if they are character variables) using ‘gtools::mixedsort‘. 3. Constructs a table to represent the confusion matrix with the sorted class labels as levels. The output is a confusion matrix,where rows represent the predicted class labels and columns represent the observed class labels.

Examples

```
# Example with numeric observed and predicted values
confusion(observed=c(1,2,3,4,5,10),predicted=c(1,2,3,4,5,11))
# Example with repeated observed and predicted values
confusion(observed=c(1,2,2,2,2),predicted=c(1,1,2,2,2))
```

`confusion_matrix_percent`

Confusion matrix with row and column percent

Description

Generates a confusion matrix from observed and predicted values,including row and column percentages.

Usage

```
confusion_matrix_percent(observed, predicted)
```

Arguments

`observed`	Vector of observed variables. These are the true class labels.
`predicted`	Vector of predicted variables. These are the predicted class labels.

Details

This function creates a confusion matrix by comparing the observed (true) class labels with the predicted class labels. Additionally, it calculates row and column percentages to provide a more detailed performance analysis.

The function performs the following steps: 1. Computes the confusion matrix from the observed and predicted values. 2. Calculates the overall accuracy by dividing the sum of diagonal elements by the total number of observations. 3. Appends row and column sums to the confusion matrix. 4. Computes precision and recall for each class and appends these metrics to the matrix. 5. Returns a formatted data frame with the confusion matrix,row and column percentages,and overall accuracy.

Note

Total measures-Accuracy: (TP+TN)/total
Total measures-Prevalence: (TP+FN)/total
Total measures-Proportion Incorrectly Classified: (FN+FP)/total
Horizontal measures-True Positive Rate-Sensitivity: TP/(TP+FN)
Horizontal measures-True Negative Rate-Specificity: TN/(FP+TN)
Horizontal measures-False Negative Rate-Miss Rate: FN/(TP+FN)
Horizontal measures-False Positive Rate-Fall-out: FP/(FP+TN)
Vertical measures-Positive Predictive value-Precision: TP/(TP+FP)
Vertical measures-Negative Predictive value: TN/(FN+TN)
Vertical measures-False Omission Rate: FN/(FN+TN)

Vertical measures-False Discovery Rate: FP/(TP+FP)

Examples

```
# Example with numeric observed and predicted values
confusion_matrix_percent(observed=c(1,2,3,4,5,10),predicted=c(1,2,3,4,5,11))

# Example with repeated observed and predicted values
confusion_matrix_percent(observed=c(1,2,2,2,2),predicted=c(1,1,2,2,2))

# Example with random observed and predicted values
observed<-factor(round(rnorm(10000,m=10,sd=1)))
predicted<-factor(round(rnorm(10000,m=10,sd=1)))
confusion_matrix_percent(observed,predicted)
```

---

convert_excel_unix_timestamp *Convert UNIX EXCEL timestamp*

Description

Convert UNIX EXCEL timestamp

Usage

```
convert_excel_unix_timestamp(timestamp)
```

Arguments

timestamp	unix or excel timestamp

Examples

```
convert_excel_unix_timestamp(1)
```

---

data_frame_index *dataframe index*

Description

dataframe index

Usage

```
data_frame_index(nrow, ncol)
```

Arguments

nrow	number of rows
ncol	number of collumns

Examples

```
data_frame_index(5, 5)
```

---

decompose_datetime *Decompose datetime objects to dataframe collumns*

Description

Decompose datetime objects to dataframe collumns

Usage

```
decompose_datetime(
  x,
  format = "",
  origin = "1970-01-01",
  tz = "GMT",
  extended = FALSE,
  breaks = c(-1, 5, 13, 16, 20, 23),
  ...
)
```

Arguments

x	datetime object
format	date time format
origin	Starting date. The default is the unix time origin "1970-01-01"
tz	Timezone
extended	if TRUE it will display additional day time categories WEEKDAY MONTH JULIAN QUARTER DAY_PERIOD
breaks	Numeric vector Breaks define hour of day for classifying into "Night", "Morning", "Noon", "Afternoon", "Evening".
...	arguments passed to as.POSIXct This argument is used if extended=TRUE

Examples

```
# a fixed date and time, so that the output is the same on every run
# (Sys.Date() and Sys.time() work as well)
d1 <- as.Date("2026-01-15")
d2 <- as.POSIXct("2026-01-15 14:30:00", tz = "GMT")
timestamp1 <- as.numeric(as.POSIXct(d1))
timestamp2 <- as.numeric(as.POSIXct(d2))
decompose_datetime(x = d1)
decompose_datetime(x = d2)
decompose_datetime(x = d1, extended = TRUE)
decompose_datetime(x = d2, extended = TRUE)
decompose_datetime(x = "01/15/1900", format = "%m/%e/%Y")
decompose_datetime(x = "01/15/1900", format = "%m/%e/%Y", extended = TRUE)
decompose_datetime(x = as.Date(as.POSIXct(10000, origin = "1970-01-01")))
decompose_datetime(
  x = as.Date(as.POSIXct(timestamp1, origin = "1970-01-01")),
  format = "%m/%e/%Y"
)
decompose_datetime(
  x = as.Date(as.POSIXct(timestamp2, origin = "1970-01-01")),
  format = "%m/%e/%Y"
)
```

---

deg2rad *Convert degrees to radians*

Description

Convert degrees to radians

Usage

```
deg2rad(degrees)
```

Arguments

degrees	degrees

Examples

```
deg2rad(180)
```

detach_package *Detach and unload a package*

Description

Removes a package from the R search path and unloads its namespace. If the package was attached more than once, all instances are removed. Does nothing if the package is not currently attached.

Usage

```
detach_package(package)
```

Arguments

package	Character string, the name of the package to detach (without the "package:" prefix), e.g. "ggplot2".

Value

Invisibly returns NULL. Called for its side effect of detaching the package.

df_admission *Admission Data*

Description

This data set contains information about graduate admission, including GRE scores, GPA, and the ranking of the undergraduate institution.

Usage

```
df_admission
```

Format

A data frame with 8 rows and 4 variables:

admit Binary variable indicating admission (0=No, 1=Yes)

gre GRE (Graduate Record Examination) score

gpa Grade Point Average

rank Ranking of the undergraduate institution (1=highest, 4=lowest)

Source

researchpy repo

Examples

```
data(df_admission)
head(df_admission)
```

`df_automotive_data` *Automotive Data*

Description

This data set contains a range of automotive information including engine location, dimensions, weight, engine type, number of cylinders, and other specifications.

Usage

```
df_automotive_data
```

Format

A data frame with 38 rows and 26 variables:

symboling Symboling code for the vehicle

normalized-losses Normalized losses in the vehicle

make Make of the vehicle

fuel-type Type of fuel used (e.g., gas, diesel)

aspiration Aspiration type (e.g., std, turbo)

num-of-doors Number of doors (e.g., two, four)

body-style Body style (e.g., sedan, hatchback, wagon)

drive-wheels Drive wheels (e.g., fwd, rwd, 4wd)

engine-location Location of the engine (e.g., front, rear)

wheel-base Wheelbase of the vehicle in inches

length Length of the vehicle in inches

width Width of the vehicle in inches

height Height of the vehicle in inches

curb-weight Curb weight of the vehicle in pounds

engine-type Type of engine (e.g., dohc, ohcv, ohc, l)

num-of-cylinders Number of cylinders in the engine

engine-size Size of the engine in cubic inches

fuel-system Fuel system used (e.g., mpfi, 2bbl, mfi, 1bbl)

bore Diameter of the cylinders in the engine

stroke Stroke length of the engine

compression-ratio Compression ratio of the engine

horsepower Horsepower generated by the engine

peak-rpm Peak RPM of the engine

city-mpg Miles per gallon in the city

highway-mpg Miles per gallon on the highway

price Price of the vehicle

Source

Downloaded from Kaggle.com by the user Ramakrishnan Srinivasan. see https://www.kaggle.com/toramky/automobile-dataset

Examples

```
data(df_automotive_data)
head(df_automotive_data)
```

---

df_blood_pressure *Blood Pressure Data*

Description

This data set contains blood pressure readings for patients before and after a certain treatment or intervention.

Usage

```
df_blood_pressure
```

Format

A data frame with 30 rows and 5 variables:

patient Unique identifier for each patient

sex Sex of the patient (e.g., Male, Female)

agegrp Age group of the patient (e.g., 30-45, 46-59)

bp_before Blood pressure reading before the intervention

bp_after Blood pressure reading after the intervention

Source

researchpy repo

Examples

```
data(df_blood_pressure)
head(df_blood_pressure)
```

`df_co2` *Carbon Dioxide Uptake in Grass Plants*

Description

The CO2 data frame has 84 rows and 5 columns of data from an experiment on the cold tolerance of the grass species Echinochloa crus-galli.

Usage

```
df_co2
```

Format

A data frame with 84 rows and 5 variables:

Plant an ordered factor with levels Qn1 < Qn2 < Qn3 < ... < Mc1 giving a unique identifier for each plant). Used as a grouping factor.

Type a factor with levels Quebec Mississippi giving the origin of the plant

Treatment a factor with levels nonchilled chilled

conc a numeric vector of ambient carbon dioxide concentrations (mL/L)

uptake a numeric vector of carbon dioxide uptake rates (in μmol/m^2/sec)

Details

Grouped formulas like `uptake ~ conc | Plant` are useful in lattice graphics and mixed-effect models. The vertical bar ('|') separates the grouping variable. This allows modeling or plotting the response (`uptake`) versus the predictor (`conc`) within each level of `Plant`.

Source

Potvin, C., Lechowicz, M. J. and Tardif, S. (1990) "The statistical analysis of ecophysiological response curves obtained from experiments involving repeated measures", Ecology, 71, 1389–1400.
Pinheiro, J. C. and Bates, D. M. (2000) Mixed-effects Models in S and S-PLUS, Springer.

Examples

```
data(df_co2)
head(df_co2)
```

df_crop_yield *Crop Yield Data*

Description

This data set contains information about crop yields based on different fertilizer types and water conditions.

Usage

```
df_crop_yield
```

Format

A data frame with 20 rows and 3 variables:

Fert Type of fertilizer used (A or B)

Water Watering condition (High or Low)

Yield Crop yield (in unspecified units)

Source

researchpy repo (simulated data, not real)

Examples

```
data(df_crop_yield)
head(df_crop_yield)
```

df_difficile *Difficile Data*

Description

This data set contains information about the impact of different doses on libido.

Usage

```
df_difficile
```

Format

A data frame with 15 rows and 3 variables:

person Unique identifier for each person

dose Dose received (e.g., 1, 2, 3)

libido Libido level of the person

Source

researchpy repo

Examples

```
data(df_difficile)
head(df_difficile)
```

`df_insurance` *Insurance Data*

Description

This data set contains information about insurance charges based on various factors such as age, sex, BMI, number of children, smoking status, and region.

Usage

```
df_insurance
```

Format

A data frame with 19 rows and 7 variables:

age Age of the individual

sex Sex of the individual (e.g., male, female)

bmi Body Mass Index of the individual

children Number of children covered by the insurance

smoker Smoking status (yes or no)

region Region where the individual resides (e.g., southwest, southeast, northwest, northeast)

charges Insurance charges

Source

researchpy repo

Examples

```
data(df_insurance)
head(df_insurance)
```

`df_ocean` *Big Five Personality Test Dataset*

Description

This dataset contains responses to an interactive online Big Five personality test conducted around 2012. Participants rated themselves on 50 personality statements, and also provided demographic and technical metadata. Responses were collected with informed consent, and missing data is coded as 0.

Usage

```
df_ocean
```

Format

A data frame with 19719 rows and 57 variables:

race Race / ethnic background (1–13, 0=missing)

age Age (integer; only responses from participants 13 and older included)

engnat Is English your native language? (1=Yes, 2=No, 0=missing)

gender 1=Male, 2=Female, 3=Other, 0=missing

hand Dominant writing hand: 1=Right, 2=Left, 3=Both, 0=missing

source How participant came to the test site: 1=Internal link, 2=Google, 3=Facebook, 4=.edu site, 6=Other / unknown

country Two-letter ISO country code (e.g., "US", "GB")

E1 I am the life of the party.

E2 I don't talk a lot.

E3 I feel comfortable around people.

E4 I keep in the background.

E5 I start conversations.

E6 I have little to say.

E7 I talk to a lot of different people at parties.

E8 I don't like to draw attention to myself.

E9 I don't mind being the center of attention.

E10 I am quiet around strangers.

N1 I get stressed out easily.

N2 I am relaxed most of the time.

N3 I worry about things.

N4 I seldom feel blue.

N5 I am easily disturbed.

N6 I get upset easily.

N7 I change my mood a lot.

N8 I have frequent mood swings.

N9 I get irritated easily.

N10 I often feel blue.

A1 I feel little concern for others.

A2 I am interested in people.

A3 I insult people.

A4 I sympathize with others' feelings.

A5 I am not interested in other people's problems.

A6 I have a soft heart.

A7 I am not really interested in others.

A8 I take time out for others.

A9 I feel others' emotions.

A10 I make people feel at ease.

C1 I am always prepared.

C2 I leave my belongings around.

C3 I pay attention to details.

C4 I make a mess of things.

C5 I get chores done right away.

C6 I often forget to put things back in their proper place.

C7 I like order.

C8 I shirk my duties.

C9 I follow a schedule.

C10 I am exacting in my work.

O1 I have a rich vocabulary.

O2 I have difficulty understanding abstract ideas.

O3 I have a vivid imagination.

O4 I am not interested in abstract ideas.

O5 I have excellent ideas.

O6 I do not have a good imagination.

O7 I am quick to understand things.

O8 I use difficult words.

O9 I spend time reflecting on things.

O10 I am full of ideas.

Details

Personality items were rated on a five-point Likert scale: #' 1=Disagree, 3=Neutral, 5=Agree. Missed=0.

race Chosen from a drop down menu. 1=Mixed Race, 2=Arctic (Siberian, Eskimo), 3=Caucasian (European), 4=Caucasian (Indian), 5=Caucasian (Middle East), 6=Caucasian (North African, Other), 7=Indigenous Australian, 8=Native American, 9=North East Asian (Mongol, Tibetan, Korean Japanese, etc), 10=Pacific (Polynesian, Micronesian, etc), 11=South East Asian (Chinese, Thai, Malay, Filipino, etc), 12=West African, Bushmen, Ethiopian, 13=Other (0=missed) **age** Entered as text (individuals reporting age < 13 were not recorded) **engnat** Response to "is English your native language?". 1=Yes, 2=No (0=missed) **gender** Chosen from a drop down menu. 1=Male, 2=Female, 3=Other (0=missed) **hand** "What hand do you use to write with?". 1=Right, 2=Left, 3=Both (0=missed)

On this page users were also asked to confirm that their answers were accurate and could be used for research. Participants who did not were not recorded). Some values were calculated from technical information.

country The participant's technical location. ISO country code. **source** How the participant came to the test. Based on HTTP Referrer. 1=from another page on the test website, 2=from google, 3=from facebook, 4=from any url with ".edu" in its domain name (e.g. xxx.edu, xxx.edu.au), 6=other source, or HTTP Referer not provided.

In psychological trait theory, the Big Five personality traits, also known as the five-factor model (FFM) and the OCEAN model, is a suggested taxonomy, or grouping, for personality traits, developed from the 1980s onwards. When factor analysis (a statistical technique) is applied to personality survey data, some words used to describe aspects of personality are often applied to the same person. For example, someone described as conscientious is more likely to be described as "always prepared" rather than "messy". This theory is based therefore on semantic associations between words and not on neuropsychological experiments. This theory uses descriptors of common language and suggests five broad dimensions commonly used to describe the human personality and psyche.

The theory identifies five factors:

- **Openness to experience (O)** (inventive / curious vs. consistent / cautious)
- **Conscientiousness (C)** (efficient / organized vs. extravagant / careless)
- **Extraversion (E)** (outgoing / energetic vs. solitary / reserved)
- **Agreeableness (A)** (friendly / compassionate vs. challenging / callous)
- **Neuroticism (N)** (sensitive / nervous vs. resilient / confident)

The five factors are represented using the acronyms OCEAN or CANOE. Beneath each proposed global factor, there are a number of correlated and more specific primary factors. For example, extroversion is typically associated with qualities such as gregariousness, assertiveness, excitement-seeking, warmth, activity, and positive emotions. Family life and the way someone was raised will affect these traits. Twin studies and other research have shown that about half of the variation between individuals results from their genetics and half from their environments. Researchers have found conscientiousness, extroversion, openness to experience, and neuroticism to be relatively stable from childhood through adulthood.

Items are grouped by Big Five traits:

- **Extraversion (E)**: E1 to E10
- **Neuroticism (N)**: N1 to N10
- **Agreeableness (A)**: A1 to A10
- **Conscientiousness (C)**: C1 to C10
- **Openness (O)**: O1 to O10

Negatively keyed items are: E2, E4, E6, E8, E10, N2, N4, A1, A3, A5, A7, C2, C4, C6, C8, O2, O4, O6. These should be reverse-coded prior to scoring.

Source

Collected via an online personality test with informed consent (~2012). Downloaded from Kaggle.com by the user Lucas Greenwell. see https://www.kaggle.com/datasets/lucasgreenwell/ocean-five-factor-personality-test-responses

Examples

```
data(df_ocean)
head(df_ocean)

# Compute Big Five average scores (after reverse scoring)
# library(dplyr)
# df_ocean <- df_ocean %>% mutate(E=rowMeans(select(., E1:E10), na.rm=TRUE))
data(df_ocean)
head(df_ocean)
```

---

df_personality *Big Five Inventory (BFI-44) Personality Dataset*

Description

Responses from 433 participants to the Big Five Inventory (BFI-44), a widely used measure of the five major dimensions of personality: Extraversion (E), Agreeableness (A), Conscientiousness (C), Neuroticism (N), and Openness to Experience (O). Items are rated on a 1-6 Likert scale. Reverse-scored items are marked with (R).

Usage

```
df_personality
```

Format

A data frame with 433 rows and 44 variables:

pers01 [E] Is talkative

pers02 [A-R] Tends to find fault with others

pers03 [C] Does a thorough job

pers04 [N] Is depressed, blue
pers05 [O] Is original, comes up with new ideas
pers06 [E-R] Is reserved
pers07 [A] Is helpful and unselfish with others
pers08 [C-R] Can be somewhat careless
pers09 [N-R] Is relaxed, handles stress well
pers10 [O] Is curious about many different things
pers11 [E] Is full of energy
pers12 [A-R] Starts quarrels with others
pers13 [C] Is a reliable worker
pers14 [N] Can be tense
pers15 [O] Is ingenious, a deep thinker
pers16 [E] Generates a lot of enthusiasm
pers17 [A] Has a forgiving nature
pers18 [C-R] Tends to be disorganized
pers19 [N] Worries a lot
pers20 [O] Has an active imagination
pers21 [E-R] Tends to be quiet
pers22 [A] Is generally trusting
pers23 [C-R] Tends to be lazy
pers24 [N-R] Is emotionally stable, not easily upset
pers25 [O] Is inventive
pers26 [E] Has an assertive personality
pers27 [A-R] Can be cold and aloof
pers28 [C] Perseveres until the task is finished
pers29 [N] Can be moody
pers30 [O] Values artistic, aesthetic experiences
pers31 [E-R] Is sometimes shy, inhibited
pers32 [A] Is considerate and kind to almost everyone
pers33 [C] Does things efficiently
pers34 [N-R] Remains calm in tense situations
pers35 [O-R] Prefers work that is routine
pers36 [E] Is outgoing, sociable
pers37 [A-R] Is sometimes rude to others
pers38 [C] Makes plans and follows through with them
pers39 [N] Gets nervous easily
pers40 [O] Likes to reflect, play with ideas
pers41 [O-R] Has few artistic interests
pers42 [A] Likes to cooperate with others
pers43 [C-R] Is easily distracted
pers44 [O] Is sophisticated in art, music, or literature

References

John, O. P., & Srivastava, S. (1999). The Big Five trait taxonomy: History, measurement, and theoretical perspectives. In L. A. Pervin & O. P. John (Eds.), *Handbook of personality: Theory and research* (2nd ed., pp. 102-138). Guilford Press.

Examples

```
data(df_personality)
head(df_personality)
```

---

df_responses *Young People Survey Responses*

Description

Survey responses from 1010 young people (15-30 years old) on music preferences, movie preferences, hobbies, phobias, health habits, personality traits, spending habits, and demographics.

Usage

```
df_responses
```

Format

A data frame with 1010 rows and 151 variables. All preference and personality items are rated on a 1-5 Likert scale (1=strongly disagree / not interested, 5=strongly agree / very interested) unless otherwise noted.

- **Participant Number**: Unique participant identifier
- **Music**: Interest in music generally
- **Slow songs or fast songs**: Preference for slow vs fast songs
- **Dance**: Interest in dance music
- **Folk**: Interest in folk music
- **Country**: Interest in country music
- **Classical music**: Interest in classical music
- **Musical**: Interest in musicals
- **Pop**: Interest in pop music
- **Rock**: Interest in rock music
- **Metal or Hardrock**: Interest in metal / hardrock
- **Punk**: Interest in punk music
- **Hiphop, Rap**: Interest in hiphop / rap
- **Reggae, Ska**: Interest in reggae / ska
- **Swing, Jazz**: Interest in swing / jazz

- **Rock n roll**: Interest in rock n roll
- **Alternative**: Interest in alternative music
- **Latino**: Interest in Latino music
- **Techno, Trance**: Interest in techno / trance
- **Opera**: Interest in opera
- **Movies**: Interest in movies generally
- **Horror**: Interest in horror films
- **Thriller**: Interest in thrillers
- **Comedy**: Interest in comedy films
- **Romantic**: Interest in romantic films
- **Sci-fi**: Interest in sci-fi films
- **War**: Interest in war films
- **Fantasy / Fairy tales**: Interest in fantasy films
- **Animated**: Interest in animated films
- **Documentary**: Interest in documentaries
- **Western**: Interest in westerns
- **Action**: Interest in action films
- **History**: Interest in history
- **Psychology**: Interest in psychology
- **Politics**: Interest in politics
- **Mathematics**: Interest in mathematics
- **Physics**: Interest in physics
- **Internet**: Interest in internet
- **PC**: Interest in computers
- **Economy Management**: Interest in economy / management
- **Biology**: Interest in biology
- **Chemistry**: Interest in chemistry
- **Reading**: Interest in reading
- **Geography**: Interest in geography
- **Foreign languages**: Interest in foreign languages
- **Medicine**: Interest in medicine
- **Law**: Interest in law
- **Cars**: Interest in cars
- **Art exhibitions**: Interest in art exhibitions
- **Religion**: Interest in religion
- **Countryside, outdoors**: Interest in countryside / outdoors
- **Dancing**: Interest in dancing

- **Musical instruments**: Interest in playing musical instruments
- **Writing**: Interest in writing
- **Passive sport**: Interest in watching sports
- **Active sport**: Interest in playing sports
- **Gardening**: Interest in gardening
- **Celebrities**: Interest in celebrities
- **Shopping**: Interest in shopping
- **Science and technology**: Interest in science and technology
- **Theatre**: Interest in theatre
- **Fun with friends**: Enjoyment of socialising with friends
- **Adrenaline sports**: Interest in adrenaline sports
- **Pets**: Interest in pets
- **Flying**: Fear of flying
- **Storm**: Fear of storms
- **Darkness**: Fear of darkness
- **Heights**: Fear of heights
- **Spiders**: Fear of spiders
- **Snakes**: Fear of snakes
- **Rats**: Fear of rats
- **Ageing**: Fear of ageing
- **Dangerous dogs**: Fear of dangerous dogs
- **Fear of public speaking**: Fear of public speaking
- **Smoking**: Smoking behaviour / attitude
- **Alcohol**: Alcohol consumption / attitude
- **Healthy eating**: Attitude toward healthy eating
- **Daily events**: Attitude toward planning daily events
- **Prioritising workload**: Ability to prioritise workload
- **Writing notes**: Habit of writing notes
- **Workaholism**: Tendency toward workaholism
- **Thinking ahead**: Tendency to think ahead
- **Final judgement**: Tendency to make final judgements
- **Reliability**: Self-rated reliability
- **Keeping promises**: Tendency to keep promises
- **Loss of interest**: Tendency to lose interest
- **Friends versus money**: Attitude toward friends vs money
- **Funniness**: Self-rated funniness
- **Fake**: Tendency to be fake

- **Criminal damage**: Attitude toward criminal damage
- **Decision making**: Decision-making style
- **Elections**: Attitude toward elections
- **Self-criticism**: Tendency toward self-criticism
- **Judgment calls**: Tendency to make judgment calls
- **Hypochondria**: Tendency toward hypochondria
- **Empathy**: Level of empathy
- **Eating to survive**: Attitude toward eating
- **Giving**: Tendency to give to others
- **Compassion to animals**: Compassion toward animals
- **Borrowed stuff**: Attitude toward returning borrowed items
- **Loneliness**: Tendency to feel lonely
- **Cheating in school**: Attitude toward cheating
- **Health**: Self-rated health
- **Changing the past**: Desire to change the past
- **God**: Belief in God
- **Dreams**: Belief in dreams
- **Charity**: Attitude toward charity
- **Number of friends**: Number of friends
- **Punctuality**: Self-rated punctuality
- **Lying**: Tendency to lie
- **Waiting**: Attitude toward waiting
- **New environment**: Comfort in new environments
- **Mood swings**: Tendency to have mood swings
- **Appearence and gestures**: Attention to appearance and gestures
- **Socializing**: Enjoyment of socializing
- **Achievements**: Drive for achievements
- **Responding to a serious letter**: Tendency to respond to serious letters
- **Children**: Attitude toward children
- **Assertiveness**: Level of assertiveness
- **Getting angry**: Tendency to get angry
- **Knowing the right people**: Value placed on knowing the right people
- **Public speaking**: Comfort with public speaking
- **Unpopularity**: Attitude toward unpopularity
- **Life struggles**: Attitude toward life struggles
- **Happiness in life**: Self-rated happiness
- **Energy levels**: Self-rated energy levels

- **Small - big dogs**: Preference for small vs big dogs
- **Personality**: Self-rated personality
- **Finding lost valuables**: Tendency to return lost valuables
- **Getting up**: Ease of getting up in the morning
- **Interests or hobbies**: Number of interests or hobbies
- **Parents' advice**: Attitude toward parents' advice
- **Questionnaires or polls**: Attitude toward questionnaires
- **Internet usage**: Hours of internet usage
- **Finances**: Self-rated financial management
- **Shopping centres**: Attitude toward shopping centres
- **Branded clothing**: Attitude toward branded clothing
- **Entertainment spending**: Spending on entertainment
- **Spending on looks**: Spending on looks
- **Spending on gadgets**: Spending on gadgets
- **Spending on healthy eating**: Spending on healthy eating
- **Age**: Age in years
- **Height**: Height in cm
- **Weight**: Weight in kg
- **Number of siblings**: Number of siblings
- **Gender**: 1=Female, 2=Male
- **Left - right handed**: 1=Right, 2=Left
- **Education**: 1=Primary, 2=Secondary, 3=College / University, 4=Masters, 5=Doctorate
- **Only child**: 1=Yes, 2=No
- **Village - town**: 1=Village, 2=City
- **House - block of flats**: 1=House, 2=Block of flats

Source

Collected from students of Statistics at FSEV UK. Available on Kaggle.

Examples

```
data(df_responses)
head(df_responses)
```

df_responses_state *Responses State Data*

Description

This data set contains simulated state information paired with participant numbers from the responses data set.

Usage

```
df_responses_state
```

Format

A data frame with 28 rows and 2 variables:

Participant Number Unique identifier for each participant

State State code where the participant resides (e.g., MI, OH, CO, CA, MA, WA)

Source

researchpy repo (simulated data, not real)

Examples

```
data(df_responses_state)
head(df_responses_state)
```

df_sexual_comp *Sexual Compatibility Data*

Description

Responses from 3,376 participants to a 10-item sexual compatibility questionnaire. The dataset is used as a teaching example in the researchpy Python statistics tutorials. The exact item wording is not publicly documented by the original source.

Usage

```
df_sexual_comp
```

Format

A data frame with 3376 rows and 13 variables. Each of Q1-Q10 is rated on a 0-4 scale; `score` is the unweighted sum (range 0-40). Note: age value 999 is a missing-data code; gender codes 0 and 3 are rare and likely represent missing or other responses.

Q1 Sexual compatibility item 1 (0-4)

Q2 Sexual compatibility item 2 (0-4)

Q3 Sexual compatibility item 3 (0-4)

Q4 Sexual compatibility item 4 (0-4)

Q5 Sexual compatibility item 5 (0-4)

Q6 Sexual compatibility item 6 (0-4)

Q7 Sexual compatibility item 7 (0-4)

Q8 Sexual compatibility item 8 (0-4)

Q9 Sexual compatibility item 9 (0-4)

Q10 Sexual compatibility item 10 (0-4)

score Total scale score (sum of Q1-Q10, range 0-40)

gender Gender (1=Male, 2=Female; 0 and 3 = missing / other)

age Age in years (999 = missing)

Source

researchpy Data-sets repository (https://github.com/researchpy/Data-sets)

Examples

```
data(df_sexual_comp)
head(df_sexual_comp)
```

---

df_titanic *Titanic Dataset*

Description

A dataset containing information about passengers on the Titanic.

Usage

```
df_titanic
```

Format

A data frame with the following variables:

PassengerId Unique identifier for each passenger

survived Survival status (0=No, 1=Yes)

pclass Passenger class (1=1st, 2=2nd, 3=3rd)

name Name of the passenger

sex Gender of the passenger

age Age of the passenger

sibsp Number of siblings / spouses aboard the Titanic

parch Number of parents / children aboard the Titanic

ticket Ticket number

fare Passenger fare

cabin Cabin number

embarked Port of embarkation (C=Cherbourg; Q=Queenstown; S=Southampton)

boat Lifeboat number

body Body number

home.dest Home destination

Examples

```
data(df_titanic)
head(df_titanic)
```

---

display_upper_lower_triangle

Combine the upper triangle of one matrix with the lower triangle of another

---

Description

Builds a matrix whose upper triangle comes from `m_upper` and whose lower triangle comes from `m_lower`, e.g. to show two correlation matrices, or correlations and p values, in a single table.

Usage

```
display_upper_lower_triangle(m_upper, m_lower, diagonal = NA)
```

Arguments

`m_upper`	A square numeric matrix (or an object that `as.matrix()` turns into one) supplying the upper triangle.
`m_lower`	A square numeric matrix of the same size supplying the lower triangle.
`diagonal`	What to place on the diagonal: `NA` (default) for `NA`, `"upper"` for the diagonal of `m_upper`, `"lower"` for the diagonal of `m_lower`, or any other value(s), either a single value or one value per diagonal element.

Value

A matrix with the dimensions and row / column names of `m_lower`. It is numeric, unless `diagonal` contains text, in which case the whole matrix becomes character.

See Also

`matrix_triangle`, `symmetric_matrix`

Examples

```
m1 <- matrix(1:9, nrow = 3, ncol = 3)
m2 <- matrix(11:19, nrow = 3, ncol = 3)
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = "upper")
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = "lower")
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = NA)
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = 1)
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = c("X1", "X2", "X3"))
display_upper_lower_triangle(m_upper = m1, m_lower = m2, diagonal = c(1, 2, 3))
display_upper_lower_triangle(m_upper = m1, m2)
```

`dotnames` *Get the names of objects passed through dots*

Description

Returns the unevaluated expressions passed in `...` as character strings. Internal helper used by `c_bind` to name the columns of its output.

Usage

```
dotnames(...)
```

Arguments

`...`	Objects whose expressions should be returned as names.

Value

A character vector with one element per argument in `...`.

Author(s)

Ananda Mahto

drop_levels *Drop unused factor levels and collapse rare levels into "Other"*

Description

Removes unused factor levels from a data frame and renames any level whose frequency is at or below a threshold to `"Other"`, then drops all unused levels.

Usage

```
drop_levels(df, factor_index = NULL, minimum_frequency = 5)
```

Arguments

`df`	A data frame containing one or more factor columns.
`factor_index`	Integer vector or `NULL`. Column indices of the factors to process. If `NULL`, all columns identified by `is.factor()` are processed. Default is `NULL`.
`minimum_frequency`	Integer. Levels with a frequency less than or equal to this value are collapsed into `"Other"`. Default is `5`.

Value

A data frame with the same structure as `df`, with rare factor levels renamed to `"Other"` and all unused levels dropped.

Examples

```
factor1 <- factor(c(rep("A", 10), rep("B", 10)), levels = c("A", "B", "C", "D"))
factor2 <- factor(c(rep("A", 10), rep("B", 10)), levels = c("A", "B", "C", "D"))
numeric1 <- c(1:20)
df <- data.frame(numeric1, factor1, factor2)
df$factor1
drop_levels(df = df, minimum_frequency = 9)
drop_levels(df = df, minimum_frequency = 10)
```

dummy_arrange *Dummy-code a multiple response vector into a binary data frame*

Description

Splits a vector of comma-separated multiple response values and returns a binary data frame where each unique response becomes a column, with `1` indicating the response was selected and `0` indicating it was not.

Usage

```
dummy_arrange(vector)
```

Arguments

vector	A character or numeric vector where each element contains one or more comma-separated response values (e.g. from a multiple choice question). Single-value responses are also accepted.

Value

A binary data frame with one row per element of `vector` and one column per unique response value, sorted alphabetically by column name. Values are `1` (selected) or `0` (not selected).

See Also

`generate_multiple_response_vector`

Examples

```
vector1 <- gsub(" ", "",
  generate_multiple_response_vector(
    responses = c("Agree", "Hi", "All"),
    responded = 1:3, length = 10
  ),
  fixed = TRUE
)
vector2 <- gsub(" ", "",
  generate_multiple_response_vector(responses = 1:4, responded = 1:4, length = 10),
  fixed = TRUE
)
vector3 <- sample(1:4, 10, replace = TRUE)
vector4 <- sample(LETTERS[1:3], 10, replace = TRUE)
dummy_arrange(vector = vector1)
dummy_arrange(vector = vector2)
dummy_arrange(vector = vector3)
dummy_arrange(vector = vector4)
```

---

environment_options *Load environment options*

Description

Load environment options

Usage

```
environment_options()
```

Examples

```
environment_options()
```

`excel_confusion_matrix` *Write matrix or dataframe to excel sheet*

Description

Useful for correlation matrices since it uses conditional formatting for matrices

Usage

```
excel_confusion_matrix(
  df,
  workbook,
  title = "Rows: Expected Collumns: Observed"
)
```

Arguments

`df`	dataframe or matrix
`workbook`	workbook
`title`	comment

Examples

```
file <- file.path(tempdir(), "excel_confusion_matrix.xlsx")
if (file.exists(file)) file.remove(file)
observed <- factor(round(rnorm(10000, m = 10, sd = 1)))
predicted <- factor(round(rnorm(10000, m = 10, sd = 1)))
confusion(observed, predicted)
cm <- confusion_matrix_percent(observed, predicted)
wb <- openxlsx::createWorkbook()
excel_confusion_matrix(cm, wb)
openxlsx::saveWorkbook(wb, file, TRUE)
```

`excel_critical_value` *Write a data frame to Excel with per-column conditional formatting thresholds*

Description

Creates a new worksheet, writes the data, and applies `excel_generic_format`. Additionally highlights cells in specified columns that meet one or two threshold conditions, making it easy to flag critical or out-of-range values.

Usage

```
excel_critical_value(
  df,
  workbook,
  sheet = "output",
  title = NULL,
  comment = NULL,
  numFmt = "#0.00",
  critical = NULL
)
```

Arguments

`df`	A data frame or matrix whose structure determines column formatting. Integer columns receive whole-number formatting; non-integer numeric columns receive the format specified by `numFmt`.
`workbook`	An openxlsx workbook object created with `openxlsx::createWorkbook()`.
`sheet`	Character. Name of the worksheet to format. Must already exist in `workbook`. Default is `"output"`.
`title`	Character or `NULL`. If provided, written as a hidden comment on cell A1. Default is `NULL`.
`comment`	A named list or `NULL`. Each name should match a column name in `df`; the value is the comment text added to that column's header cell. Names not found in `df` are silently ignored. Default is `NULL`.
`numFmt`	Character. Excel number format string applied to non-integer numeric columns. Default is `"#0.00"`.
`critical`	A named list or `NULL`. Each name must match a column in `df`. The value is either: • A single character string with an Excel expression (e.g. `"<0.05"`, `">20"`, `"=0"`). Matching cells are highlighted in red. • A character vector of length 2 with two expressions, for example `c(">20", "<11")`. The first condition highlights in red, the second in purple. `NA` cells in the target column are skipped. Default is `NULL`.

Details

Unlike excel_generic_format, this function creates the worksheet and writes the data internally — do not call addWorksheet() or writeData() beforehand. Threshold expressions follow Excel conditional formatting syntax and are applied row by row, skipping NA values.

Value

Called for its side effects. Adds a formatted worksheet to workbook; returns NULL invisibly.

See Also

excel_generic_format, excel_matrix

Examples

```
comment <- list(
  mpg = "Miles/(US) gallon",
  cyl = "Number of cylinders",
  disp = "Displacement (cu.in.)",
  hp = "Gross horsepower",
  drat = "Rear axle ratio",
  wt = "Weight (1000 lbs)",
  qsec = "1/4 mile time",
  vs = "Engine (0=V-shaped,1=straight)",
  am = "Transmission (0=automatic,1=manual)",
  gear = "Number of forward gears",
  carb = "Number of carburetors",
  extra_comment1 = "test1",
  extra_comment2 = "test2"
)
file <- file.path(tempdir(), "excel_critical_value.xlsx")
if (file.exists(file)) file.remove(file)
wb <- openxlsx::createWorkbook()
df <- generate_missing(generate_correlation_matrix())
critical <- list(X1 = "<0.05", X5 = "<0")
excel_critical_value(
  df = df, workbook = wb, sheet = "critical", comment = list(X1 = "test"),
  numFmt = "#0.00", critical = critical
)
openxlsx::saveWorkbook(wb, file, TRUE)
file <- file.path(tempdir(), "excel_critical_value_comment.xlsx")
if (file.exists(file)) file.remove(file)
wb <- openxlsx::createWorkbook()
df <- generate_missing(mtcars)
critical <- list(mpg = ">20", am = "=0")
excel_critical_value(
  df = df, workbook = wb, sheet = "critical", comment = comment,
  numFmt = "#0.00", critical = critical
)
openxlsx::saveWorkbook(wb, file, TRUE)
file <- file.path(tempdir(), "excel_critical_value_comment_min_max.xlsx")
if (file.exists(file)) file.remove(file)
```

```
wb <- openxlsx::createWorkbook()
df <- generate_missing(mtcars)
critical <- list(mpg = c(">20", "<11"), am = "=0")
excel_critical_value(
  df = df, workbook = wb, sheet = "critical", comment = comment,
  numFmt = "#0.00", critical = critical
)
openxlsx::saveWorkbook(wb, file, TRUE)
```

`excel_generic_format` *Format an Excel worksheet with styles, comments, and frozen panes*

Description

Applies consistent formatting to an existing worksheet in an openxlsx workbook. Handles header styling, cell borders, number formatting, column auto-widths, frozen panes, and optional comments on column headers.

Usage

```
excel_generic_format(
  df,
  workbook,
  sheet = "output",
  title = NULL,
  comment = NULL,
  numFmt = "#0.00"
)
```

Arguments

`df`	A data frame or matrix whose structure determines column formatting. Integer columns receive whole-number formatting; non-integer numeric columns receive the format specified by `numFmt`.
`workbook`	An openxlsx workbook object created with `openxlsx::createWorkbook()`.
`sheet`	Character. Name of the worksheet to format. Must already exist in `workbook`. Default is `"output"`.
`title`	Character or `NULL`. If provided, written as a hidden comment on cell A1. Default is `NULL`.
`comment`	A named list or `NULL`. Each name should match a column name in `df`; the value is the comment text added to that column's header cell. Names not found in `df` are silently ignored. Default is `NULL`.
`numFmt`	Character. Excel number format string applied to non-integer numeric columns. Default is `"#0.00"`.

Details

The function assumes that `openxlsx::writeData()` has already been used to write the data to the worksheet, with both `colNames = TRUE` and `rowNames = TRUE`, as it offsets column indices by 1 to account for the row name column. Formatting applied:

- Thin gray borders on all data cells
- Thin black borders on the header row and row name column
- Column widths set to auto
- First row and first column frozen
- Base font set to Liberation Sans 10pt

Value

Called for its side effects. Modifies `workbook` in place; returns `NULL` invisibly.

Examples

```
comment <- list(
  mpg = "Miles/(US) gallon",
  cyl = "Number of cylinders",
  disp = "Displacement (cu.in.)",
  hp = "Gross horsepower",
  drat = "Rear axle ratio",
  wt = "Weight (1000 lbs)",
  qsec = "1/4 mile time",
  vs = "Engine (0=V-shaped,1=straight)",
  am = "Transmission (0=automatic,1=manual)",
  gear = "Number of forward gears",
  carb = "Number of carburetors",
  extra_comment1 = "test1",
  extra_comment2 = "test2"
)
mtcor <- data.frame(cor(mtcars))
file <- file.path(tempdir(), "excel_generic.xlsx")
if (file.exists(file)) file.remove(file)
wb <- openxlsx::createWorkbook()
openxlsx::addWorksheet(wb, "sheet")
openxlsx::addWorksheet(wb, "correlation")
openxlsx::writeData(wb, sheet = "sheet", x = mtcars, colNames = TRUE, rowNames = TRUE)
openxlsx::writeData(wb, sheet = "correlation", x = mtcor, colNames = TRUE, rowNames = TRUE)
excel_generic_format(
  df = mtcars, workbook = wb, sheet = "sheet", title = "test",
  comment = comment, numFmt = "#0.00"
)
excel_generic_format(
  df = mtcor, workbook = wb, sheet = "correlation", title = "correlation",
  comment = comment, numFmt = "#0.00"
)
openxlsx::saveWorkbook(wb, file, TRUE)
```

`excel_matrix` *Write a matrix or data frame to an Excel worksheet with optional conditional formatting*

Description

Creates a new worksheet in an openxlsx workbook, writes the data, and applies formatting via `excel_generic_format`. Optionally adds a red-yellow-green colour scale for value ranges and highlights the diagonal cells in red, which is useful for correlation matrices.

Usage

```
excel_matrix(
  df,
  workbook,
  sheet = "output",
  title = NULL,
  comment = NULL,
  numFmt = "#0.00",
  conditional_formatting = FALSE,
  diagonal = FALSE,
  diagonal_length = nrow(df)
)
```

Arguments

`df`	A data frame or matrix whose structure determines column formatting. Integer columns receive whole-number formatting; non-integer numeric columns receive the format specified by `numFmt`.
`workbook`	An openxlsx workbook object created with `openxlsx::createWorkbook()`.
`sheet`	Character. Name of the worksheet to format. Must already exist in `workbook`. Default is `"output"`.
`title`	Character or `NULL`. If provided, written as a hidden comment on cell A1. Default is `NULL`.
`comment`	A named list or `NULL`. Each name should match a column name in `df`; the value is the comment text added to that column's header cell. Names not found in `df` are silently ignored. Default is `NULL`.
`numFmt`	Character. Excel number format string applied to non-integer numeric columns. Default is `"#0.00"`.
`conditional_formatting`	Logical. If `TRUE`, applies a red-yellow-green colour scale to all data cells, where low values are red, mid values yellow, and high values green. Default is `FALSE`.
`diagonal`	Logical. If `TRUE`, fills diagonal cells with a red background. Only applied when the data frame is square (`nrow == ncol`). Default is `FALSE`.
`diagonal_length`	Integer. Number of diagonal cells to highlight when `diagonal = TRUE`. Defaults to `nrow(df)`.

Details

Unlike excel_generic_format, this function creates the worksheet and writes the data internally — do not call addWorksheet() or writeData() beforehand. The diagonal highlight is skipped silently for non-square data frames.

Value

Called for its side effects. Adds a formatted worksheet to workbook; returns NULL invisibly.

See Also

excel_generic_format

Examples

```
comment <- list(
  mpg = "Miles/(US) gallon",
  cyl = "Number of cylinders",
  disp = "Displacement (cu.in.)",
  hp = "Gross horsepower",
  drat = "Rear axle ratio",
  wt = "Weight (1000 lbs)",
  qsec = "1/4 mile time",
  vs = "Engine (0=V-shaped,1=straight)",
  am = "Transmission (0=automatic,1=manual)",
  gear = "Number of forward gears",
  carb = "Number of carburetors",
  extra_comment1 = "test1",
  extra_comment2 = "test2"
)
mtcor <- data.frame(cor(mtcars))
file <- file.path(tempdir(), "excel_matrix.xlsx")
if (file.exists(file)) file.remove(file)
wb <- openxlsx::createWorkbook()
excel_matrix(mtcars, wb,
  sheet = "matrix", comment = comment,
  conditional_formatting = TRUE, diagonal = FALSE
)
excel_matrix(mtcars, wb,
  sheet = "diagonal_non_square", comment = comment,
  conditional_formatting = FALSE, diagonal = TRUE
)
excel_matrix(mtcars[1:10, 1:10], wb,
  sheet = "diagonal_square", comment = comment[1:10],
  conditional_formatting = FALSE, diagonal = TRUE
)
excel_matrix(mtcars, wb,
  sheet = "matrix_diagonal_non_square", comment = comment,
  conditional_formatting = TRUE, diagonal = TRUE
)
excel_matrix(mtcars[1:10, 1:10], wb,
  sheet = "matrix_diagonal_square", comment = comment[1:10],
```

```
  conditional_formatting = TRUE, diagonal = TRUE
)
excel_matrix(mtcor, wb,
  sheet = "r", comment = comment,
  conditional_formatting = FALSE, diagonal = FALSE
)
excel_matrix(mtcor, wb,
  sheet = "conditional_formatting_r", comment = comment,
  conditional_formatting = TRUE, diagonal = TRUE
)
openxlsx::saveWorkbook(wb, file, TRUE)
```

`extract_components` *Extract and plot variance components from a mixed model*

Description

Extracts variance components from a mixed linear model fitted with `mixlm::lm()` and computes each component's percentage contribution to total variance. Results are returned as both a summary data frame and a horizontal bar chart, making it easy to identify which sources of variance (e.g. person, item, time, interactions) dominate the measurement design.

Usage

```
extract_components(model, title = "")
```

Arguments

`model`	A mixed model object returned by `mixlm::lm()`. The model must include random effects specified with `r()` so that variance components are available via `mixlm::Anova()`.
`title`	Character string used as the plot title. Default is `""`.

Value

A named list with two elements:

components A data frame with one row per variance component containing columns `component` (effect name), `VC` (estimated variance component), and `vc_percent` (percentage of total absolute variance explained by that component).

plot A `ggplot` horizontal bar chart displaying `vc_percent` for each component, with a line and points overlaid to show the profile across components.

Examples

```
design <- expand.grid(time = 1:3, item = 1:3, person = 1:10)
design <- change_data_type(design, type = "factor")
design$response <- rowSums(change_data_type(design[, 1:2], type = "numeric")) +
  rnorm(90, 0, 0.1)
model <- mixlm::lm(response ~ r(time) * r(person) + r(item) * r(person), data = design)
extract_components(model)
```

---

extract_tirt_params *Extract Thurstonian IRT Parameters from a lavaan-Fitted Model*

Description

Extracts and aligns the core parameter blocks needed for Thurstonian IRT scoring from a fitted lavaan object (as stored in `fit_lavaan_obj$fit`). The returned pieces are aligned to the row order of `Lambda`, which is critical for correct downstream scoring.

Specifically, this function returns:

- `Lambda`: factor loading matrix.
- `theta_diag`: residual variances (diagonal of `theta`), reordered to `rownames(Lambda)` when names are available.
- `tau`: thresholds from `tau`, reordered to `rownames(Lambda)` and with row suffixes like `"|t1"` removed.
- `nu`: indicator intercepts; defaults to zero when unavailable.
- `Psi`: latent covariance matrix (`psi`).

Usage

```
extract_tirt_params(fit_lavaan_obj)
```

Arguments

`fit_lavaan_obj`	A fitted object containing a lavaan fit in `$fit`. For example, an object produced by a wrapper that stores a lavaan model under `fit_lavaan_obj$fit`.

Details

Threshold rows in lavaan are often named like `"item|t1"`. This function strips everything after `"|"` before matching thresholds to indicators.

Value

A named list with elements: `Lambda`, `theta_diag`, `tau`, `nu`, and `Psi`.

Examples

```
library(thurstonianIRT)
data("triplets")
# define the blocks of items
blocks <-
  set_block(c("i1", "i2", "i3"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, 1)) +
  set_block(c("i4", "i5", "i6"), traits = c("t1", "t2", "t3"),
            signs = c(-1, 1, 1)) +
  set_block(c("i7", "i8", "i9"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, -1)) +
  set_block(c("i10", "i11", "i12"), traits = c("t1", "t2", "t3"),
            signs = c(1, -1, 1))
# generate the data to be understood by 'thurstonianIRT'
triplets_long <- make_TIRT_data(
  data = triplets, blocks = blocks, direction = "larger",
  format = "pairwise", family = "bernoulli", range = c(0, 1)
)
# fit the data using lavaan
fit <- fit_TIRT_lavaan(triplets_long)
pars <- extract_tirt_params(fit)
```

---

fixed *Mark a pattern as a fixed string*

Description

Flags a pattern to be interpreted as a literal string rather than a regular expression. Pass the result to `str_replace`, `str_replace_all`, `str_count`, or `str_split_fixed` wherever you want exact character matching instead of regex matching.

Usage

```
fixed(pattern)
```

Arguments

`pattern`	A character string to match literally.

Value

The same character string with class `"fixed_pattern"`.

Examples

```
# Without fixed(), "." matches any character (regex)
str_replace_all("a.b.c", ".", "-")

# With fixed(), "." matches only a literal dot
str_replace_all("a.b.c", fixed("."), "-")
```

flatten_list *Flatten a two-dimensional list into a data frame*

Description

Converts a two-dimensional list to a data frame by applying `ldply` across the top-level elements.

Usage

```
flatten_list(mydata)
```

Arguments

`mydata`	A list where each element can be coerced to a data frame.

Value

A data frame combining all list elements row-wise, with an additional `.id` column containing the top-level list names.

generate_comparisons_matrix *Generate comparisons matrix*

Description

Generate comparisons matrix

Usage

```
generate_comparisons_matrix(items)
```

Arguments

`items`	number of items

Examples

```
generate_comparisons_matrix(2)
generate_comparisons_matrix(3)
generate_comparisons_matrix(4)
generate_comparisons_matrix(5)
generate_comparisons_matrix(6)
```

`generate_correlation_matrix` *Generate a data frame with a predetermined correlation structure*

Description

Simulates multivariate normal data whose columns reproduce a target correlation matrix, using Cholesky decomposition. If no matrix is supplied, a random symmetric positive-definite matrix is generated automatically.

Usage

```
generate_correlation_matrix(correlation_martix, nrows = 10)
```

Arguments

`correlation_martix`	A symmetric positive-definite matrix specifying the desired correlations between columns. Must pass Cholesky decomposition. If omitted, a random correlation matrix is generated.
`nrows`	Integer. Number of observations (rows) to generate. Default is `10`.

Details

Uses Cholesky decomposition (`chol()`) to factor the target correlation matrix, then multiplies by independent standard normal draws to produce correlated columns. The resulting correlations approximate the target matrix, with accuracy improving as `nrows` increases.

Value

A data frame with `nrows` rows and `ncol(correlation_martix)` columns of simulated numeric values.

See Also

`generate_data`, `symmetric_matrix`

Examples

```
df <- data.frame(matrix(.999, ncol = 2, nrow = 2))
correlation_martix <- as.matrix(df)
diag(correlation_martix) <- 1
df <- generate_correlation_matrix(correlation_martix, nrows = 100)
stats::cor(df)
```

`generate_data` *Generate a data frame of random numbers*

Description

Creates a data frame populated with either normally or uniformly distributed random values, useful for testing and simulation.

Usage

```
generate_data(
  nrows = 10,
  ncols = 5,
  mean = 0,
  sd = 1,
  min = 1,
  max = 5,
  type = "normal"
)
```

Arguments

`nrows`	Integer. Number of rows to generate. Default is `10`.
`ncols`	Integer. Number of columns to generate. Default is `5`.
`mean`	Numeric. Mean of the normal distribution. Only used when `type = "normal"`. Default is `0`.
`sd`	Numeric. Standard deviation of the normal distribution. Only used when `type = "normal"`. Default is `1`.
`min`	Integer. Minimum value of the uniform distribution. Only used when `type = "uniform"`. Default is `1`.
`max`	Integer. Maximum value of the uniform distribution. Only used when `type = "uniform"`. Default is `5`.
`type`	Character. Distribution to sample from. One of `"normal"` or `"uniform"`. Default is `"normal"`.

Value

A data frame with `nrows` rows and `ncols` columns of randomly generated numeric values.

Examples

```
generate_data(nrows = 10, ncols = 5, mean = 0, sd = 1, type = "normal")
generate_data(nrows = 10, ncols = 5, min = 1, max = 5, type = "uniform")
```

`generate_factor` *Generate a data frame of random factor vectors*

Description

Creates a data frame (or single factor) populated with factor values sampled from a supplied pool, either randomly or in a balanced distribution across levels.

Usage

```
generate_factor(vector = LETTERS[1:5], nrows = 2, ncols = 10, type = "random")
```

Arguments

`vector`	Character vector of factor levels to sample from. Default is `LETTERS[1:5]`.
`nrows`	Integer. Number of rows to generate. For `type = "balanced"`, `nrows` should be divisible by `length(vector)`. Default is `2`.
`ncols`	Integer. Number of columns to generate. When `ncols = 1`, a single factor vector is returned instead of a data frame. Default is `10`.
`type`	Character. Sampling method. One of: • `"random"` — each value is sampled independently with replacement. • `"balanced"` — each level appears exactly `nrows / length(vector)` times per column. Default is `"random"`.

Value

A data frame of factors with `nrows` rows and `ncols` columns, or a single factor vector when `ncols = 1`.

Examples

```
generate_factor(vector = LETTERS[1:5], ncols = 5, nrows = 10, type = "random")
generate_factor(vector = LETTERS[1:5], ncols = 5, nrows = 10, type = "balanced")
generate_factor(vector = LETTERS[1:5], ncols = 1, nrows = 10, type = "balanced")
generate_factor(vector = LETTERS[1:5], ncols = 1, nrows = 10, type = "random")
```

generate_matrix_A *Generate Matrix A*

Description

Generate Matrix A

Usage

```
generate_matrix_A(blocks = 3, items = 3)
```

Arguments

blocks	number of blocks
items	number of items per block

Examples

```
generate_matrix_A(blocks=3,items=3)
```

generate_matrix_lambda_hat *Generate matrix lambda for specified number of comparisons*

Description

Generate matrix lambda for specified number of comparisons

Usage

```
generate_matrix_lambda_hat(blocks = 3, items = 3)
```

Arguments

blocks	number of blocks
items	number of items per block

Examples

```
generate_matrix_lambda_hat(blocks=3,items=4)
```

generate_missing *Introduce missing values into a vector or data frame*

Description

Randomly replaces a fixed number of values with NA, either in a vector or across every column of a data frame independently.

Usage

```
generate_missing(df, missing = 5)
```

Arguments

df	A numeric vector or data frame. The object into which missing values are introduced.
missing	Integer. Number of values to replace with NA per vector or per column. Must not exceed the length of the vector or nrow(df). Default is 5.

Value

The input object with missing values replaced by NA. Returns the same type as the input (vector or data frame).

Examples

```
generate_missing(rnorm(10), missing = 5)
generate_missing(generate_data(nrow = 10, ncol = 2), missing = 5)
```

generate_multiple_response_vector *Generate a multiple response vector*

Description

Creates a character vector where each element contains a comma-separated string of randomly sampled categories, simulating multiple response survey data.

Usage

```
generate_multiple_response_vector(
  responses = 1:4,
  responded = 1:4,
  length = 10
)
```

Arguments

responses	Integer or character vector. The pool of unique response categories to sample from. A single number n means the categories 1:n (e.g. responses = 4 is the same as 1:4). Default is 1:4.
responded	Integer vector. Controls how many categories are selected per observation — one value is sampled from this vector at each iteration. A single number k means "up to k": each observation gets a random number of categories between 1 and k (e.g. responded = 3 gives 1, 2 or 3 categories). The values should not exceed the number of categories in responses. Default is 1:4.
length	Integer. Number of observations to generate. Default is 10.

Value

A character vector of length length, where each element is a comma-separated string of sampled response categories.

Examples

```
generate_multiple_response_vector(responses = 1:4, responded = 1:4, length = 10)
```

---

generate_string *Generate random strings*

Description

Produces a character vector of random strings by sampling from a character pool.

Usage

```
generate_string(
  vector = c(LETTERS, letters, 0:9),
  vector_length = 1,
  nchar = 5
)
```

Arguments

vector	Character vector. The pool of characters to sample from. Default is c(LETTERS, letters, 0:9).
vector_length	Integer. Number of strings to generate. Default is 1.
nchar	Integer. Length of each generated string. Default is 5.

Value

A character vector of length vector_length.

Examples

```
generate_string(nchar = 10)
generate_string(nchar = 10, vector_length = 10)
```

---

generate_unique_comparisons_index *Generate index for unique comparisons*

Description

Generate index for unique comparisons

Usage

```
generate_unique_comparisons_index(items)
```

Arguments

items	number of items

Examples

```
generate_unique_comparisons_index(1)
generate_unique_comparisons_index(2)
generate_unique_comparisons_index(3)
generate_unique_comparisons_index(4)
generate_unique_comparisons_index(5)
generate_unique_comparisons_index(6)
```

---

get_fwd *Get the file path of the currently running script*

Description

Returns the normalised absolute path of the R script that is currently executing. The function tries the following methods in order:

1. The `--file=` command-line argument (set when running via `Rscript script.R`).
2. The `fileName` variable in the first call-stack frame (set by some IDEs).
3. The `ofile` variable in the first call-stack frame (set when the script is loaded with `source()`).
4. The active document path from `rstudioapi` (RStudio only).
5. The source editor path from `rstudioapi` as a fallback.

Usage

```
get_fwd()
```

Value

A character string containing the normalised absolute path of the current script, or an empty string (`""`) if the path cannot be determined.

Examples

```
filename<-get_fwd()
filename
```

---

get_script_directory *Get script directory*

Description

Returns the directory of the currently active script as a string with a trailing slash. Works across multiple environments: RStudio, command line execution, and generic R sessions.

Usage

```
get_script_directory()
```

Details

The function tries three approaches in order:
1. If RStudio is available, uses `rstudioapi` to get the active document path
2. If running from the command line via `Rscript --file=`, parses the file argument
3. Falls back to `getwd()` as a last resort

Value

A character string with the directory path, always ending with "/"

Note

The fallback to `getwd()` may not reflect the script's actual location if the working directory has been changed during the session.

Examples

```
# Returns the directory of the active script in RStudio
directory <- get_script_directory()
directory
```

icc_cfa *Select responses for each dimension*

Description

Select responses for each dimension

Usage

```
icc_cfa(eta, gamma, lambda, psi)
```

Arguments

eta	eta or ability
gamma	gamma or threshold
lambda	lambda or loading
psi	psi or error

Examples

```
icc_cfa(seq(-6,6,.1),1,1,1)
```

increase_index *index dataframe picks*

Description

index dataframe picks

Usage

```
increase_index(blocks, items)
```

Arguments

blocks	number of blocks
items	number of items per block

Examples

```
increase_index(3,3)
```

install_all_packages *Install all missing CRAN packages*

Description

Compares the set of currently installed packages against the full list of packages available on CRAN and installs any that are missing. Already-installed packages are not re-downloaded or updated. Note that CRAN contains thousands of packages, so this function can take a very long time and requires a large amount of disk space.

Usage

```
install_all_packages()
```

Value

Invisibly returns `NULL`. Called for its side effect of installing packages.

install_load *Install and load multiple packages*

Description

Checks whether each package in `package` is already installed. Missing packages are downloaded from CRAN with `dependencies = TRUE` and then loaded. Packages that are already installed are loaded directly without reinstalling.

Usage

```
install_load(package)
```

Arguments

`package`	Character vector of package names to install (if needed) and load.

Value

A named logical vector (one element per package) indicating whether each package was successfully attached: `TRUE` if loaded, `FALSE` if loading failed.

Author(s)

Steven Worthington

Examples

```
install_load("car")
install_load(c("car", "ggplot2"))
```

k_fold *K-Fold train test sampling*

Description

Splits a dataframe into train and test dataframes for model evaluation. Prepared data include data objects for xgboost.

Usage

```
k_fold(df, model_formula, k = 10)
```

Arguments

df	Dataframe containing the dataset to be split.
model_formula	Model formula specifying the predictors and outcome variable.
k	Integer value representing the number of folds. Defaults to 10.

Details

This function performs k-fold cross-validation by splitting the input dataframe into k folds. Each fold serves as a test set once,while the remaining k-1 folds form the training set.

The function prepares data objects for xgboost model training and evaluation, including train / test datasets and xgboost DMatrix objects.

The output is a list containing the following elements: -‘f‘: List of train and test datasets for each fold. -‘index‘: Vector of fold indices. -‘model_formula‘: Model formula used for generating the datasets. -‘variables‘: Names of the variables in the model formula. -‘predictors‘: Names of the predictor variables. -‘outcome‘: Name of the outcome variable. -‘xgb‘: List of xgboost DMatrix objects for training and testing.

Examples

```
# Example with the 'infert' dataset
infert_formula<-formula(case~education+spontaneous+induced)
result<-k_fold(infert,k=10,model_formula=infert_formula)

# Example with the 'mtcars' dataset
model_formula<-as.formula(mpg~cyl+disp+hp+drat+wt+qsec+vs+am+gear+carb)
result<-k_fold(mtcars,k=2,model_formula=model_formula)
```

k_sample *Train test sampling*

Description

Splits a dataframe into train and test dataframes for model evaluation. Prepared data include data objects for xgboost.

Usage

```
k_sample(df, model_formula, k = 1)
```

Arguments

df	Dataframe containing the dataset to be split.
model_formula	Model formula specifying the predictors and outcome variable.
k	Integer value representing the number of folds. Defaults to 1 (train-test split).

Details

This function performs k-fold cross-validation or a simple train-test split (if k=1) by splitting the input dataframe into k folds. Each fold serves as a test set once,while the remaining k-1 folds form the training set.

The function prepares data objects for xgboost model training and evaluation, including train, test, and validation datasets and xgboost DMatrix objects.

The output is a list containing the following elements: -‘f‘: List of train,test,and validation datasets for each fold. -‘index‘: Vector of fold indices. -‘model_formula‘: Model formula used for generating the datasets. -‘variables‘: Names of the variables in the model formula. -‘predictors‘: Names of the predictor variables. -‘outcome‘: Name of the outcome variable. -‘xgb‘: List of xgboost DMatrix objects for training,testing,and validation.

Examples

```
# Example with the 'infert' dataset
infert_formula<-formula(case~education+spontaneous+induced)
result<-k_sample(df=infert,k=10,model_formula=infert_formula)

# Example with the 'mtcars' dataset
model_formula<-formula(mpg~cyl+disp+hp+drat+wt+qsec+vs+am+gear+carb)
result<-k_sample(df=mtcars,k=10,model_formula=model_formula)
```

key_to_cfa_model *Convert a key list to a lavaan CFA model string*

Description

Converts the named list key format used by report_alpha into a lavaan model syntax string suitable for passing directly to lavaan::cfa(). Each list element becomes one factor definition line of the form factorname =~ item1+item2+....

Usage

```
key_to_cfa_model(key)
```

Arguments

key	A named list where each name is a factor (trait) label and each element is a character vector of item column names belonging to that factor, e.g. list(f1 = c("x1","x2","x3"), f2 = c("x4","x5","x6")).

Value

A single character string containing the full lavaan model specification with one factor definition per line.

Examples

```
set.seed(12345)
population_model <- "t1=~x1+.5*x2+.5*x3
                     t2=~x4+.5*x5+.5*x6
                     t3=~x7+.5*x8+.5*x9"
model_data <- lavaan::simulateData(population_model, sample.nobs = 1000)
key <- list(f1 = paste0("x", 1:3), f2 = paste0("x", 4:6), f3 = paste0("x", 7:9))
model <- key_to_cfa_model(key)
fit <- lavaan::cfa(model, model_data)
```

matrix_triangle *Keep the lower or upper triangle of a matrix*

Description

Keeps one triangle of a matrix, including the diagonal, and fills the other triangle with the value of off_diagonal; optionally replaces the diagonal. Useful for displaying correlation or covariance matrices without repeated values.

Usage

```
matrix_triangle(m, off_diagonal = NA, diagonal = NULL, type = "lower")
```

Arguments

m	A numeric matrix, or an object that as.matrix() turns into a numeric matrix (e.g. a data frame of numbers).
off_diagonal	Value used to fill the dropped triangle. Default NA.
diagonal	Value(s) to place on the diagonal: a single value or one value per diagonal element. If NULL (default), the original diagonal is kept.
type	Character, "lower" (default) or "upper": the triangle to keep.

Value

A numeric matrix with the same dimensions and row / column names as m.

See Also

display_upper_lower_triangle, symmetric_matrix

Examples

```
m <- matrix(1:9, nrow = 3, ncol = 3)
matrix_triangle(m = m)
matrix_triangle(m = m, diagonal = NA, type = "lower")
matrix_triangle(m = m, diagonal = NULL, type = "lower")
matrix_triangle(m = m, diagonal = NA, type = "upper")
matrix_triangle(m = m, diagonal = NULL, type = "upper")
```

---

min_max_index *Indices of the minimum and maximum values in a vector*

Description

Returns the positions of the minimum and maximum values in a vector. When there are ties all tied positions are returned.

Usage

```
min_max_index(vector)
```

Arguments

vector	A numeric vector.

Value

A named list with two elements:

max_index Integer vector of positions where the maximum value occurs.

min_index Integer vector of positions where the minimum value occurs.

Examples

```
vector1 <- c(1, 2, 3, 4, 5, 4, 3, 2, 1)
vector2 <- c(1, 2, 3, 4, 5, 5, 3, 2, 1)
vector3 <- c(1, 2, 3, 5, 5, 4, 3, 2, 1)
vector4 <- c(1, 2, 3, 4, 6, 4, 3, 2, 1)
vector5 <- c(1, 6, 3, 4, 6, 4, 3, 2, 1)
vector <- vector1
which(vector == max(vector), arr.ind = TRUE)
which(vector == min(vector), arr.ind = TRUE)
min_max_index(vector1)
min_max_index(vector2)
min_max_index(vector3)
min_max_index(vector4)
min_max_index(vector5)
```

---

model_loadings *Pattern and structure matrix*

Description

Pattern and structure matrix

Usage

```
model_loadings(model, cut = NULL, matrix_type = "pattern", sort = TRUE, ...)
```

Arguments

model	psych EFA model
cut	cut point for loadings
matrix_type	"pattern" "structure" "all"
sort	if TRUE it will sort loadings
...	arguments passed to psych::fa.sort

Note

Check to see if you have multicolinearity values above .8 in the matrix are problematic
Structure matrix represents Loadings after rotation
Pattern matrix represents Loadings before rotation

Examples

```
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
model_loadings(model=model,cut=NULL,matrix_type="pattern")
model_loadings(model=model,cut=0.4,matrix_type="structure")
model_loadings(model=model,cut=0.4,matrix_type="all",sort=FALSE)
```

`name_triplet_pairs` *Create Pair Labels from Consecutive Triplets of Items*

Description

Builds pair labels from items grouped in triplets.

In simple terms: items are taken 3 at a time, and for each triplet the function creates the three pair combinations: (1,2), (1,3), and (2,3).

Labels are returned as strings such as `"i1i2"`, `"i1i3"`, `"i2i3"` (or with your chosen separator / prefix).

Usage

```
name_triplet_pairs(n, prefix = "i", sep = "", strict = TRUE)
```

Arguments

`n`	Either: • A single integer (total number of items, e.g. `15`). • A vector of item indices (e.g. `4:18`).
`prefix`	Character prefix added before each item index. Default is `"i"`.
`sep`	Character separator inserted between the two item labels. Default is `""`.
`strict`	Logical. If `TRUE` (default), stop with an error when the number of items is not a multiple of 3. If `FALSE`, silently drops leftover items so only complete triplets are used.

Details

If there are T triplets, output length is $3T$, because each triplet contributes exactly 3 pairs.

For one triplet `(a,b,c)`, the generated labels are: `ab`, `ac`, `bc` (with chosen `prefix` and `sep`).

Value

A character vector of pair labels.

Examples

```
# 15 items -> 5 triplets -> 15 pair labels
name_triplet_pairs(15)

# Custom separator
name_triplet_pairs(6, prefix = "i", sep = "_")

# Start from specific indices
name_triplet_pairs(4:9)
# triplets are (4,5,6) and (7,8,9)
```

```
# Non-multiple of 3 with strict=FALSE -> trims extras
name_triplet_pairs(10, strict = FALSE)

# Vector input with trimming when needed
name_triplet_pairs(4:18, strict = FALSE)
```

`off_diagonal_index` *Row and column indices around the diagonal of a square matrix*

Description

For each diagonal position `i` of a `length` x `length` matrix, returns the indices of the diagonal cell and of its two neighbours in the same row: the cell to the right (just above the diagonal, column `i + 1`) and the cell to the left (just below the diagonal, column `i - 1`).

Usage

```
off_diagonal_index(length)
```

Arguments

`length`	Integer (at least 1). Number of rows and columns of the square matrix.

Value

A data frame (not a matrix) with `length` rows and four numeric columns:

x1 Row index `i`.

x2 Column index of the diagonal cell (same as `x1`).

x3 Column index of the cell to the right, `i + 1`; in the last row this is `length + 1`, outside the matrix.

x4 Column index of the cell to the left, `i - 1`; in the first row this is `0`, outside the matrix.

See Also

`matrix_triangle`

Examples

```
off_diagonal_index(length = 6)
```

outlier_summary *Percentage of outliers at three z-score thresholds*

Description

Z-standardises `vector` and counts the percentage of observations whose absolute z-score exceeds 1.96, 2.58, and 3.29, corresponding approximately to the 95 %, 99 %, and 99.9 % tails of the normal distribution. Designed to be applied across columns with `sapply`.

Usage

```
outlier_summary(vector)
```

Arguments

`vector`	Numeric vector. Missing values are removed before the z-standardisation and the counts.

Value

A one-row data frame with three character columns:

abs_z_1.96 Percentage of observations with $|z| \geq 1.96$.

abs_z_2.58 Percentage of observations with $|z| \geq 2.58$.

abs_z_3.29 Percentage of observations with $|z| \geq 3.29$.

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
outlier_summary(vector)
data.frame(sapply(mtcars, outlier_summary))
```

output_compare_model_logistic *Compare logistic regression models models*

Description

Compare logistic regression models models

Usage

```
output_compare_model_logistic(model1, model2)
```

Arguments

model1	object glm model
model2	object glm model

Examples

```
modelcategoricalpredictor <- glm(case ~ education, data = infert, family = binomial)
modelcontinuouspredictor <- glm(case ~ age, data = infert, family = binomial)
modeltwopredictors <- glm(case ~ education * age, data = infert, family = binomial)
modelmultiple <- glm(case ~ education * age * parity, data = infert, family = binomial)
anova(modelcategoricalpredictor, modelcontinuouspredictor)
output_compare_model_logistic(
  model1 = modelcategoricalpredictor,
  model2 = modeltwopredictors
)
output_compare_model_logistic(
  model1 = modelcontinuouspredictor,
  model2 = modeltwopredictors
)
output_compare_model_logistic(
  model1 = modelcontinuouspredictor,
  model2 = modelcategoricalpredictor
)
```

---

output_separator *Print a formatted console output block with separators*

Description

Prints a heading, optional instructions, and optional output to the console, surrounded by # separator lines for visual clarity.

Usage

```
output_separator(
  string,
  output = NULL,
  instruction = NULL,
  length = getOption("width")/4
)
```

Arguments

string	Character. The title displayed between the main separators.
output	Object or NULL. The main content to print below the heading. If NULL, nothing is printed in its place. Default is NULL.
instruction	Character or NULL. Explanatory text printed between the heading and the output, followed by a shorter separator. Default is NULL.

length	Numeric. Width of the main separator in characters. Default is one quarter of the current console width (getOption("width") / 4).

Value

Called for its side effects. Returns NULL invisibly.

Examples

```
output_separator(string = "TEST", output = "TEST", instruction = "TEST", length = 100)
output_separator(string = "TEST", instruction = "TEST", length = 100)
output_separator(string = "TEST", output = "TEST", length = 100)
output_separator(string = "TEST")
```

---

padNA *Pad a data frame to a target number of rows with NAs*

Description

Extends a data frame to rowsneeded rows by appending (or prepending) NA-filled rows. Internal helper used by c_bind.

Usage

```
padNA(df, rowsneeded, first = TRUE)
```

Arguments

df	A data frame to pad.
rowsneeded	Integer target row count. Must be greater than or equal to nrow(df).
first	Logical. When TRUE (default) NA rows are appended at the bottom; when FALSE they are prepended at the top.

Value

A data frame with rowsneeded rows and the same columns as df.

Author(s)

Ananda Mahto

`plot_acf` *Autocorrelation, autocovariance, and partial autocorrelation plot*

Description

Produces a faceted ggplot2 chart showing the autocorrelation function (ACF), autocovariance function, and partial autocorrelation function (PACF) of a time series side by side. Each facet includes dashed 95% confidence interval lines computed from the distribution of the ACF values, making it easy to identify significant lags and seasonal patterns. Missing values are excluded via `na.action = stats::na.exclude`.

Usage

```
plot_acf(df, lag.max = length(df), base_size = 10, title = "")
```

Arguments

`df`	A `ts` object containing the time series to analyse.
`lag.max`	Integer. Maximum number of lags to compute. Default is `length(df)`, which gives all possible lags.
`base_size`	Base font size passed to `theme_bw()`. Default is `10`.
`title`	Character string used as the plot title. Default is `""`.

Value

A `ggplot` object with three free-scale facets:

Correlation Autocorrelation function — measures linear dependence between the series and its own lagged values.

Covariance Autocovariance function — the un-normalised version of the ACF.

Partial.Correlation Partial autocorrelation function — correlation at each lag after removing the effect of shorter lags, useful for identifying AR model order.

Each facet includes dashed horizontal lines for the 95% CI lower bound (blue), mean (black), and upper bound (blue).

Examples

```
ts_data <- ts(UKDriverDeaths, start = 1969, end = 1984, frequency = 12)
plot_acf(df = ts_data, base_size = 20)
```

plot_boxplot *Side-by-side boxplots for all numeric columns*

Description

Melts all numeric columns of `df` into a single long format and draws them as side-by-side horizontal boxplots on one plot. Non-numeric columns are silently dropped.

Usage

```
plot_boxplot(df, title = "", base_size = 10)
```

Arguments

`df`	Data frame or numeric vector. Non-numeric columns are silently dropped.
`title`	Character string used as the plot title. Default is `""`.
`base_size`	Base font size passed to `theme_bw()`. Default is `10`.

Value

A single `ggplot` object.

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
plot_boxplot(df = vector)
plot_boxplot(df = df)
```

plot_cfa *Batch-plot CFA across layouts and display modes*

Description

Drop-in replacement for `plot_cfa()` - same signature, returns a named list of ggplot objects instead of base-graphics recordings.

Usage

```
plot_cfa(model, ...)
```

Arguments

`model`	A fitted lavaan object
`...`	Extra arguments forwarded to `plot_cfa_gg()`

Value

Named list of ggplot objects (same keys as the original `plot_cfa`)

`plot_cfa_gg` *Plot CFA model (semPlot-free)*

Description

Plot CFA model (semPlot-free)

Usage

```
plot_cfa_gg(
  model,
  what = c("std", "est", "eq"),
  layout = c("tree", "circle", "spring"),
  label_size = 3.2,
  edge_label_size = 2.6,
  color_latent = "#4f8ef7",
  color_observed = "#e8eaf0",
  ...
)
```

Arguments

`model`	A fitted lavaan object
`what`	One of "std" (standardized), "est" (unstandardized), or "eq" (parameter labels)
`layout`	One of "tree", "circle", or "spring"
`label_size`	Size of node labels
`edge_label_size`	Size of path coefficient labels
`color_latent`	Fill colour for latent variable nodes
`color_observed`	Fill colour for observed variable nodes
`...`	Ignored (kept for API compatibility with plot_cfa)

Value

A named list of ggplot objects

Examples

```
model <- "LATENT1=~X1+X2+X3
       LATENT2=~X4+X5+X6"
df <- lavaan::simulateData(
  model = model, model.type = "cfa",
  return.type = "data.frame", sample.nobs = 100
)
df <- generate_missing(df)
fit <- lavaan::cfa(model, data = df, missing = "ML")
plot_cfa_gg(fit, what = "std")
plot_cfa_gg(fit, what = "std", layout = "tree")
plot_cfa_gg(fit, what = "std", layout = "circle")
plot_cfa_gg(fit, what = "std", layout = "spring")
```

---

plot_confusion *Plot confusion matrix*

Description

This function creates a confusion matrix plot with observed and predicted outcomes,including row and column percentages,and various accuracy metrics.

Usage

```
plot_confusion(observed, predicted, base_size = 10, title = "")
```

Arguments

observed	Vector of observed outcomes. This can be numeric or factor values representing the true class labels.
predicted	Vector of predicted outcomes. This should have the same length as the observed vector and represent the predicted class labels.
base_size	Integer value representing the base font size for the plot. Defaults to 10.
title	String representing the title of the plot. Defaults to an empty string.

Details

This function generates a confusion matrix plot using ggplot2. It provides a visual representation of the confusion matrix with observed outcomes on the x-axis and predicted outcomes on the y-axis. The cells of the matrix are filled with the count of observations and annotated with the corresponding values.

The plot also includes various accuracy metrics in the caption,such as: -Overall Accuracy: Proportion of correctly classified observations (diagonal elements). -Off-diagonal Accuracy: Proportion of misclassified observations (off-diagonal elements). -Cohen's Kappa (Unweighted,Linear,and Squared): Measures the agreement between observed and predicted outcomes.

Examples

```
# Example with numeric class labels
plot_confusion(observed=c(1,2,3,1,2,3),predicted=c(1,2,3,1,2,3))

# Example with factor class labels
observed<-c(rep("male",10),rep("female",10),"male","male")
predicted<-c(rep("male",10),rep("female",10),"female","female")
plot_confusion(observed=observed,predicted=predicted)
```

---

plot_confusion_extended

Plot a confusion matrix with the full set of derived measures

---

Description

Draws a 2x2 confusion matrix (raw counts, shaded blue by magnitude) alongside the row, column and overall measures:

Total measures Accuracy, Prevalence, Proportion Incorrectly Classified

Horizontal measures Sensitivity / Miss Rate (observed positive column), Specificity / Fall-out (observed negative column)

Vertical measures Precision / False Discovery Rate (predicted positive row), Negative Predictive Value / False Omission Rate (predicted negative row)

Usage

```
plot_confusion_extended(
  observed,
  predicted,
  positive = NULL,
  base_size = 10,
  title = ""
)
```

Arguments

`observed`	Vector of true class labels (2 unique values).
`predicted`	Vector of predicted class labels (2 unique values, same domain).
`positive`	Value treated as the positive class. Defaults to the second sorted level (matches rwf's `confusion()` convention).
`base_size`	Base font size. Default 10.
`title`	Plot title suffix.

Details

With TP, TN, FP and FN the numbers of true positives, true negatives, false positives and false negatives, and $N = TP + TN + FP + FN$ the total number of cases:

Accuracy $\frac{TP+TN}{N}$

Prevalence $\frac{TP+FN}{N}$

Proportion Incorrectly Classified $\frac{FN+FP}{N} = 1 - \text{Accuracy}$

Sensitivity (true positive rate, recall) $\frac{TP}{TP+FN}$

Miss Rate (false negative rate) $\frac{FN}{TP+FN} = 1 - \text{Sensitivity}$

Specificity (true negative rate) $\frac{TN}{FP+TN}$

Fall-out (false positive rate) $\frac{FP}{FP+TN} = 1 - \text{Specificity}$

Precision (positive predictive value) $\frac{TP}{TP+FP}$

False Discovery Rate $\frac{FP}{TP+FP} = 1 - \text{Precision}$

Negative Predictive Value $\frac{TN}{FN+TN}$

False Omission Rate $\frac{FN}{FN+TN} = 1 - \text{NPV}$

Examples

```
plot_confusion_extended(observed=c(0,0,0,1,1,1,1),
                        predicted=c(0,0,0,1,1,1,1))
plot_confusion_extended(observed=c(0,0,0,1,1,1,1),
                        predicted=c(0,0,0,1,1,1,0),
                        positive=0)
plot_confusion_extended(observed=c(0,0,0,1,1,1,1),
                        predicted=c(0,0,0,1,1,1,0),
                        positive=1)
```

---

plot_corrplot *Correlation matrix plots*

Description

Correlation matrix plots

Usage

```
plot_corrplot(mydata, title = "", base_size = 10, fill_limits = c(-1, 0, 1))
```

Arguments

mydata	correlation matrix
title	plot title
base_size	base font size
fill_limits	lower and upper limit for fill

Examples

```
plot_corrplot(stats::cor(mtcars),title="Correlation")
plot_corrplot(stats::cor(mtcars),base_size=20)
```

---

plot_crosstable *Bubble plots for pairwise cross-tabulations*

Description

Creates a bubble (point) plot for each pair of categorical variables where point size encodes cell frequency and point colour encodes the levels of the first variable. Variable pairs can be supplied explicitly via `combinations`, or generated automatically from all unique pairs within `factor_index`. A progress bar is displayed during computation.

Usage

```
plot_crosstable(
  df,
  factor_index,
  combinations = NULL,
  shape = 16,
  angle = 0,
  base_size = 10,
  title = "",
  pb = FALSE
)
```

Arguments

`df`	A data frame containing the variables to plot.
`factor_index`	Integer vector of column indices. When `combinations` is `NULL`, all unique pairwise combinations of the selected columns are plotted (self-pairs and duplicate pairs are excluded).
`combinations`	A data frame with two character columns named `index1` and `index2`, each row specifying one variable pair to plot. Takes precedence over `factor_index`.
`shape`	Integer specifying the ggplot2 point shape. Default is `16` (filled circle).
`angle`	Numeric angle (in degrees) for x-axis tick labels. Default is `0`.
`base_size`	Base font size passed to `theme_bw()`. Default is `10`.
`title`	Character string used as the plot title. Default is `""`.
`pb`	Logical; whether to display a progress bar in the console. Default is `FALSE`.

Value

A named list of `ggplot` objects, one per variable pair. Each element is named `"var1_var2"` and shows a bubble chart with cell frequency as the point size, frequency counts as text labels, and total observations in the caption. Variable pairs with zero total observations are silently dropped.

Examples

```
combinations <- data.frame(index1 = c("vs", "am"),
                           index2 = c("cyl", "cyl"))
plot_crosstable(df = mtcars, factor_index = 8:9)
plot_crosstable(df = mtcars, combinations = combinations)
plot_crosstable(df = mtcars, combinations = combinations, pb = TRUE)
```

---

plot_duplicate_y_axis *Duplicate the y axis on the right side of a ggplot*

Description

Takes two ggplot objects and renders p1 with the y axis of p2 mirrored onto the right side. Useful when overlaying two series with different scales or simply to frame the plot with matching axes on both sides.

Usage

```
plot_duplicate_y_axis(p1, p2)
```

Arguments

p1	A ggplot object. This plot is drawn with the duplicated right axis.
p2	A ggplot object whose left y axis is mirrored to the right of p1. Typically the same as p1.

Value

Invisibly returns NULL. The combined plot is drawn to the current graphics device.

Examples

```
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet, group = Chick)) +
  geom_line() +
  ggtitle("Growth curve for individual chicks")
plot_duplicate_y_axis(p1 = p1, p2 = p1)
```

plot_hinvert_title_grob *Invert a title grob horizontally*

Description

Helper used by `duplicate_y_axis` to mirror a grob's width layout and text alignment so it renders correctly on the right-hand side of a plot.

Usage

```
plot_hinvert_title_grob(grob)
```

Arguments

`grob`	A grob object (titleGrob or similar) to invert.

Value

The modified grob.

plot_histogram *Histograms per numeric column*

Description

Produces one `ggplot` histogram per numeric column of `df`. Each plot caption shows the observation count, mean, SD, and median of the column. A progress bar is printed to the console.

Usage

```
plot_histogram(
  df,
  bins = 30,
  title = "",
  base_size = 10,
  xlims = NULL,
  fill = "gray25",
  color = "gray50",
  ylab = "Count",
  pb = FALSE
)
```

Arguments

df	Data frame or numeric vector. Non-numeric columns are silently dropped.
bins	Number of histogram bins passed to ggplot2::geom_histogram(). Default is 30.
title	Character string used as the plot title. Default is "".
base_size	Base font size passed to theme_bw(). Default is 10.
xlims	Length-2 numeric vector setting the x-axis limits, e.g. c(0, 50). When NULL (default) limits are determined automatically.
fill	Fill colour of the histogram bars. Default is "gray25".
color	Outline colour of the histogram bars. Default is "gray50".
ylab	Y-axis label. Default is "Count".
pb	Logical; whether to display a progress bar in the console. Default is FALSE.

Value

A named list of ggplot objects, one per numeric column.

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
plot_histogram(df = vector)
plot_histogram(df = df, xlims = c(0, 50))
plot_histogram(df = df)
plot_multiplot(plotlist = plot_histogram(df = mtcars[,1:2]), cols = 2)
```

---

plot_icc_thurstonian *Plot thurstonian icc*

Description

Plot icc curves, computed with the compute_icc_thurstonian function, for binary thurstonian coded items for a single dimension

Usage

```
plot_icc_thurstonian(mydata, title = "Item Characteristic Curve")
```

Arguments

mydata	dataframe from compute_icc_thurstonian function
title	plot title

Examples

```
gamma<-c(0.556,-1.253,-1.729,0.618,0.937,0.295,-0.672,-1.127,-0.446,0.632,1.147,0.498)
psi<-c(2.172,1.883,2.055,1.869,2.231,2.100,1.762,1.803,1.565,1.892,1.794,1.686)
lambda<-c(1.082,1.082,-1.297,-1.297,0.802,0.802,1.083,1.083)
gamma<-gamma[response_dimension(c(1:12),3,c(1,2))]
psi<-psi[response_dimension(c(1:12),3,c(1,2))]
eta<-seq(-6,6,by=1)
result<-compute_icc_thurstonian(eta=eta,gamma=gamma,lambda=lambda,psi=psi,plot=TRUE)
plot_icc_thurstonian(result$icc)
```

---

plot_interaction *Plot two-way interaction graphs for all IV pair and DV combinations*

Description

For every unique pair of independent variables (IV1 x IV2) and every dependent variable (DV), produces a line-and-point interaction plot with group means on the y-axis. IV1 levels appear on the x-axis (flipped) and IV2 levels are represented by colour and line group. Optional error bars and per-group sample size annotations are included. When the number of combinations exceeds four times the available CPU cores the plots are produced in parallel via `future.apply`, otherwise sequentially.

Usage

```
plot_interaction(
  df,
  dv,
  iv,
  base_size = 20,
  type = "se",
  order_factor = TRUE,
  title = "",
  note = ""
)
```

Arguments

`df`	A data frame containing both the independent and dependent variables.
`dv`	Integer vector of column indices for the continuous dependent variables.
`iv`	Integer vector of column indices for the categorical independent variables. The columns are coerced to factors automatically.
`base_size`	Base font size in pt passed to `theme_bw`. Default `20`.
`type`	Type of error bar to display. One of `"se"` (standard error), `"ci"` (95% confidence interval), `"sd"` (standard deviation), or `""` (no error bars). Default `"se"`.
`order_factor`	Logical. If `TRUE` factor levels on the x-axis are sorted by the group mean of the DV (descending). Default `TRUE`.

title	Character. Plot title applied to every panel. Default "".
note	Character. Caption / footnote appended to every panel. Default "".

Value

A named list with three elements:

- plot_data — named list of summary data frames (one per IV1-IV2-DV combination) as returned by Rmisc::summarySE.
- plot_data_df — single data frame combining all summary data frames row-wise.
- plots — named list of ggplot objects (one per combination).

All list elements are named "iv1_iv2_dv".

Examples

```
# Single DV, two IVs
plot_interaction(df = mtcars, dv = 2, iv = 8:9, base_size = 20, type = "se")

# Multiple DVs, two IVs
plot_interaction(df = mtcars, dv = 2:3, iv = 8:9, base_size = 20, type = "se")

# No error bars, unordered factor axis
plot_interaction(df = mtcars, dv = 2, iv = 9:10, base_size = 20, type = "",
                 order_factor = FALSE)
```

---

plot_irt_onefactor *Return data for irt plots*

Description

Return data for irt plots

Usage

```
plot_irt_onefactor(model, theta = seq(-6, 6, 0.1), title = "", base_size = 10)
```

Arguments

model	object mirt
theta	theta
title	plot title
base_size	base size

Examples

```
cormatrix<-psych::sim.rasch(nvar=5,n=50000,low=-4,high=4,d=NULL,a=1,mu=0,sd=1)$items
model<-mirt::mirt(cormatrix,1,empiricalhist=TRUE,calcNull=TRUE)
plot_irt_onefactor(model=model,base_size=10,title="Normal Test")
cormatrix<-psych::sim.rasch(nvar=5,n=50000,low=-6,high=-4,d=NULL,a=1,mu=0,sd=1)$items
model<-mirt::mirt(cormatrix,1,empiricalhist=TRUE,calcNull=TRUE)
plot_irt_onefactor(model=model,base_size=10,title="Easy Items")
cormatrix<-psych::sim.rasch(nvar=5,n=50000,low=4,high=6,d=NULL,a=1,mu=0,sd=1)$items
model<-mirt::mirt(cormatrix,1,empiricalhist=TRUE,calcNull=TRUE)
plot_irt_onefactor(model=model,base_size=10,title="Difficult Items")
cormatrix<-psych::sim.rasch(nvar=5,n=50000,low=-4,high=-4,d=NULL,a=0.01,
                            mu=0,sd=1)$items
model<-mirt::mirt(cormatrix,1,empiricalhist=TRUE,calcNull=TRUE)
plot_irt_onefactor(model=model,base_size=10,title="Low Discrimination")
cormatrix<-psych::sim.poly(nvar=5,n=50000,low=-4,high=4,a=1,c=0,z=1,d=NULL,
                           mu=0,sd=1,cat=5,mod="logistic",theta=NULL)$items
model<-mirt::mirt(cormatrix,1,itemtype="graded")
plot_irt_onefactor(model=model,base_size=10,title="graded response")
```

`plot_loadings` *Plot loadings*

Description

Plot loadings

Usage

```
plot_loadings(
  model,
  matrix_type = NULL,
  title = "",
  base_size = 10,
  color = c("#5E912C", "white", "#5F2C91"),
  sort = TRUE
)
```

Arguments

`model`	psych EFA model
`matrix_type`	"pattern" "structure"
`title`	plot title
`base_size`	base font size
`color`	color ranges for heatmap
`sort`	TRUE or FALSE sort loadings

Examples

```
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
plot_loadings(model=model,matrix_type="structure")
plot_loadings(model=model,matrix_type="pattern")
cm<-matrix(c(1,.8,.8,.1,.1,.1,
             .8,1,.8,.1,.1,.1,
             .8,.8,1,.1,.1,.1,
             .1,.1,.1,1,.8,.8,
             .1,.1,.1,.8,1,.8,
             .1,.1,.1,.8,.8,1),
             ncol=6,nrow=6)
df1<-generate_correlation_matrix(cm,nrows=10000)
model1<-psych::fa(df1,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
plot_loadings(model=model1,matrix_type="pattern",base_size=30)
cm<-matrix(c(1,.1,.1,.1,.1,.1,
             .1,1,.1,.1,.1,.1,
             .1,.1,1,.1,.1,.1,
             .1,.1,.1,1,.8,.8,
             .1,.1,.1,.8,1,.8,
             .1,.1,.1,.8,.8,1),
             ncol=6,nrow=6)
df1<-generate_correlation_matrix(cm,nrows=10000)
model2<-psych::fa(df1,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
plot_loadings(model=model2,matrix_type="pattern",base_size=30)
cm<-matrix(c(1,.01,.01,.01,.01,.01,
             .01,1,.01,.01,.01,.01,
             .01,.01,1,.01,.01,.01,
             .01,.01,.01,1,.01,.01,
             .01,.01,.01,.01,1,.01,
             .01,.01,.01,.01,.01,1),
             ncol=6,nrow=6)
df1<-generate_correlation_matrix(cm,nrows=10000)
model3<-psych::fa(df1,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
plot_loadings(model=model3,matrix_type="pattern",base_size=10)
```

---

plot_logistic_model *Logistic model plot*

Description

Logistic model plot

Usage

```
plot_logistic_model(df, outcome = "outcome", title = "", base_size = 10)
```

Arguments

df	dataframe with predictor and outcome outcome should be last

outcome	name of outcome variable
title	Character plot title
base_size	base font size

Examples

```
df <- data.frame(
  outcome = c(rep(1, 10), rep(0, 10)),
  pd1 = c(rep(1, 11), rep(0, 9)),
  pd2 = c(rep(1, 9), rep(0, 11)),
  pc1 = c(rnorm(10, mean = 5), rnorm(10, mean = 10)),
  pc2 = c(rnorm(10, mean = 5), rnorm(10, mean = 20))
)
plot_logistic_model(df = df, base_size = 15)
```

---

plot_mosaic *Mosaic plots for pairwise categorical variables*

Description

Creates a mosaic plot for every ordered pair of categorical variables within `factor_index`. In each plot, bar widths represent the marginal proportion of the first variable and bar heights represent the conditional proportion of the second variable given the first, making it straightforward to assess both marginal distributions and conditional relationships simultaneously. Rows with missing values are excluded pair-wise. A progress bar is displayed during computation.

Usage

```
plot_mosaic(df, factor_index, base_size = 10, title = "", pb = FALSE)
```

Arguments

df	A data frame containing the variables to plot.
factor_index	Integer vector of column indices identifying the categorical variables. All ordered pairs of distinct columns are plotted.
base_size	Base font size passed to `theme_bw()`. Default is `10`.
title	Character string prepended to each plot title. Default is `""`.
pb	Logical; whether to display a progress bar in the console. Default is `FALSE`.

Value

A named list of `ggplot` objects, one per ordered variable pair. Each element is named `"var1 var2"` and shows a mosaic chart with bar widths proportional to the marginal distribution of `var1`, bar heights proportional to the conditional distribution of `var2` given `var1`, and total complete-case observations in the caption. Variables with fewer than two observed levels are handled gracefully by adding a placeholder level.

Examples

```
plot_mosaic(df = mtcars, factor_index = 8:9)
plot_mosaic(df = mtcars, factor_index = 9:10)
plot_mosaic(df = mtcars, factor_index = 9:10, pb = TRUE)
```

`plot_mtmm` *Multitrait-multimethod (MTMM) matrix plot*

Description

Visualises a Campbell-Fiske multitrait-multimethod matrix as a faceted tile plot. For each trait-method combination the function computes a scale score (row mean of items), calculates Cronbach's alpha (shown on the diagonal), and correlates all scale scores across traits and methods. Each cell is colour-coded by its MTMM classification, making it easy to evaluate convergent and discriminant validity at a glance.

Usage

```
plot_mtmm(df, key, method, subject, title = "")
```

Arguments

`df`	A data frame in long format where each row corresponds to one subject under one method. Item response columns must be present for all traits and methods.
`key`	A named list mapping trait names to character vectors of item column names, e.g. `list(t1 = c("x1","x2","x3"), t2 = c("x4","x5","x6"))`.
`method`	Character string giving the name of the column that identifies the measurement method for each row.
`subject`	Character string giving the name of the column that identifies the subject (used to align scores across methods).
`title`	Character string appended to the plot title. Default is `""`.

Value

A `ggplot` object showing a faceted tile plot where rows and columns correspond to traits, facets correspond to method pairs, cell values are correlations (or alpha on the diagonal), and fill colour encodes one of four MTMM relationship types:

monotrait-monomethod (reliability) Same trait, same method — Cronbach's alpha.

monotrait-heteromethod (validity) Same trait, different methods — convergent validity.

heterotrait-monomethod Different traits, same method — discriminant validity within method.

heterotrait-heteromethod Different traits, different methods — discriminant validity across methods.

Examples

```
set.seed(12345)
population_model <- "t1=~x1+.9*x2+.9*x3
                     t2=~x4+.9*x5+.9*x6
                     t3=~x7+.9*x8+.9*x9"
model_data <- lavaan::simulateData(population_model, sample.nobs = 1000)
model_data <- model_data[sample(1:1000, 1000, TRUE), ]
model_data <- rbind(model_data, model_data, model_data)
model_data$method <- c(rep("m1", 1000), rep("m2", 1000), rep("m3", 1000))
model_data$id <- rep(1:1000, 3)
key <- list(t1 = paste0("x", 1:3), t2 = paste0("x", 4:6), t3 = paste0("x", 7:9))
plot_mtmm(df = model_data, key = key, method = "method", subject = "id")
```

---

plot_multiplot *Arrange multiple ggplot objects in a grid layout*

Description

Combines multiple ggplot objects into a single paged display using a grid layout. Plots are arranged by column across one or more pages, with each page recorded and returned as a list.

Usage

```
plot_multiplot(..., plotlist = NULL, cols = 2, layout = NULL)
```

Arguments

`...`	ggplot objects passed directly.
`plotlist`	A list of ggplot objects. Combined with any plots passed via `...`.
`cols`	Integer. Number of columns in the layout grid. Ignored if `layout` is provided. Default is `2`.
`layout`	A matrix specifying plot positions. Each cell contains the index of the plot to display at that position. If `NULL`, a layout is generated automatically from `cols`. Default is `NULL`.

Value

If a single plot is provided, returns it directly. Otherwise returns a list of plots recorded with `grDevices::recordPlot()`, one per page.

Examples

```
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet, group = Chick)) +
  geom_line() +
  ggtitle("Growth curve for individual chicks") +
  theme_bw()
p2 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet)) +
```

```
  geom_point(alpha = .3) +
  geom_smooth(alpha = .2, size = 1, method = "loess", formula = "y~x") +
  ggtitle("Fitted growth curve per diet") +
  theme_bw()
p3 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, colour = Diet)) +
  geom_density() +
  ggtitle("Final weight, by diet") +
  theme_bw()
p4 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, fill = Diet)) +
  geom_histogram(colour = "black", binwidth = 50) +
  facet_grid(Diet ~ .) +
  ggtitle("Final weight, by diet") +
  theme_bw()
cars_plot <- plot_histogram(mtcars[1:2])
plot_multiplot(p1, p2, p3, p4, cols = 2)
plot_multiplot(plotlist = plot_histogram(mtcars[, 1:2]), cols = 2)
plot_multiplot(plotlist = plot_histogram(mtcars[, 1:4]),
               layout = matrix(1:4, ncol = 2, byrow = TRUE))
plot_multiplot(plotlist = cars_plot, layout = matrix(1:2, ncol = 2, byrow = TRUE))
plot_multiplot(plotlist = cars_plot, cols = 3)
```

`plot_normality_diagnostics` *Normality diagnostic plots (histogram, density, boxplot, Q-Q)*

Description

For each numeric column of `df`, produces a 2×2 panel of base-graphics normality diagnostics: histogram, density curve, boxplot, and Q-Q plot with a reference line. When `file` is supplied the panels are also written to a PDF.

Usage

```
plot_normality_diagnostics(
  df,
  breaks = NULL,
  title = "",
  file = NULL,
  w = 10,
  h = 10,
  pb = FALSE
)
```

Arguments

`df`	Data frame or numeric vector. Non-numeric columns are silently dropped. Columns with fewer than three non-missing values or zero variance are skipped.

breaks	Histogram breaks passed to graphics::hist(). This may be a method name ("Sturges", "Scott", "FD") or a positive integer specifying the number of bins. Default is "Sturges".
title	Character string used as the outer plot title and as the PDF title. Default is "".
file	Character string naming the output PDF file (without extension). When NULL (default) no PDF is written.
w	Width of the PDF in inches. Default is 10.
h	Height of the PDF in inches. Default is 10.
pb	Logical; whether to display a progress bar in the console. Default is FALSE.

Value

A named list of recorded plots (one element per numeric column), returned invisibly. Each element is a plot recorded with grDevices::recordPlot().

Examples

```
file = file.path(tempdir(), "rnorm")
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
plot_normality_diagnostics(df = vector, file = file, breaks = 30)
plot_normality_diagnostics(df = vector)
plot_normality_diagnostics(df = df, title = "mtcars")
plot_normality_diagnostics(df = df, title = "mtcars", pb = TRUE)
plot_normality_diagnostics(df = df, title = "mtcars")
```

---

plot_oneway *Plot group means with error bars for all IV-DV combinations*

Description

For every combination of independent variable (IV) and dependent variable (DV) supplied, produces a horizontal dot plot of group means with optional error bars (standard error, confidence interval, or standard deviation). Sample size per group is annotated on each panel. When the number of IV-DV combinations exceeds four times the available CPU cores the plots are produced in parallel via future.apply, otherwise sequentially.

Usage

```
plot_oneway(
  df,
  dv,
  iv,
  base_size = 20,
  type = "se",
  order_factor = TRUE,
```

```
  title = "",
  note = "",
  width = 60
)
```

Arguments

df	A data frame containing both the independent and dependent variables.
dv	Integer vector of column indices for the continuous dependent variables.
iv	Integer vector of column indices for the categorical independent variables. The columns are coerced to factors automatically.
base_size	Base font size in pt passed to theme_bw. Default 20.
type	Type of error bar to display. One of "se" (standard error), "ci" (95% confidence interval), "sd" (standard deviation), or "" (no error bars). Default "se".
order_factor	Logical. If TRUE factor levels on the x-axis are sorted by the group mean of the DV (descending). Default TRUE.
title	Character. Plot title applied to every panel. Default "".
note	Character. Caption / footnote appended to every panel. Default "".
width	Integer. Character width at which long axis labels are wrapped. Default 60.

Value

A named list with three elements:

- plot_data — named list of summary data frames as returned by Rmisc::summarySE (one per IV-DV pair).
- plot_data_df — single data frame combining all summary data frames row-wise.
- plots — named list of ggplot objects (one per IV-DV pair).

All list elements are named "iv_dv".

Examples

```
# Single IV, single DV
plot_oneway(df = mtcars, dv = 2, iv = 9)

# Multiple IVs and DVs
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10)

# Error bar types
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "se")
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "ci")
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "sd")
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "")

# Factor ordering
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "", order_factor = FALSE)
plot_oneway(df = mtcars, dv = 2:3, iv = 9:10, type = "", order_factor = TRUE)
```

`plot_oneway_diagnostics` *Diagnostic plots for one-way ANOVA models*

Description

For every combination of independent variable (IV) and dependent variable (DV), fits a linear model and produces a 6-panel diagnostic plot via `ggfortify::autoplot`: Residuals vs Fitted, Normal Q-Q, Scale-Location, Cook's Distance, Residuals vs Leverage, and Cook's Distance vs Leverage. **Interpretation:**

- *Residuals vs Fitted* — points should be randomly scattered with no pattern; a funnel shape indicates heteroscedasticity.
- *Normal Q-Q* — points should follow the diagonal; large deviations indicate non-normality.

When the number of IV-DV combinations exceeds four times the available CPU cores the plots are produced in parallel via `future.apply`, otherwise sequentially.

Usage

```
plot_oneway_diagnostics(df, dv, iv, base_size = 10)
```

Arguments

`df`	A data frame containing both the independent and dependent variables.
`dv`	Integer vector of column indices for the continuous dependent variables.
`iv`	Integer vector of column indices for the categorical independent variables.
`base_size`	Base font size in pt passed to `theme_bw`. Default `10`.

Value

A named list of ggplot objects (one 6-panel plot per IV-DV pair), named `"iv_dv"`.

Examples

```
# Single DV, multiple IVs
plot_oneway_diagnostics(df = mtcars, dv = 1, iv = 9:10)
# Multiple DVs and IVs
plot_oneway_diagnostics(df = mtcars, dv = 1:2, iv = 9:10)
```

plot_outlier *Dot plot of outliers by detection method*

Description

For each numeric column of `df`, draws a dot plot with observations coloured by outlier status and row-name labels repelled away from flagged points. Three outlier-detection rules are available via `method`: mean ± 2 SD, median ± 2 MAD (rescaled), or boxplot IQR fences. Reference lines for the centre and the upper / lower bounds are overlaid on each plot.

Usage

```
plot_outlier(df, method = "mean", title = "", base_size = 10, pb = FALSE)
```

Arguments

`df`	Data frame or numeric vector. Non-numeric columns are silently dropped.
`method`	Character string selecting the outlier-detection rule: `"mean"` Flags observations more than 2 standard deviations from the mean. `"median"` Flags observations more than 2 rescaled MADs ($2 \times \mathrm{MAD}/0.6745$) from the median. `"boxplot"` Flags observations outside $Q1 - 1.5 \times IQR$ or $Q3 + 1.5 \times IQR$. Default is `"mean"`.
`title`	Character string used as the plot title. Default is `""`.
`base_size`	Base font size passed to `theme_bw()`. Default is `10`.
`pb`	Logical; whether to display a progress bar in the console. Default is `FALSE`.

Value

A named list of `ggplot` objects, one per numeric column.

Author(s)

unknown

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
plot_outlier(df = vector, method = "mean", title = "random vector")
plot_outlier(df = vector, method = "median")
plot_outlier(df = vector, method = "boxplot")
plot_outlier(df = df, method = "mean", title = "random vector")
plot_outlier(df = df, method = "median")
plot_outlier(df = df, method = "boxplot")
plot_multiplot(plotlist = plot_outlier(df = mtcars[1:5, 2:3], method = "mean"), cols = 2)
```

plot_qq *Q-Q plots against the normal distribution*

Description

Produces one quantile-quantile plot per numeric column of `df`, comparing the empirical distribution to the theoretical normal. A reference line is fitted through the 25th and 75th percentiles (the same convention used by `stats::qqline()`). Non-numeric columns are skipped silently. A progress bar is printed to the console.

Usage

```
plot_qq(df, title = "", base_size = 10, pb = FALSE)
```

Arguments

`df`	Data frame or vector. Non-numeric columns are skipped.
`title`	Character string used as the plot title. Default is `""`.
`base_size`	Base font size passed to `theme_bw()`. Default is `10`.
`pb`	Logical; whether to display a progress bar in the console. Default is `FALSE`.

Value

A named list of `ggplot` objects, one per numeric column.

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
plot_qq(df = vector)
plot_qq(df = df)
plot_multiplot(plotlist = plot_qq(df = mtcars[,1:2]), cols = 2)
```

plot_response_frequencies
Horizontal bar charts of response frequencies

Description

Creates one horizontal bar chart per variable showing the frequency count of each observed level. Missing values are excluded before tabulation. Variables with no valid observations are silently dropped from the output.

Usage

```
plot_response_frequencies(
  df,
  factor_index,
  base_size = 10,
  title = "",
  width = 100,
  reorder = FALSE
)
```

Arguments

df	A data frame containing the variables to plot.
factor_index	Integer vector of column indices identifying the variables to plot.
base_size	Base font size passed to theme_bw(). Default is 10.
title	Character string prepended to each plot title. Default is "".
width	Integer controlling the character wrap width applied to the variable name in the plot title. Default is 100.
reorder	Logical. When TRUE bars are ordered by frequency in ascending order (longest bar at the top). When FALSE (default) the original level order is preserved.

Value

A named list of ggplot objects, one per variable, named by the column name. Each plot is a horizontal bar chart with counts on the x-axis and total observations shown in the caption.

Examples

```
df_ocean_N<-df_ocean[,grep("N",names(df_ocean))]
plot_response_frequencies(df = df_ocean_N)
plot_response_frequencies(df = df_ocean_N, factor_index = 1)
```

---

plot_roc *Plot Receiver Operating Characteristic (ROC) curve*

Description

Generates a ROC curve from observed outcomes and predicted probabilities.

Usage

```
plot_roc(observed, predicted, base_size = 10, title = "")
```

Arguments

observed	Vector of observed outcomes. These are the true class labels.
predicted	Vector of predicted outcome probabilities. These are the predicted probabilities for the positive class.
base_size	Integer value representing the base font size for the plot. Defaults to 10.
title	String representing the title of the plot. Defaults to an empty string.

Details

This function generates a ROC curve to evaluate the performance of a binary classification model. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

The function performs the following steps: 1. Computes the ROC curve and its confidence interval using 'pROC::roc'. 2. Generates ROC plots for both reversed and non-reversed order of class levels. 3. Creates a list of ROC plots,each with an AUC value,control level,and direction.

The output is a list of ggplot objects representing the ROC curves for different class level orders.

Examples

```
# Example with random observed and predicted values
observed<-round(abs(rnorm(100,m=0,sd=0.5)))
predicted<-abs(rnorm(100,m=0,sd=0.5))
plot_roc(observed=observed,predicted=predicted)

# Example with generated correlation matrix
df1<-data.frame(matrix(0.999,ncol=2,nrow=2))
correlation_matrix<-as.matrix(df1)
diag(correlation_matrix)<-1
df1<-generate_correlation_matrix(correlation_matrix,nrows=1000)
df1$X1<-ifelse(abs(df1$X1) < 1,0,1)
df1$X2<-abs(df1$X2)
df1$X2<-(df1$X2-min(df1$X2))/(max(df1$X2)-min(df1$X2))
plot_roc(observed=round(abs(df1$X1),0),predicted=abs(df1$X2))
```

---

plot_scatterplot *Scatter plots for all variable pairs in a data frame*

Description

Generates a scatter plot with a smoothing line and marginal histograms for every pair of numeric columns in `df`. Each plot includes Pearson r, explained variance, the regression equation, and the regression angle in its caption (when the default `y ~ x` formula is used).

When the number of pairs exceeds four times the available CPU cores the plots are produced in parallel via `future.apply`, otherwise sequentially.

Usage

```
plot_scatterplot(
  df,
  method = lm,
  formula = y ~ x,
  base_size = 10,
  coord_equal = FALSE,
  all_orders = FALSE,
  title = "",
  combinations = NULL,
  str_aes = TRUE
)
```

Arguments

df	A data frame of numeric variables. When df has exactly two columns the first is treated as the predictor and the second as the outcome.
method	Smoothing method passed to geom_smooth. It accepts "lm", "glm", "gam", "loess", or a function such as MASS::rlm. Default lm.
formula	Formula passed to geom_smooth. Default y ~ x. When a non-default formula is supplied the caption shows only pairwise n.
base_size	Base font size in pt passed to theme_bw. Default 10.
coord_equal	Logical. If TRUE both axes share the same scale and limits. Default FALSE.
all_orders	Logical. If TRUE both (X, Y) and (Y, X) orderings are plotted for each pair. Default FALSE.
title	Character. Plot title applied to every panel. Default "".
combinations	A two-column data frame specifying which pairs to plot. Column 1 is the x-variable name, column 2 is the y-variable name. When NULL (default) all pairs are derived automatically from df.
str_aes	Logical. If TRUE variable names are passed through str_aes() to clean axis labels. Default TRUE.

Value

A named list of ggplot objects, one per variable pair. Names follow the pattern "x_y".

Examples

```
result <- plot_scatterplot(df = mtcars, title = "", coord_equal = TRUE, base_size = 10)
plot_multiplot(plotlist = result[1:12], cols = 4)

# Two-column data frame: first column = predictor, second = outcome
plot_scatterplot(df = mtcars[, 1:2], base_size = 10, coord_equal = TRUE,
                 all_orders = FALSE)

# Custom variable pairs
plot_scatterplot(
```

```
  df = mtcars, base_size = 10, coord_equal = TRUE,
  combinations = data.frame(
    x = c("mpg", "mpg", "mpg"),
    y = c("cyl", "hp", "disp")
  )
)

# Simulated near-perfect correlation
x <- rnorm(1000)
y <- x + rnorm(x, sd = .1)
plot_scatterplot(df = data.frame(x, y), title = "Random Simulation", coord_equal = TRUE)
```

`plot_scree` *Scree plot displaying the Kaiser and Jolife criteria for factor extraction*

Description

Scree plot displaying the Kaiser and Jolife criteria for factor extraction

Usage

```
plot_scree(df, base_size = 15, title = "", color = c("#5F2C91", "#5E912C"))
```

Arguments

`df`	dataframe
`base_size`	base font size
`title`	plot title
`color`	color of line and point outline

Examples

```
plot_scree(df=mtcars,title="",base_size=15)
```

`plot_separability` *Plot separability*

Description

This function creates a separability plot showing the density distribution of predicted probabilities for different observed categories.

Usage

```
plot_separability(observed, predicted, base_size = 10, title = "")
```

Arguments

observed	Vector of observed outcomes. This can be numeric or factor values representing the true class labels.
predicted	Vector of predicted outcome probabilities. This should have the same length as the observed vector and represent the predicted probabilities.
base_size	Integer value representing the base font size for the plot. Defaults to 10.
title	String representing the title of the plot. Defaults to an empty string.

Details

This function generates a separability plot using ggplot2. It shows the density distribution of predicted probabilities for different observed categories. The plot helps to visualize how well the predicted probabilities separate the different observed categories.

The plot includes the following components: -Density curves for each observed category, representing the distribution of predicted probabilities. -A legend indicating the observed categories. -The total number of observations is included in the plot caption.

Examples

```
# Example with numeric class labels
df1<-data.frame(matrix(.999,ncol=2,nrow=2))
correlation_matrix<-as.matrix(df1)
diag(correlation_matrix)<-1
df1<-generate_correlation_matrix(correlation_matrix,nrows=1000)
df1$X1<-ifelse(abs(df1$X1) < 1,0,1)
df1$X2<-abs(df1$X2)
df1$X2<-(df1$X2-min(df1$X2))/(max(df1$X2)-min(df1$X2))
plot_separability(observed=round(abs(df1$X1),0),predicted=abs(df1$X2))
```

---

plot_trees_xgboost *Plot trees for xgboost::xgb.train*

Description

Uses `xgboost::xgb.plot.multi.trees` to draw the trees of an xgboost model combined into one diagram (keeping the 10 most important features) and saves it as a self-contained `<file>.html` in the working directory. Works with xgboost 3 and with older versions.

Usage

```
plot_trees_xgboost(model, train, file = "xgboost")
```

Arguments

model	object from xgboost::xgb.train
train	Train dataset; its column names label the features. With xgboost 3 they are used only when the model itself has no feature names and the number of columns matches the number of model features.
file	output filename

Value

Called for its side effect of writing <file>.html; returns the value of htmlwidgets::saveWidget invisibly.

Examples

```
infert_formula<-formula(case~education+spontaneous+induced)
boston_formula<-formula(medv~crim+zn+indus+chas+nox+rm+age+
                          dis+rad+tax+ptratio+black+lstat)
train_test_classification<-k_fold(df=infert,model_formula=infert_formula)
train_test_regression<-k_fold(df=MASS::Boston,model_formula=boston_formula)
xgb_classification<-xgboost::xgb.train(
                    data=train_test_classification$xgb$f1$train,
                    watchlist=train_test_classification$xgb$f1$watchlist,
                    eta=.1,
                    nthread=8,
                    nround=20,
                    objective="binary:logistic")
xgb_regression<-xgboost::xgb.train(
                data=train_test_regression$xgb$f1$train,
                watchlist=train_test_regression$xgb$f1$watchlist,
                eta=.3,
                nthread=8,
                nround=20)
# xgboost::xgb.plot.multi.trees(model=xgb_classification,features_keep=2)
# plot_trees_xgboost(model=xgb_classification,
#                    train=train_test_classification$xgb$f1,
#                    file=file.path(tempdir(),"Classification"))
# plot_trees_xgboost(model=xgb_regression,
#                    train=train_test_regression$xbg$f1,
#                    file=file.path(tempdir(),"Regression"))
```

---

plot_ts *Line plot for a time series*

Description

Converts a ts object into a ggplot2 line chart with semi-transparent points and an overlaid linear trend line. The returned ggplot object can be extended with additional layers (e.g. geom_vline() to mark events).

Usage

```
plot_ts(df, base_size = 10, ylab = "Count", title = "")
```

Arguments

df	A ts object containing the time series to plot.
base_size	Base font size passed to theme_bw(). Default is 10.
ylab	Character string for the y-axis label. Default is "Count".
title	Character string used as the plot title. Default is "".

Value

A ggplot object showing the time series as a line with points, a linear trend line fitted via lm, and a caption reporting the total number of observations.

Examples

```
ts_data <- ts(UKDriverDeaths, start = 1969, end = 1984, frequency = 12)
result <- plot_ts(ts_data, title = "UK driver deaths")
for (i in 1969:1984) {
  result <- result + geom_vline(xintercept = i, color = "blue", size = 1, alpha = .5)
}
result
autoplot(stl(ts_data, s.window = "periodic")) +
  theme_bw(base_size = 10) +
  labs(title = "UK driver deaths")
forecast::gglagplot(data.frame(ts_data), do.lines = FALSE, lags = 100) +
  theme_bw(base_size = 10) + labs(title = "UK driver deaths", y = "count")
```

---

proportion_accurate *Proportion overall accuracy of a confusion matrix*

Description

Calculates the overall accuracy and Cohen's kappa statistics of a confusion matrix.

Usage

```
proportion_accurate(observed, predicted)
```

Arguments

observed	Vector of observed variables. These are the true class labels.
predicted	Vector of predicted variables. These are the predicted class labels.

Details

This function evaluates the performance of a confusion matrix by calculating the overall accuracy and Cohen's kappa statistics.

The function performs the following steps: 1. Computes the confusion matrix from the observed and predicted values. 2. Calculates the diagonal proportion (overall accuracy) and the off-diagonal proportion. 3. Computes Cohen's kappa statistics (unweighted,linear,and squared weights).

The output is a data.frame containing the following metrics: -'cm_diagonal': Proportion of correct classifications (diagonal elements). -'cm_off_diagonal': Proportion of misclassified observations (off-diagonal elements). -'kappa_unweighted': Cohen's kappa statistic with no weights. -'kappa_linear': Cohen's kappa statistic with linear weights. -'kappa_squared': Cohen's kappa statistic with squared weights.

Examples

```
# Example with numeric observed and predicted values
proportion_accurate(observed=c(1,2,3,4,5,10),predicted=c(1,2,3,4,5,11))
```

---

questions_by_keys *Convert a key vector to a list of question indices by dimension*

Description

Takes a scoring key that maps each question to a dimension and returns a list where each element contains the indices of questions belonging to that dimension.

Usage

```
questions_by_keys(key)
```

Arguments

key	Integer vector. Each element indicates which dimension the corresponding question belongs to. Values must be consecutive integers starting from 1 up to the number of dimensions.

Value

A named list of length `max(key)`, where element `i` contains the integer indices of all questions assigned to dimension `i`.

Examples

```
key <- c(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
questions_by_keys(key)
```

`questions_dimensions_dataframe` *Build a question-to-dimension mapping table*

Description

Returns a data frame that maps each question to its dimension, including questionrder, short dimension name, and full dimension description. Useful for documenting scoring keys and validating test structure.

Usage

```
questions_dimensions_dataframe(
  key,
  dimensions,
  elaborate_dimensions,
  questions
)
```

Arguments

`key`	Integer vector. Each element indicates which dimension the corresponding question belongs to. Values must be consecutive integers starting from 1 up to the number of dimensions.
`dimensions`	Character vector. Short dimension names, one per dimension. Length must equal `max(key)`.
`elaborate_dimensions`	Character vector. Full dimension descriptions, one per dimension. Length must equal `max(key)`.
`questions`	Character vector. Question labels in the same order as `key`. Length must equal `length(key)`.

Value

A data frame with one row per question and four columns:

ORDER The question's position index within its dimension.

DIMENSION The short dimension name the question belongs to.

ELABORATE DIMENSION The full dimension description.

QUESTION The question label.

See Also

`questions_by_keys`

Examples

```
key <- c(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
dimensions <- paste0("Dimension", 1:10)
elaborate_dimensions <- paste0("Elaborated_Dimension", 1:10)
questions <- paste0("Question", 1:65)
questions_dimensions_dataframe(key, dimensions, elaborate_dimensions, questions)
```

---

rad2deg *Convert radians to degrees*

Description

Convert radians to degrees

Usage

```
rad2deg(radians)
```

Arguments

radians	radians

Examples

```
rad2deg(pi)
```

---

rank_df_to_binary *Convert scale to thurstonian binary with n items per block and n blocks*

Description

Convert scale to thurstonian binary with n items per block and n blocks

Usage

```
rank_df_to_binary(mydata, items, reverse = TRUE)
```

Arguments

mydata	dataframe
items	number of items in block
reverse	if TRUE assumes that the highest value is first item in rank if FALSE the lowest value is the first item in rank

Examples

```
set.seed(12345)
mydata<-data.frame(i1=rnorm(10,mean=2,sd=.5),
                   i2=rnorm(10,mean=2,sd=.5),
                   i3=rnorm(10,mean=2,sd=.5),
                   i4=rnorm(10,mean=2,sd=.5),
                   i5=rnorm(10,mean=2,sd=.5),
                   i6=rnorm(10,mean=2,sd=.5))
rank_df_to_binary(mydata[,c("i1","i2","i3","i4")],4)
rank_df_to_binary(mydata,3)
```

---

rank_to_binary *Convert scale to thurstonian binary with n items per ranking block*

Description

Convert scale to thurstonian binary with n items per ranking block

Usage

```
rank_to_binary(mydata, items, reverse = TRUE)
```

Arguments

mydata	dataframe
items	number of items in block
reverse	if TRUE assumes that the highest value is first item in rank if FALSE the lowest value is the first item in rank

Examples

```
set.seed(12345)
mydata<-data.frame(i1=round(rnorm(10,mean=2,sd=1),2),
                   i2=round(rnorm(10,mean=2,sd=1),2),
                   i3=round(rnorm(10,mean=2,sd=1),2),
                   i4=round(rnorm(10,mean=2,sd=1),2),
                   i5=round(rnorm(10,mean=2,sd=1),2),
                   i6=round(rnorm(10,mean=2,sd=1),2))
rank_to_binary(mydata[,c("i1","i2","i3")],items=3)
rank_to_binary(mydata[,c("i1","i2","i3")],items=3,reverse=FALSE)
rank_to_binary(mydata,items=3)
```

rank3_to_triplets *Convert thurstonian binary triplets to scale*

Description

Convert thurstonian binary triplets to scale

Usage

```
rank3_to_triplets(mydata)
```

Arguments

mydata	dataframe

Examples

```
set.seed(12345)
mydata<-data.frame(i1=rnorm(10,mean=2,sd=.5),
                   i2=rnorm(10,mean=2,sd=.5),
                   i3=rnorm(10,mean=2,sd=.5),
                   i4=rnorm(10,mean=2,sd=.5),
                   i5=rnorm(10,mean=2,sd=.5),
                   i6=rnorm(10,mean=2,sd=.5))
result<-rank_to_binary(mydata[,1:3])
rank3_to_triplets(result)
```

rbind_all *Row-bind two data frames with different column sets*

Description

Combines two data frames or matrices by rows even when they do not share the same columns. Columns present in one input but absent in the other are added and filled with NA before binding. Row names from both inputs are preserved unless they would produce duplicates, in which case default integer row names are used.

Usage

```
rbind_all(df1, df2)
```

Arguments

df1	A data frame or matrix.
df2	A data frame or matrix.

Value

A data frame containing all rows from `df1` followed by all rows from `df2`, with the union of both column sets. Cells where a column did not exist in the original input are `NA`.

Examples

```
df1 <- generate_correlation_matrix(n = 10)
df2 <- generate_correlation_matrix(n = 10)
names(df2)[4] <- "X11"
rbind_all(df1 = df1, df2 = df2)
row.names(df1) <- 21:30
rbind_all(df1 = df1, df2 = df2)
```

---

recode_scale_dummy *Scale and dummy code*

Description

Scales numeric variables between 0 and 1 and creates dummy coding for character and factor variables.

Usage

```
recode_scale_dummy(df, categories = 10)
```

Arguments

`df`	Dataframe containing the dataset to be scaled and dummy coded.
`categories`	Numeric value representing the number of unique values a vector must have to perform dummy coding. Defaults to 10.

Details

This function processes a dataframe by scaling numeric variables and creating dummy codes for character and factor variables. The numeric variables are scaled between 0 and 1,while the character and factor variables are converted to dummy variables if they have fewer unique values than the specified ‘categories‘ parameter.

The function performs the following steps: 1. Identifies numeric variables in the dataframe and scales them. 2. Identifies character and factor variables and creates dummy variables if they meet the criteria. 3. Combines the scaled numeric variables and dummy variables into a single dataframe.

The output is a dataframe with scaled numeric variables and dummy-coded character / factor variables.

Examples

```
# Example with the 'infert' dataset
recode_scale_dummy(infert)

# Example with a custom dataframe
df<-data.frame(numeric_var=c(1,2,3,4,5),
               factor_var=factor(c('A','B','A','B','C')))
recode_scale_dummy(df)
```

`remove_nc` *Replace and remove non-computable values*

Description

Cleans a data frame by replacing non-computable values (`NA`, `NaN`, `Inf`, `-Inf`, and empty strings) with a chosen replacement, then optionally drops rows or columns that still contain missing values or have zero variance.

Usage

```
remove_nc(
  df,
  value = NA,
  remove_rows = FALSE,
  aggressive = FALSE,
  remove_cols = FALSE,
  remove_zero_variance = FALSE
)
```

Arguments

`df`	A data frame to clean.
`value`	The replacement value for all non-computable entries. Default is `NA`.
`remove_rows`	Logical. When `TRUE`, rows containing `NA` after replacement are removed according to the `aggressive` setting. Default is `FALSE`.
`aggressive`	Logical. Only used when `remove_rows = TRUE`. • `TRUE` — remove a row if *any* value is `NA`. • `FALSE` — remove a row only if *all* values are `NA`. Default is `FALSE`.
`remove_cols`	Logical. When `TRUE`, columns where *all* values are `NA` are dropped. Default is `FALSE`.
`remove_zero_variance`	Logical. Only used when `remove_cols = TRUE`. When `TRUE`, columns with only one unique non-missing value (zero variance) are also dropped. Default is `FALSE`.

Value

A data frame with non-computable values replaced and, depending on the flags, rows and / or columns removed.

Examples

```
df <- mtcars
df[1, ] <- as.numeric(NaN)
df[2, ] <- as.numeric(Inf)
df[3, ] <- as.numeric(-Inf)
df[4, ] <- as.numeric(NA)
df[5, ] <- ""
remove_nc(df = df, value = NA)
cdf(remove_nc(df = df, value = NA))
df <- generate_missing(mtcars, missing = 5)
remove_nc(df, remove_rows = TRUE, aggressive = FALSE)
remove_nc(df, remove_rows = TRUE, aggressive = TRUE)
df <- generate_missing(generate_correlation_matrix(nrows = 5), missing = 2)
df$X2 <- NA
df$X3 <- 1
remove_nc(df, remove_cols = TRUE, remove_zero_variance = FALSE)
remove_nc(df, remove_cols = TRUE, remove_zero_variance = TRUE)
```

---

remove_outliers *Replace outliers with NA using IQR fences*

Description

Replaces values outside the boxplot fences with NA. The fences are computed as $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$, where $Q1$ and $Q3$ are the quantiles specified by probs. Designed to be applied across columns with sapply.

Usage

```
remove_outliers(vector, probs = c(0.25, 0.75), na.rm = TRUE, ...)
```

Arguments

vector	Numeric vector.
probs	Length-2 numeric vector giving the lower and upper quantile probabilities used to define the fence boundaries. Default is c(0.25, 0.75) (standard quartiles).
na.rm	Logical; whether to remove NA values when computing quantiles and IQR. Default is TRUE.
...	Additional arguments passed to stats::quantile().

Value

A numeric vector the same length as vector with outlying values replaced by NA.

Examples

```
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
remove_outliers(vector)
data.frame(sapply(df, remove_outliers))
```

`remove_user_packages` *Remove all user-installed packages*

Description

Uninstalls every package that is not part of the R base or recommended distribution. Packages installed in Microsoft R Open (MRO) library paths are also preserved. Only packages with no `Priority` field (i.e. neither `"base"` nor `"recommended"`) are removed. **Warning:** this operation is irreversible. All third-party packages will need to be reinstalled afterwards.

Usage

```
remove_user_packages()
```

Value

Invisibly returns a named list holding the result of `remove.packages()` for each removed package (one element per package). Called primarily for its side effect of uninstalling packages.

`replace_na_with_previous`
Last observation carried forward (LOCF) imputation

Description

Replaces each `NA` in a vector with the most recent preceding non-`NA` value (last observation carried forward, LOCF). If the first element is `NA`, it is replaced with the first non-`NA` value found anywhere in the vector. To apply LOCF to every column of a data frame use `df[] <- lapply(df, replace_na_with_previous)`.

Usage

```
replace_na_with_previous(vector)
```

Arguments

`vector`	A vector of any type that may contain `NA` values.

Value

A vector of the same length and type as `vector` with `NA` values replaced by the preceding non-`NA` element. Returns the original vector unchanged if it contains no `NA` values.

Examples

```
set.seed(12345)
df1 <- generate_missing(rnorm(10), missing = 5)
df2 <- generate_missing(rnorm(10), missing = 5)
df3 <- generate_missing(rnorm(10), missing = 5)
df4 <- generate_missing(rnorm(10), missing = 5)
df5 <- generate_missing(rnorm(10), missing = 5)
df <- data.frame(df1, df2, df3, df4, df5)
row.names(df) <- paste0("A", row.names(df))
replace_na_with_previous(df1)
df[] <- lapply(df, replace_na_with_previous)
```

---

report_alpha *Cronbach's alpha reliability report for multiple scales*

Description

Computes Cronbach's alpha and a comprehensive set of item-level reliability statistics for one or more scales using `psych::alpha()`. Supports item reversal, bootstrap confidence intervals, and optional Excel export. Scales are classified by their alpha value (Unacceptable < 0.60, Acceptable 0.60–0.70, Good and Acceptable 0.70–0.80, Good 0.80–0.90, Excellent > 0.90).

Usage

```
report_alpha(
  df,
  key = NULL,
  questions = NULL,
  reverse = NULL,
  mini = NULL,
  maxi = NULL,
  file = NULL,
  ...
)
```

Arguments

`df`	A data frame containing all item columns.
`key`	A named list where each name is a scale label and each element is a character vector of item column names belonging to that scale, e.g. `list(f1 = c("X1","X2","X3"))`. When `NULL` (default) all columns in `df` are treated as a single scale named `"dimension"`.

questions	A named list (same structure as key) of question label strings to append to item names in the output tables. When NULL (default) only column names are used.
reverse	A named list (same structure as key) of numeric sign vectors (1 = keep, -1 = reverse) passed to psych::reverse.code(). When NULL (default) no reversal is applied.
mini	Numeric scalar specifying the minimum possible scale rating used for item reversal. When NULL (default) the empirical minimum is used.
maxi	Numeric scalar specifying the maximum possible scale rating used for item reversal. When NULL (default) the empirical maximum is used.
file	Character string naming the output Excel file (without extension). When NULL (default) no file is written. The workbook contains four sheets: total statistics, bootstrap CIs (if requested), item statistics, and alpha-if-item-removed.
...	Additional arguments passed to psych::alpha(), such as cumulative, n.iter (bootstrap iterations), or check.keys.

Value

A named list with the following elements:

result_total Scale-level statistics including raw alpha, standardised alpha, Guttman's Lambda 6, average inter-item correlation, signal-to-noise ratio, mean, SD, Kaiser criterion, and alpha classification.

result_boot Bootstrap confidence intervals for alpha (only populated when n.iter > 1 is passed via ...).

result_item_statistics Item-level statistics including corrected and uncorrected item-total correlations, item mean and SD, and response frequencies.

result_dropped Alpha-if-item-removed statistics for each item within each scale.

Examples

```
# psych::alpha bootstraps (n.iter) with parallel::mclapply, which set.seed() alone
# cannot make reproducible; on a single core the results depend only on the seed
op <- options(mc.cores = 1)
set.seed(12345)
df <- data.frame(matrix(.5, ncol = 6, nrow = 6))
correlation_martix <- as.matrix(df)
diag(correlation_martix) <- 1
df <- round(generate_correlation_matrix(correlation_martix, nrows = 1000), 0) + 5
key <- list(
  f1 = c("X1", "X2", "X3"),
  f2 = c("X4", "X5", "X6")
)
reverse <- list(
  f1 = c(1, 1, 1),
  f2 = c(1, 1, 1)
)
report_alpha(df = df, key = key, cumulative = TRUE, n.iter = 1)
report_alpha(df = df, key = key, reverse = reverse, check.keys = FALSE, n.iter = 2)
```

```
report_alpha(df = df, key = key, check.keys = FALSE, n.iter = 2,
             file = file.path(tempdir(), "alpha"))
options(op)
```

`report_cfa` *Report*

Description

Report

Usage

```
report_cfa(model, file = NULL, w = 10, h = 10)
```

Arguments

`model`	lavaan object
`file`	output filename
`w`	width of pdf file
`h`	height of pdf file

Examples

```
file = file.path(tempdir(), "cfa")
model <- "LATENT=~ITEM1+ITEM2+ITEM3+ITEM4+ITEM5"
df <- lavaan::simulateData(
  model = model, model.type = "cfa",
  return.type = "data.frame", sample.nobs = 100
)
df <- generate_missing(df)
fit <- lavaan::cfa(model, data = df, missing = "ML")
report_cfa(fit)
report_cfa(fit, file = file)
```

`report_choric_serial` *Report polychoric tetrachoric polyserial biserial correlation*

Description

Report polychoric tetrachoric polyserial biserial correlation

Usage

```
report_choric_serial(
  x,
  y = NULL,
  file = NULL,
  w = 10,
  h = 10,
  type = "tetrachoric",
  ...
)
```

Arguments

x	The input may be in one of four forms: a) a data frame or matrix of dichotomous data (e.g., the lsat6 from the bock data set) or discrete numerical (i.e., not too many levels, e.g., the big 5 data set, bfi) for polychoric, or continuous for the case of biserial and polyserial b) a 2 x 2 table of cell counts or cell frequencies (for tetrachoric) or an n x m table of cell counts (for both tetrachoric and polychoric) c) a vector with elements corresponding to the four cell frequencies (for tetrachoric) d) a vector with elements of the two marginal frequencies (row and column) and the comorbidity (for tetrachoric)
y	matrix or dataframe of discrete scores. In the case of tetrachoric, these should be dichotomous, for polychoric not too many levels, for biserial they should be discrete (e.g., item responses) with not too many (<10?) categories
file	output filename
w	width of pdf file
h	height of pdf file
type	"tetrachoric" "polychoric" "polyserial" "biserial"
...	arguments passed to psych::polychoric

Examples

```
report_choric_serial(generate_data(min=0,max=1,type="uniform"),
             type="tetrachoric",file=file.path(tempdir(),"tetrachoric"))
report_choric_serial(generate_data(nrows=100,min=1,max=5,type="uniform"),
             type="polychoric")
report_choric_serial(x=psych::lsat6,y=psych::lsat6,
                    type="polyserial",file=file.path(tempdir(),"polyserial"))
report_choric_serial(x=psych::lsat6,y=psych::lsat6,
                    type="biserial",file=file.path(tempdir(),"biserial"))
```

`report_correlation` *Report correlation matrix*

Description

Report correlation matrix

Usage

```
report_correlation(
  x,
  y = NULL,
  use = "pairwise",
  method = "pearson",
  adjust = "holm",
  alpha = 0.05,
  ci = TRUE,
  file = NULL,
  w = 10,
  h = 10,
  base_size = 20,
  scatterplot = TRUE
)
```

Arguments

`x`	matrix or dataframe
`y`	a second matrix or dataframe with the same number of rows as x
`use`	"pairwise" is the default value and will do pairwise deletion of cases. "complete" will select just complete cases
`method`	"pearson" "spearman" "kendall"
`adjust`	"holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"
`alpha`	alpha level of confidence intervals
`ci`	By default, confidence intervals are found. However, this leads to a great slow-down of speed. So, for just the rs, ts and ps, set ci=FALSE
`file`	output filename
`w`	width of pdf file
`h`	height of pdf file
`base_size`	base font size
`scatterplot`	if TRUE it will output scatterplots

Examples

```
report_correlation(x=generate_missing(mtcars[,1:3],10))
report_correlation(x=generate_missing(mtcars[,1:3],10),
                   file=file.path(tempdir(),"correlation"),scatterplot=TRUE)
report_correlation(x=mtcars[,1:3],file=file.path(tempdir(),"correlation"))
```

---

report_dataframe *Write matrix or dataframe to excel sheet*

Description

Useful for generic data where conditional formatting of a specific column is required

Usage

```
report_dataframe(df, file = NULL, type = "critical_value", ...)
```

Arguments

df	dataframe or matrix
file	output filename of excel file
type	"critical_value" "matrix"
...	arguments passed to excel_critical_value or to excel_matrix

Examples

```
comment <- list(
  mpg = "Miles/(US) gallon",
  cyl = "Number of cylinders",
  disp = "Displacement (cu.in.)",
  hp = "Gross horsepower",
  drat = "Rear axle ratio",
  wt = "Weight (1000 lbs)",
  qsec = "1/4 mile time",
  vs = "Engine (0=V-shaped,1=straight)",
  am = "Transmission (0=automatic,1=manual)",
  gear = "Number of forward gears",
  carb = "Number of carburetors"
)
report_dataframe(mtcars,
 sheet = "report", file = file.path(tempdir(), "mtcars"), comment = comment, numFmt = "#0.00",
  critical = list(am = "<0.05")
)
report_dataframe(mtcars,
  sheet = "report", file = NULL, comment = comment, numFmt = "#0.00",
  critical = list(am = "<0.05")
)
```

report_efa *Output EFA model*

Description

Output EFA model

Usage

```
report_efa(
  model,
  df,
  file = NULL,
  w = 10,
  h = 5,
  cut = 0,
  base_size = 10,
  scores = FALSE
)
```

Arguments

model	psych EFA model
df	dataframe
file	output filename
w	width of pdf file
h	height of pdf file
cut	cut point for loadings
base_size	base font size
scores	if TRUE it will output factor scores in excel file

Note

Orthogonal=varimax, Oblique=oblimin

Examples

```
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="minres",oblique.scores=TRUE)
report_efa(model=model,df=mtcars,file=file.path(tempdir(),"efa"))
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="uls",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="ols",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="wls",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="gls",oblique.scores=TRUE)
```

```
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="pa",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="ml",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="minchi",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="minrank",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="old.min",oblique.scores=TRUE)
report_efa(model=model,df=mtcars)
model<-psych::fa(mtcars,nfactors=2,rotate="oblimin",fm="alpha",oblique.scores=TRUE)
#report_efa(model=model,df=mtcars)
```

`report_factorial_anova` *Factorial, Repeated Measures and Mixed ANOVA Report*

Description

Runs a factorial ANOVA for one or more dependent variables with `ez::ezANOVA`, for between-subjects, within-subjects (repeated measures) and mixed designs, and collects in one report:

- the ANOVA table with the assumption tests: Levene's test for between-subjects designs, Mauchly's test and the Greenhouse-Geisser and Huynh-Feldt corrections for within-subjects designs
- effect sizes for every term from `sjstats::anova_stats`
- pairwise post hoc comparisons for every main effect and interaction from `emmeans`

The report can also be written to an Excel workbook.

In simple terms, this tests whether the means differ across the levels of several factors and their combinations, checks the assumptions of the test and shows which conditions differ from which.

Usage

```
report_factorial_anova(
  df,
  dv,
  wid,
  within = NULL,
  within_full = NULL,
  between = NULL,
  within_covariates = NULL,
  between_covariates = NULL,
  observed = NULL,
  diff = NULL,
  reverse_diff = FALSE,
  type = 3,
```

```
  white.adjust = TRUE,
  detailed = TRUE,
  return_aov = TRUE,
  file = NULL,
  post_hoc_test = TRUE
)
```

Arguments

df	A data frame in long format, with one row per observation.
dv	Character vector with the names of the dependent variables. Each one is analysed separately.
wid	Name of the column that identifies the participant (subject). Each participant needs a unique value.
within	Character vector with the names of the within-subjects factors.
within_full	Character vector with the names of all the within-subjects factors of the full design, when within lists only a subset of them and the data have not been averaged per condition yet.
between	Character vector with the names of the between-subjects factors.
within_covariates	Character vector with the names of within-subjects covariates.
between_covariates	Character vector with the names of between-subjects covariates.
observed	Character vector with the names of factors, already listed in within or between, that are observed rather than manipulated (for example sex). They change the generalized eta-squared that ez::ezANOVA reports.
diff	Character vector with the names of factors to collapse into a difference score.
reverse_diff	If TRUE, reverses the direction of the difference score requested by diff.
type	Type of sums of squares, 1, 2 or 3 (default 3).
white.adjust	If TRUE (default), uses heteroscedasticity-corrected F tests (HC3). It affects only designs with between-subjects factors alone.
detailed	If TRUE (default), adds the sums of squares and the intercept to the ANOVA table.
return_aov	Must be TRUE (default): the effect sizes and the post hoc comparisons are computed from the aov object that ez::ezANOVA returns.
file	Name of the Excel file to write, without the extension. If NULL (default), nothing is written.
post_hoc_test	If TRUE (default), writes the post hoc comparisons to the Excel file.

Details

Before the analysis, the columns in wid, within, within_full, between and the covariates are converted to factors, and the data are averaged per participant and condition. If a participant has several observations in a condition, the ANOVA uses their mean.

The models are fitted with sum-to-zero contrasts (`contr.sum`), which Type III sums of squares need. The previous `contrasts` option is restored when the function returns.

Assumption tests:

- Levene's test (between-subjects designs): a significant result means that the variances differ across groups.
- Mauchly's test (within-subjects factors with more than two levels): a significant result means that sphericity is violated; then use the Greenhouse-Geisser (`p[GG]`) or Huynh-Feldt (`p[HF]`) corrected p-values.

The effect sizes in `omnibus_effect_size` are computed by `sjstats::anova_stats` from the `aov` object, rounded to three decimals. They use sequential (Type I) sums of squares and no heteroscedasticity correction, whatever `type` and `white.adjust` are, so their F values can differ from those in `omnibus` when the design is unbalanced or `white.adjust = TRUE`.

The post hoc comparisons are the `emmeans` pairwise comparisons for every main effect and every interaction, with the p-values adjusted (Tukey) within each effect.

The Excel workbook has a sheet for the ANOVA table ("ANOVA within", "ANOVA between" or "ANOVA"), "effect size", "post hoc" (if `post_hoc_test = TRUE`), "descriptives" and "call".

Value

A list with:

- `omnibus`: the ANOVA table of every dependent variable (`dv`), with `DFn` and `DFd` (degrees of freedom of the effect and the error), `SSn` and `SSd` (sums of squares, when reported), `F`, `p` and `ges` (generalized eta-squared, when reported). Levene's test is added with the suffix `[L]`, Mauchly's test with `[M]`, and the sphericity corrections as `GGe`, `p[GG]`, `HFe` and `p[HF]`.
- `omnibus_effect_size`: for every term, from `sjstats::anova_stats`, the eta-squared, partial eta-squared, omega-squared, partial omega-squared, epsilon-squared, Cohen's f and power
- `post_hoc`: pairwise comparisons for every main effect and interaction, with the estimate, standard error, degrees of freedom, t ratio and p-value
- `object`: the full output of `ez::ezANOVA` for every dependent variable, including the `aov` object

Source

A wrapper around `ez::ezANOVA` (Lawrence, 2026), with effect sizes from `sjstats::anova_stats` (Lüdecke, 2025) and post hoc comparisons from `emmeans` (Lenth & Piaskowski, 2026).

References

Bakeman, R. (2005). Recommended effect size statistics for repeated measures designs. Behavior Research Methods, 37(3), 379-384. doi:10.3758/BF03192707

Greenhouse, S. W., & Geisser, S. (1959). On methods in the analysis of profile data. Psychometrika, 24(2), 95-112. doi:10.1007/BF02289823

Huynh, H., & Feldt, L. S. (1976). Estimation of the Box correction for degrees of freedom from sample data in randomized block and split-plot designs. Journal of Educational Statistics, 1(1), 69-82. doi:10.3102/10769986001001069

Lawrence, M. A. (2026). ez: Easy analysis and visualization of factorial experiments (R package version 4.5-0). doi:10.32614/CRAN.package.ez

Lenth, R., & Piaskowski, J. (2026). emmeans: Estimated marginal means, aka least-squares means (R package version 2.0.4). doi:10.32614/CRAN.package.emmeans

Long, J. S., & Ervin, L. H. (2000). Using heteroscedasticity consistent standard errors in the linear regression model. The American Statistician, 54(3), 217-224. doi:10.1080/00031305.2000.10474549

Lüdecke, D. (2025). sjstats: Statistical functions for regression models (R package version 0.19.1). doi:10.5281/zenodo.1284472

Mauchly, J. W. (1940). Significance test for sphericity of a normal n-variate distribution. The Annals of Mathematical Statistics, 11(2), 204-209. doi:10.1214/aoms/1177731915

Examples

```
# 36 participants: B1 and B2 vary between participants (4 per cell),
# W1 and W2 vary within participants (every participant has all 9 conditions)
set.seed(12345)
df <- expand.grid(W1 = c("A", "B", "C"), W2 = c("D", "E", "F"), id = 1:36,
                  stringsAsFactors = FALSE)
df$B1 <- c("G", "H", "I")[(df$id - 1) %% 3 + 1]
df$B2 <- c("J", "K", "L")[(df$id - 1) %/% 3 %% 3 + 1]
df <- df[, c("id", "B1", "B2", "W1", "W2")]
correlation_matrix <- matrix(0.01, ncol = 4, nrow = 4)
diag(correlation_matrix) <- 1
dvs <- generate_correlation_matrix(correlation_matrix, nrows = nrow(df)) + 10
names(dvs) <- paste0("DV", 1:4)
df <- data.frame(df, dvs)
# DV1 differs across the levels of W1 and B1
df$DV1 <- df$DV1 + match(df$W1, c("A", "B", "C")) + match(df$B1, c("G", "H", "I"))
# within-subjects design
r1 <- report_factorial_anova(
  df = df, wid = "id", dv = c("DV1", "DV2"),
  within = c("W1", "W2"), within_full = c("W1", "W2"),
  file = file.path(tempdir(), "anova_within")
)
# between-subjects design
r2 <- report_factorial_anova(
  df = df, wid = "id", dv = c("DV1", "DV2"),
  between = c("B1", "B2"),
  file = file.path(tempdir(), "anova_between")
)
# mixed design
r3 <- report_factorial_anova(
  df = df, wid = "id", dv = c("DV1", "DV2"),
  within = c("W1", "W2"), within_full = c("W1", "W2"),
  between = c("B1", "B2"),
  file = file.path(tempdir(), "anova_mixed"),
  post_hoc_test = FALSE
)
# within-subjects design with within-subjects covariates
r4 <- report_factorial_anova(
  df = df, wid = "id", dv = c("DV1", "DV2"),
```

```
  within = c("W1", "W2"), within_full = c("W1", "W2"),
  within_covariates = c("DV3", "DV4"),
  file = file.path(tempdir(), "anova_within_cov")
)
```

`report_hlr` *Report HLR*

Description

Report HLR

Usage

```
report_hlr(
  df,
  corlist,
  factorlist,
  predictor,
  random_effect,
  file = NULL,
  sheet = "report"
)
```

Arguments

`df`	dataframe
`corlist`	Numeric outcome index
`factorlist`	Numeric predictor index
`predictor`	Character predictor name
`random_effect`	Character random effect name
`file`	Character file
`sheet`	Character sheet

Examples

```
report_hlr(df=infert,corlist=8,factorlist=1,
           predictor="case",random_effect="case")
```

report_irt *Output for irt model*

Description

Output for irt model

Usage

```
report_irt(model, file = NULL)
```

Arguments

model	object mirt
file	output filename

Examples

```
set.seed(12345)
cormatrix<-psych::sim.rasch(nvar=5,n=50000,low=-4,high=4,d=NULL,a=1,mu=0,sd=1)$items
irt_onefactor<-mirt::mirt(cormatrix,1,empiricalhist=TRUE,calcNull=TRUE)
irt_twofactor<-mirt::mirt(cormatrix,2,empiricalhist=TRUE,calcNull=TRUE)
irt_threefactor<-mirt::mirt(cormatrix,3,empiricalhist=TRUE,calcNull=TRUE)
report_irt(model=irt_onefactor,file=file.path(tempdir(),"one_factor"))
report_irt(model=irt_twofactor,file=file.path(tempdir(),"two_factors"))
report_irt(model=irt_threefactor,file=file.path(tempdir(),"three_factors"))
```

report_lda *Report for MASS::lda*

Description

Report for MASS::lda

Usage

```
report_lda(model, file = NULL)
```

Arguments

model	object from MASS::lda
file	output filename

Examples

```
model<-MASS::lda(case~.,data=infert)
result<-report_lda(model=model)
result<-report_lda(model=model,file=file.path(tempdir(),"lda"))
model<-MASS::lda(Species~.,data=iris)
result<-report_lda(model=model,file=file.path(tempdir(),"lda"))
```

`report_logistic` *Report logistic regression*

Description

Report logistic regression

Usage

```
report_logistic(
  model,
  validation_data = NULL,
  file = NULL,
  title = "",
  w = 10,
  h = 10,
  base_size = 10,
  fast = FALSE
)
```

Arguments

`model`	object glm
`validation_data`	validation data
`file`	output filename
`title`	plot title
`w`	width of pdf file. Relevant only when file string is not empty
`h`	height of pdf file. Relevant only when file string is not empty
`base_size`	base font size
`fast`	if TRUE it will not output individual scores and residuals

Note

(1) Problematic values for standardized residuals > +-1.96
Standardized residuals are residuals divided by an estimated standard deviation and they can be interpreted as z scores in that:
95 99 99.99 (2) Problematic values for dfBeta >=1
dfBeta estimates coefficients if the respective case is removed from the dataset
(3) Problematic values for Hat values (leverage) 2 or 3 times the average (k+1/n)
Hat values (leverage),gauge the influence of the observed value of the outcome variable over the predicted values
The average leverage value is defined as (k+1)/n, k=number of predictors, n=number of participants.
Leverage values lie between 0 (no influence) and 1 (complete influence over prediction)
If no cases exert undue influence over the model then all leverage values should be close to (k+1)/n
Hoaglin and Welsch (1978) recommends investigating cases with values greater than twice the average (2(k+1)/n)
Stevens (2002) recommends investigating cases with values greater than three times the average (3(k+1)/n)
(4) Problematic values for VIFs > 10
ASSUMPTIONS
(1) Linearity between continuous predictors and the logit (test whether the interaction term between the predictor and its log transformation is significant)
(2) Independence of errors
(3) No multicolinearity

Examples

```
modelcategoricalpredictor0 <- glm(case ~ education, data = infert, family = binomial)
modelcategoricalpredictor1 <- glm(case ~ education, data = infert, family = gaussian)
# modelcategoricalpredictor2<-glm(case ~ education, data = infert, family = Gamma)
# modelcategoricalpredictor3<-glm(case ~ education, data = infert, family = inverse.gaussian)
modelcategoricalpredictor4 <- glm(case ~ education, data = infert, family = poisson)
modelcategoricalpredictor5 <- glm(case ~ education, data = infert, family = quasi)
modelcategoricalpredictor6 <- glm(case ~ education, data = infert, family = quasibinomial)
modelcategoricalpredictor7 <- glm(case ~ education, data = infert, family = quasipoisson)
modelcontinuouspredictor0 <- glm(case ~ stratum, data = infert, family = binomial)
modeltwopredictors0 <- glm(case ~ education + stratum, data = infert, family = binomial)
modeltwopredictors1 <- glm(case ~ education + stratum, data = infert, family = gaussian)
# modeltwopredictors2<-glm(case ~ education + stratum, data = infert, family = Gamma)
# modeltwopredictors3<-glm(case ~ education + stratum, data = infert, family = inverse.gaussian)
modeltwopredictors4 <- glm(case ~ education + stratum, data = infert, family = poisson)
modeltwopredictors5 <- glm(case ~ education + stratum, data = infert, family = quasi)
modeltwopredictors6 <- glm(case ~ education + stratum, data = infert, family = quasibinomial)
modeltwopredictors7 <- glm(case ~ education + stratum, data = infert, family = quasipoisson)
report_logistic(model = modelcategoricalpredictor0)
report_logistic(model = modelcategoricalpredictor1)
# report_logistic(model=modelcategoricalpredictor2)
# report_logistic(model=modelcategoricalpredictor3)
report_logistic(model = modelcategoricalpredictor4)
report_logistic(model = modelcategoricalpredictor5)
report_logistic(model = modelcategoricalpredictor6)
```

```
report_logistic(model = modelcategoricalpredictor7)
report_logistic(model = modelcontinuouspredictor0)
report_logistic(model = modeltwopredictors0)
report_logistic(
  model = modelcategoricalpredictor0,
  file = file.path(tempdir(), "logistic_categorical_predictor"),
  validation_data = infert
)
report_logistic(
  model = modelcontinuouspredictor0,
  file = file.path(tempdir(), "logistic_continuous_predictor"),
  validation_data = infert
)
report_logistic(
  model = modeltwopredictors0,
  file = file.path(tempdir(), "logistic_two_predictors"),
  validation_data = infert[1:10, ]
)
```

`report_manova` *MANOVA Report*

Description

Reports the four multivariate test statistics (Pillai's trace, Wilks' lambda, the Hotelling-Lawley trace and Roy's largest root) with their approximate F tests for every term of a `manova` model, and the Type III MANOVA table from `car::Anova`. The tables are printed and can also be written to an Excel workbook.

In simple terms, this tests whether groups differ on several outcomes considered together, rather than on each outcome separately.

Usage

```
report_manova(model, file = NULL)
```

Arguments

`model`	A model fitted with `stats::manova`.
`file`	Name of the Excel file to write, without the extension. If `NULL` (default), nothing is written.

Details

With $\lambda_1, \ldots, \lambda_s$ the eigenvalues of HE^{-1}, where H is the hypothesis and E the error sums of squares and cross-products matrix of a term:

- Pillai's trace, $V = \sum \lambda_i/(1 + \lambda_i)$: the sum of the proportions of variance explained on the discriminant functions. It is the most robust of the four to violations of the assumptions and a good default.

- Wilks' lambda, $\Lambda = \prod 1/(1+\lambda_i)$: the proportion of variance not explained; smaller values mean larger effects.
- Hotelling-Lawley trace, $T = \sum \lambda_i$: the sum of the ratios of explained to unexplained variance.
- Roy's largest root, λ_1: uses only the first discriminant function. Its F statistic is an upper bound, so its p-value is a lower bound and the test is liberal.

The four tests agree when the term has one degree of freedom. The table also includes the intercept.

The Type III table tests each term adjusted for all the others, so it needs sum-to-zero contrasts such as `contr.sum` or `contr.helmert`. Set them with `options(contrasts = c("contr.sum", "contr.poly"))` before fitting the model with `manova`; `report_manova` does not change them.

Assumptions of MANOVA:

- independent observations, randomly sampled
- dependent variables measured on an interval scale
- multivariate normality of the dependent variables within each group
- equal variance-covariance matrices across groups: equal variances of each dependent variable and equal correlations between them in every group

The Excel workbook has the sheets "critical" (the four tests) and "call".

Value

Invisibly, a list with:

- `multivariate`: data frame with the Pillai, Wilks, Hotelling-Lawley and Roy statistics, their approximate F tests and p-values (column `type` names the test)
- `type_three`: Type III MANOVA table from `car::Anova`
- `call`: the model call

References

Fox, J., & Weisberg, S. (2019). An R companion to applied regression (3rd ed.). Sage. https://www.john-fox.ca/Companion/

Hotelling, H. (1951). A generalized T test and measure of multivariate dispersion. In J. Neyman (Ed.), Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability (pp. 23-41). University of California Press.

Olson, C. L. (1976). On choosing a test statistic in multivariate analysis of variance. Psychological Bulletin, 83(4), 579-586. doi:10.1037/00332909.83.4.579

Pillai, K. C. S. (1955). Some new test criteria in multivariate analysis. The Annals of Mathematical Statistics, 26(1), 117-121. doi:10.1214/aoms/1177728599

Roy, S. N. (1953). On a heuristic method of test construction and its use in multivariate analysis. The Annals of Mathematical Statistics, 24(2), 220-238. doi:10.1214/aoms/1177729029

Wilks, S. S. (1932). Certain generalizations in the analysis of variance. Biometrika, 24(3/4), 471-494. doi:10.2307/2331979

Examples

```
## Set orthogonal contrasts.
op <- options(contrasts = c("contr.helmert", "contr.poly"))
model_mixed <- manova(cbind(yield, foo) ~ N * P * K, within(npk, foo <- rnorm(24)))
model_between <- manova(cbind(rnorm(24), rnorm(24)) ~
                          round(rnorm(24), 0) * round(rnorm(24), 0))
report_manova(model = model_mixed)
result <- report_manova(model = model_between)
result$multivariate
## Restore the previous contrasts.
options(op)
```

---

report_normality_tests

Battery of normality tests

Description

Runs eight normality tests on each numeric column of `df`. The tests are Shapiro-Wilk, Anderson-Darling, Cramér-von Mises, Shapiro-Francia, Jarque-Bera (the robust version of Gel and Gastwirth, 2008), Kolmogorov-Smirnov, Lilliefors, and Pearson chi-squared. Each column is z-standardised before testing. Columns with fewer than 8 or more than 4999 non-missing observations are skipped with a console message. Results are printed to the console; when `file` is supplied they are also written to a `.log` file and a colour-coded `.xlsx` file with significant p-values ($p \leq 0.05$) highlighted.

Usage

```
report_normality_tests(df, file = NULL)
```

Arguments

`df`	Data frame or numeric vector.
`file`	Character string naming the output files (without extension). When supplied, a `.log` and an `.xlsx` file are written. When `NULL` (default) no files are written.

Value

Invisibly returns `NULL`. Called for its side effects of printing results and optionally writing output files.

Examples

```
file = file.path(tempdir(), "normality_tests")
vector <- generate_missing(rnorm(50), missing = 10)
df <- generate_missing(mtcars[, 1:2], missing = 10)
report_normality_tests(df = df)
report_normality_tests(df = vector, file = file)
```

`report_oneway` *One-Way ANOVA Report for Several Variables*

Description

For every combination of a dependent variable (`dv`) and a grouping variable (`iv`), runs and collects in one report:

- Fisher's F test, assuming equal variances (`compute_one_way_test`)
- Welch's F test, not assuming equal variances (`compute_one_way_test`)
- the Kruskal-Wallis test (`compute_kruskal_wallis_test`)
- Levene's and Bartlett's tests of equal variances
- Tukey and Games-Howell post hoc comparisons (`compute_posthoc`)

with effect sizes, and a Bonferroni adjustment for the number of combinations. The report can also be written to an Excel workbook, with optional PDF plots.

In simple terms, this compares the group means of several outcomes across several grouping variables at once, checks whether the groups have equal variances, and shows which groups differ from which.

Usage

```
report_oneway(
  df,
  dv,
  iv,
  file = NULL,
  w = 10,
  h = 10,
  base_size = 10,
  note = "",
  title = "",
  type = "ci",
  plot_means = FALSE,
  plot_diagnostics = FALSE,
  pb = FALSE
)
```

Arguments

`df`	A data frame.
`dv`	Integer vector with the column indices of the numeric dependent variables.
`iv`	Integer vector with the column indices of the grouping variables.
`file`	Name of the output file, without the extension. If `NULL` (default), nothing is written.

w	Width of the PDF pages, in inches.
h	Height of the PDF pages, in inches.
base_size	Base font size of the plots.
note	Text for the footnote of the mean plots.
title	Title of the mean plots.
type	Error bars of the mean plots: "ci" (95% confidence interval, default), "se" (standard error), "sd" (standard deviation) or "" (none).
plot_means	If TRUE, writes plots of the group means to a PDF file.
plot_diagnostics	If TRUE, writes ANOVA diagnostic plots to a PDF file.
pb	Logical; whether to display a progress bar in the console.

Details

For each combination, rows with a missing value in the dependent or the grouping variable are removed, and groups with a single observation are dropped, because their variance cannot be estimated. Combinations left with fewer than two groups are skipped.

The Bonferroni adjustment divides 0.05 by the number of combinations of dv and iv; significant compares each p-value with that critical value. The post hoc p-values are already adjusted for the pairs within each combination.

Levene's test uses the deviations from the group means. Bartlett's test is more powerful when the data are normal but sensitive to non-normality. A significant result in either suggests unequal variances.

Which test to read:

- Fisher's F assumes equal variances (homoscedasticity); use it with the Tukey post hoc test.
- Welch's F does not assume equal variances; use it with the Games-Howell post hoc test. It is the safer default when the variances or the group sizes differ.
- The Kruskal-Wallis test does not assume normality and suits ordinal or skewed outcomes, but it is not a remedy for unequal variances.
- The Tukey test (Tukey-Kramer) handles unequal group sizes but assumes equal variances.
- The Games-Howell test handles unequal group sizes and unequal variances.

The Excel workbook contains the sheets "Fisher", "Welch", "Kruskal", "Homogeneity", "Tukey", "Games-Howell" and "Descriptives". The plots are written to PDF files named after file. With pb = TRUE, a progress bar is printed to the console while the tests run.

Value

A list with:

- instructions: short notes on when to use each test
- fisher, welch: one row per combination, with the output of compute_one_way_test (sums of squares, F, degrees of freedom, p-value, effect sizes and power)

- kruskal_wallis: the output of compute_kruskal_wallis_test, with one row per combination
- tukey, games_howell: one row per pair of groups in each combination, with the output of compute_posthoc (LEVEL names the pair)
- homogeneity: Levene's and Bartlett's tests for every combination

Every table has the columns DV and IV, bonferroni_p (the Bonferroni-adjusted critical p-value) and significant (whether p < bonferroni_p).

References

Bartlett, M. S. (1937). Properties of sufficiency and statistical tests. Proceedings of the Royal Society of London. Series A, 160(901), 268-282. doi:10.1098/rspa.1937.0109

Dunn, O. J. (1961). Multiple comparisons among means. Journal of the American Statistical Association, 56(293), 52-64. doi:10.1080/01621459.1961.10482090

Games, P. A., & Howell, J. F. (1976). Pairwise multiple comparison procedures with unequal n's and/or variances: A Monte Carlo study. Journal of Educational Statistics, 1(2), 113-125. doi:10.3102/10769986001002113

Kramer, C. Y. (1956). Extension of multiple range tests to group means with unequal numbers of replications. Biometrics, 12(3), 307-310. doi:10.2307/3001469

Kruskal, W. H., & Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association, 47(260), 583-621. doi:10.1080/01621459.1952.10483441

Levene, H. (1960). Robust tests for equality of variances. In I. Olkin (Ed.), Contributions to probability and statistics: Essays in honor of Harold Hotelling (pp. 278-292). Stanford University Press.

Welch, B. L. (1951). On the comparison of several mean values: An alternative approach. Biometrika, 38(3/4), 330-336. doi:10.2307/2332579

See Also

compute_one_way_test, compute_kruskal_wallis_test, compute_posthoc

Examples

```
report_oneway(
  df = df_blood_pressure,
  dv = c(
    which("bp_before" == names(df_blood_pressure)),
    which("bp_after" == names(df_blood_pressure))
  ),
  iv = c(
    which("sex" == names(df_blood_pressure)),
    which("agegrp" == names(df_blood_pressure))
  ),
  file = file.path(tempdir(), "anova"),
  plot_diagnostics = FALSE,
  plot_means = FALSE
)
report_oneway(df = mtcars, dv = 2:4, iv = 9:10,
```

```
              file = file.path(tempdir(), "anova_oneway_two_factor"))
report_oneway(df = mtcars, dv = 2:4, iv = 9, file = file.path(tempdir(), "anova_oneway_one_factor"))
report_oneway(
  df = mtcars, dv = 2:4, iv = 9, file = file.path(tempdir(), "anova_oneway_one_factor"),
  plot_means = TRUE, plot_diagnostics = TRUE
)
```

`report_pdf` *Save or display a list of plots as a multi-page PDF*

Description

Writes one or more plot objects to a multi-page PDF file using `cairo_pdf`, optionally also printing them to the active graphics device.

Usage

```
report_pdf(
  ...,
  plotlist = NULL,
  file = NULL,
  title = NULL,
  w = 10,
  h = 10,
  print_plot = TRUE
)
```

Arguments

`...`	Plot objects passed directly (ggplot or recorded plots).
`plotlist`	A list of plot objects. Combined with any plots passed via `....`
`file`	Character or `NULL`. Output filename without extension. If `NULL`, no PDF is written. Default is `NULL`.
`title`	Character or `NULL`. Optional suffix appended to `file` (separated by an underscore) to form the final filename. Default is `NULL`.
`w`	Numeric. Width of the PDF in inches. Default is `10`.
`h`	Numeric. Height of the PDF in inches. Default is `10`.
`print_plot`	Logical. If `TRUE`, plots are also printed to the active graphics device. Default is `TRUE`.

Value

Called for its side effects. Returns `NULL` invisibly.

Examples

```
file = file.path(tempdir(), "report")
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet, group = Chick)) +
  geom_line() +
  ggtitle("Growth curve for individual chicks") +
  theme_bw()
p2 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet)) +
  geom_point(alpha = .3) +
  geom_smooth(alpha = .2, size = 1, method = "loess", formula = "y~x") +
  ggtitle("Fitted growth curve per diet") +
  theme_bw()
cars_plot_multiplot <- plot_multiplot(plotlist = plot_histogram(mtcars[, 1:2]), cols = 2)
cars_plot_base <- plot_normality_diagnostics(mtcars[, 1:2])
report_pdf(p1, p2, print_plot = TRUE)
report_pdf(p1, p2, file = file, print_plot = FALSE)
report_pdf(plotlist = cars_plot_multiplot, print_plot = TRUE)
report_pdf(plotlist = cars_plot_multiplot, file = file, title = "report",
           print_plot = FALSE)
report_pdf(plotlist = cars_plot_base, print_plot = TRUE)
report_pdf(plotlist = cars_plot_base, file = file, print_plot = FALSE)
```

---

report_regression *Regression*

Description

Regression

Usage

```
report_regression(
  model,
  base_size = 10,
  title = "",
  file = NULL,
  w = 10,
  h = 10,
  plot_diagnostics = TRUE
)
```

Arguments

model	object ml
base_size	base font size
title	plot title
file	output filename
w	width of pdf file. Relevant only when file string is not empty

`h` height of pdf file. Relevant only when file string is not empty

`plot_diagnostics`
if TRUE it will output linear model diagnostics plots

Note

1. Problematic values for standardized residuals > +-1.96
   Standardized residuals are residuals divided by an estimated standard deviation and they can be interpreted as z scores in that:
   - 95.00% of z-scores lie between -1.96 and +1.96
   - 99.00% of z-scores lie between -2.58 and +2.58
   - 99.99% of z-scores lie between -3.29 and +3.29
2. **Studentized residuals**: indicate the ability of the model to predict that case. They follow a t distribution.
3. **DFFits**: indicate the difference between the adjusted predicted value and the original predicted value. Adjusted predicted value for a case refers to the predicted value of that case, when that case is excluded from model fit.
4. **Cook's distance**: indicates leverage. Problematic values for cook's distance > 1 Cook and Weisberg (1982).
5. **Hat values**: indicate leverage. Problematic values for Hat values 2 or 3 times the average (k+1/n) The average leverage value is defined as (k+1)/n, k=number of predictors, n=number of participants. Leverage values lie between 0 (no influence) and 1 (complete influence over prediction).
   - Hoaglin and Welsch (1978) recommends investigating cases with values greater than twice the average (2(k+1)/n)
   - Stevens (2002) recommends investigating cases with values greater than three times the average (3(k+1)/n)

T-tests: test the hypothesis that b's are different from 0
Multiple R^2: Variance Explained
Adjusted R^2: Indicates how much variance in Y would be accounted for if the model is derived from the population from which the sample was taken. Ideally, R^2 = Adjusted R^2
F-Statistic: tests the null hypothesis is that the overall model has no effect
Covariance ratios: critical values CVR>1+[3(k+1)/n] CRV<1-[3(k+1)/n]. In general we should obtain small values or we may have to remove cases

ASSUMPTIONS

1. variable types: All predictors must be quantitative or categorical (with two levels), and the outcome variable must be quantitative (interval data), continuous and unbounded (no constraints on the variability of the outcome)
2. Non-zero variance
3. No perfect multicollinearity
4. Predictors are uncorrelated with -external variables-

5. Homoscedasticity: At each level of the predictor variable(s), the variance of the residual terms should be constant. Residuals at each level of the predictor(s) should have similar variance (homoscedasticity)
6. Independent errors: For any two observations the residual terms should be uncorrelated (or independent)
   This eventuality is sometimes described as a lack of autocorrelation. This assumption can be tested with the Durbin-Watson test,which tests for serial correlations between errors. Specifically, it tests whether adjacent residuals are correlated The size of the Durbin-Watson statistic depends upon the number of predictors in the model and the number of observations As a very conservative rule of thumb, values less than 1 or greater than 3 are definitely cause for concern; however,values closer to 2 may still be problematic depending on your sample and model R also provides a p-value of the autocorrelation. Be very careful with the Durbin-Watson test, though, as it depends on the order of the data: if you reorder your data, you-ll get a different value
7. Normally distributed errors: It is assumed that the residuals in the model are random, normally distributed variables with a mean of 0
8. Independence: It is assumed that all of the values of the outcome variable are independent (in other words, each value of the outcome variable comes from a separate entity)
9. Linearity: The mean values of the outcome variable for each increment of the predictor(s) lie along a straight line

Examples

```
form <- formula(mpg ~ qsec)
regressionmodel <- lm(form, data = mtcars)
multipleregressionmodel <- lm(mpg ~ qsec * hp * wt * drat, data = mtcars)
res <- report_regression(model = regressionmodel, plot_diagnostics = TRUE)
res <- report_regression(model = multipleregressionmodel)
res <- report_regression(model = regressionmodel, file = file.path(tempdir(), "regression"))
res <- report_regression(
  model = multipleregressionmodel,
  file = file.path(tempdir(), "regression"),
  plot_diagnostics = TRUE
)
```

---

report_ttests *Run Pairwise t-tests and Return a Reporting Table*

Description

Performs t-tests for each selected dependent variable against each selected independent variable, across all pairwise level combinations of the independent variable. Also computes descriptive statistics, effect sizes, Bartlett homogeneity test results, and Bonferroni adjustment.

In simple terms: this function creates a full t-test report table you can export or use in downstream summaries.

Usage

```
report_ttests(df, dv, iv, file = NULL, ...)
```

Arguments

df	A data frame containing both the independent and dependent variables.
dv	Integer vector of column indices for the continuous dependent variables.
iv	Integer vector of column indices for the categorical independent variables.
file	output filename
...	Further arguments passed to stats::t.test() for every comparison, for example alternative, mu, var.equal or conf.level. The formula and data are built by the function, so do not pass x, y, formula, data or paired.

Details

Missing values are removed per analysis pair using complete cases on the current dependent and independent variables.

For each independent variable, all pairwise level combinations are tested using utils::combn.

The function also calls report_dataframe to generate a formatted report.

Value

A data frame where each row is one pairwise group comparison for one dependent-independent variable combination. Returned columns mean:

- DV: Name stored in the DV column by the current implementation. Note: this currently contains the independent variable name.
- IV: Name stored in the IV column by the current implementation. Note: this currently contains the dependent variable name.
- level1: First group level being compared.
- level2: Second group level being compared.
- n1: Sample size in level1.
- n2: Sample size in level2.
- t: t statistic from t.test.
- df: Degrees of freedom for the t statistic.
- p: p-value from t.test.
- CI_l: Lower confidence interval bound for the mean difference.
- CI_u: Upper confidence interval bound for the mean difference.
- alternative: Alternative hypothesis used by t.test.
- method: Test label from t.test (for example Welch Two Sample t-test).
- mean1: Mean of the dependent variable in level1.
- mean2: Mean of the dependent variable in level2.

- sd1: Standard deviation in level1.
- sd2: Standard deviation in level2.
- sd_pooled: Pooled standard deviation, sqrt((sd1^2 + sd2^2) / 2).
- d: Cohen d effect size, abs(mean2 - mean1) / sd_pooled.
- r: Effect-size r derived from d using the function formula.
- k_squared[bartlett]: Bartlett test statistic for equal variances.
- df[bartlett]: Degrees of freedom of Bartlett test.
- p[bartlett]: p-value of Bartlett test. Small values suggest heteroscedasticity.
- bonferroni_p: Bonferroni-adjusted alpha threshold computed for the number of tests in the output table.
- significant: Logical-like character flag (TRUE / FALSE) indicating whether p is below bonferroni_p.

Examples

```
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2
)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3
)
report_ttests(
  df = df_insurance,
  dv = which("charges" == names(df_insurance)),
  iv = c(2, 4)
)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "two.sided"
)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "less"
)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "greater"
)
report_ttests(
```

```
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  var.equal = TRUE,
  file = file.path(tempdir(), "ttest")
)
effectsize::cohens_d(bp_before ~ sex, data = df_blood_pressure, pooled_sd = TRUE)
effectsize::cohens_d(bp_after ~ sex, data = df_blood_pressure, pooled_sd = TRUE)
effectsize::hedges_g(bp_before ~ sex, data = df_blood_pressure, pooled_sd = TRUE)
effectsize::hedges_g(bp_after ~ sex, data = df_blood_pressure, pooled_sd = TRUE)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2
)
report_ttests(
  df = df_blood_pressure,
  dv = which("bp_after" == names(df_blood_pressure)),
  iv = 2
)
```

`report_wtests` *Run Pairwise Wilcoxon Tests and Return a Reporting Table*

Description

Performs Wilcoxon rank-sum tests for each selected dependent variable against each selected independent variable, across all pairwise level combinations of the independent variable. Also computes descriptive statistics, effect sizes, Bartlett homogeneity results, and Bonferroni adjustment.

In simple terms: this function builds a full nonparametric comparison table, similar to report_ttests, but using wilcox.test for group differences.

Usage

```
report_wtests(df, dv, iv, file = NULL, ...)
```

Arguments

`df`	A data frame containing both the independent and dependent variables.
`dv`	Integer vector of column indices for the continuous dependent variables.
`iv`	Integer vector of column indices for the categorical independent variables.
`file`	output filename
`...`	Further arguments passed to `stats::wilcox.test()` for every comparison, e.g. `alternative`, `mu`, `exact` or `correct`. The formula and data are built by the function and `conf.int = TRUE` is always used, so do not pass `x`, `y`, `formula`, `data`, `paired` or `conf.int`.

Details

Missing values are removed per analysis pair using complete cases on the current dependent and independent variables.

For each independent variable, all pairwise level combinations are tested using utils::combn.

The function calls stats::wilcox.test with conf.int = TRUE and passes any additional arguments on through ....

The function also calls report_dataframe to generate a formatted report.

Value

A data frame where each row is one pairwise level comparison for one dependent-independent variable combination. Returned columns mean:

- DV: Name stored in the DV column by the current implementation. Note: this currently contains the independent variable name.
- IV: Name stored in the IV column by the current implementation. Note: this currently contains the dependent variable name.
- level1: First group level being compared.
- level2: Second group level being compared.
- n1: Sample size in level1.
- n2: Sample size in level2.
- W: Wilcoxon test statistic from wilcox.test.
- p: p-value from wilcox.test.
- CI_l: Lower confidence interval bound from wilcox.test.
- CI_u: Upper confidence interval bound from wilcox.test.
- alternative: Alternative hypothesis used by wilcox.test.
- method: Test label from wilcox.test.
- mean1: Mean of the dependent variable in level1.
- mean2: Mean of the dependent variable in level2.
- sd1: Standard deviation in level1.
- sd2: Standard deviation in level2.
- sd_pooled: Pooled standard deviation, sqrt((sd1^2 + sd2^2) / 2).
- d: Cohen d effect size, abs(mean2 - mean1) / sd_pooled.
- r: Wilcoxon effect size from compute_wilcoxon_effect_size.
- k_squared[bartlett]: Bartlett test statistic for equal variances.
- df[bartlett]: Degrees of freedom of Bartlett test.
- p[bartlett]: p-value of Bartlett test. Small values suggest heteroscedasticity.
- bonferroni_p: Bonferroni-adjusted alpha threshold computed for the number of tests in the output table.
- significant: Logical-like character flag (TRUE / FALSE) indicating whether p is below bonferroni_p.

Examples

```
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2
)
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3
)
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "two.sided"
)
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "less"
)
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  alternative = "greater"
)
report_wtests(
  df = df_blood_pressure,
  dv = which("bp_before" == names(df_blood_pressure)),
  iv = 2:3,
  var.equal = TRUE,
  file = file.path(tempdir(), "wilcoxontest")
)
```

---

report_xgboost *Report for xgboost::xgb.train*

Description

Report for xgboost::xgb.train

Usage

```
report_xgboost(
  model,
  validation_data = NULL,
```

```
  label = NULL,
  file = "xgboost",
  w = 10,
  h = 10,
  base_size = 10,
  title = "",
  fast = FALSE
)
```

Arguments

model	object from xgboost::xgb.train
validation_data	validation data
label	outcome variable name
file	output filename
w	width of pdf file
h	height of pdf file
base_size	base font size
title	plot title
fast	if TRUE error values are not saved in output

Details

Works with xgboost 3, which stores the call, parameters and evaluation log as attributes of the model, and with older versions that store them as list elements. Validation predictions use the model's own feature order. When `file` is given, the plots are written to `<file>.pdf` and the tables to `<file>.xlsx` (sheets: confusion matrix for classification, Model Summary, Validation Metrics, Feature Importance, Hyperparameters and, unless `fast = TRUE`, Evaluation Log).

Value

Invisibly, a list with

plots ggplot objects: `regression` (observed vs predicted on the validation data), `performance` (cut-point diagnostics, not for regression objectives), `depth`, `cover`, `weight` (tree structure), `error` (training log) and `importance`.

result tables: `importance` (from `xgboost::xgb.importance`), `parameters` (hyperparameters), `model_call`, `evaluation_log`, `model_summary` (objective, boosting rounds, number of features, and with early stopping the best iteration and score). With validation data, there is also `validation_metrics` (regression: N, RMSE, MAE, mean error, R squared; binary classification: N, AUC, log loss and accuracy at a 0.5 cut-off).

observed, predicted validation outcome and predictions (`NULL` without `validation_data`).

feature_names the model's features, in the model's order.

Examples

```
infert_formula<-formula(case~education+spontaneous+induced)
boston_formula<-formula(medv~crim+zn+indus+chas+nox+rm+age+
                        dis+rad+tax+ptratio+black+lstat)
train_test_classification<-k_fold(df=infert,model_formula=infert_formula)
train_test_regression<-k_fold(df=MASS::Boston,model_formula=boston_formula)
xgb_classification<-xgboost::xgb.train(
                    params=xgboost::xgb.params(objective="binary:logistic"),
                    data=train_test_classification$xgb$f1$train,
                    evals=train_test_classification$xgb$f1$watchlist,
                    nround=20)
xgb_regression<-xgboost::xgb.train(
                 data=train_test_regression$xgb$f1$train,
                 evals=train_test_regression$xgb$f1$watchlist,
                 nround=20)
report_xgboost(model=xgb_classification,
               validation_data=train_test_classification$f$test$f1,
               label=train_test_classification$outcome,
               file=file.path(tempdir(),"Classification"))
report_xgboost(model=xgb_regression,
               validation_data=train_test_regression$f$test$f1,
               label=train_test_regression$outcome,
               file=file.path(tempdir(),"Regression"))
```

---

response_dimension *index parameter and items relative to their dimensions*

Description

index parameter and items relative to their dimensions

Usage

```
response_dimension(response, dimensions, items)
```

Arguments

response	vector one to number of items
dimensions	number of dimensions
items	item comparisons

Examples

```
response_dimension(c(1:18),3,c(1,2))
response_dimension(c(1:18),3,c(1,3))
response_dimension(c(1:18),3,c(2,3))
```

`response_frequency` *Response frequency table for ordinal or Likert-scale variables*

Description

Tabulates how often each response category was chosen for one or more ordinal variables (e.g. Likert scale items). For each variable the function returns the count, proportion, or percentage of respondents who selected each response option, along with the number of missing or out-of-range responses. The function is a guard against accidental use on continuous variables: if the number of unique values exceeds `max` the function returns `NULL`.

Usage

```
response_frequency(
  df,
  max = 10,
  uniqueitems = NULL,
  type = "percent",
  file = NULL
)
```

Arguments

`df`	A data frame whose columns are the ordinal variables to tabulate.
`max`	Integer. Maximum number of unique response options allowed before the function returns `NULL`. Use this to prevent accidentally tabulating continuous variables. Default is `10`.
`uniqueitems`	Vector of all valid response values (e.g. `1:5` for a 5-point Likert scale). When `NULL` (default) the unique values observed in `df` are used.
`type`	Character string controlling the metric returned. One of `"frequency"` (raw counts), `"proportion"` (counts divided by valid responses), `"percent"` (proportion multiplied by 100, default), or `"all"` (all three metrics stacked row-wise).
`file`	Character string naming the output Excel file (without extension). When `NULL` (default) no file is written.

Value

A data frame with one row per variable (three rows per variable when `type = "all"`) containing the following columns:

type `"Frequency"`, `"Proportion"`, or `"Percent"`.

variable Name of the column from `df`.

(response columns) One column per value in `uniqueitems`, named by the response value, containing the frequency, proportion, or percent of respondents who chose that category.

miss Observations with values outside `uniqueitems`. For proportions this is the missing rate; for percent it is the missing percentage.

responses Total number of valid (non-missing) responses.

Returns `NULL` if the number of unique values exceeds `max`.

Examples

```
file = file.path(tempdir(), "descriptives")
df_ocean_N<-df_ocean[,grep("N",names(df_ocean))]
response_frequency(df_ocean_N)
response_frequency(df_ocean_N,
                   uniqueitems = 1:5)
response_frequency(df_ocean_N,
                   uniqueitems = 1:5,
                   type = "proportion")
response_frequency(df_ocean_N,
                   uniqueitems = 1:5,
                   type = "percent")
response_frequency(df_ocean_N,
                   uniqueitems = 1:5,
                   type = "all")
response_frequency(df_ocean_N,
                   uniqueitems = 1:5,
                   type = "all",
  file = file)
```

---

result_confusion_performance

Plot performance of confusion matrix for different cut off points

---

Description

This function generates a plot to visualize the performance of a confusion matrix at various cut-off points. It evaluates the proportion of correct classifications and identifies the optimal cut-off point.

Usage

```
result_confusion_performance(
  observed,
  predicted,
  step = 0.1,
  base_size = 10,
  title = ""
)
```

Arguments

observed	Vector of observed outcomes. These are the true class labels.
predicted	Vector of predicted outcome probabilities. These are the predicted probabilities for the positive class.
step	Numeric value representing the stepping for tested cut values. Defaults to 0.1.
base_size	Integer value representing the base font size for the plot. Defaults to 10.
title	String representing the title of the plot. Defaults to an empty string.

Details

This function evaluates the performance of a confusion matrix at different cut-off points. It iterates through a range of cut-off points, calculates the confusion matrix,and evaluates the proportion of correct classifications for each cut-off.

The function generates a plot that includes: -The proportion of correct classifications for different cut-off points. -Vertical lines indicating the optimal cut-off point. -A legend representing different performance metrics. -A caption showing the number of observations and the optimal cut-off point.

The function returns a list containing the plot,the data frame with cut-off performance,the optimal cut-off point, and the confusion matrix at the optimal cut-off.

Examples

```
# Example with numeric class labels
df<-data.frame(matrix(.999,ncol=2,nrow=2))
correlation_matrix<-as.matrix(df)
diag(correlation_matrix)<-1
df<-generate_correlation_matrix(correlation_matrix,nrows=1000)
df$X1<-ifelse(abs(df$X1) < 1,0,1)
df$X2<-abs(df$X2)
df$X2<-(df$X2-min(df$X2))/(max(df$X2)-min(df$X2))
result_confusion_performance(observed=round(abs(df$X1),0),
                             predicted=abs(df$X2),
                             step=0.01)
```

---

round_dataframe *Round numeric columns in a data frame*

Description

Applies a rounding or transformation function to every numeric column in a data frame, leaving non-numeric columns (factor, character, etc.) unchanged.

Usage

```
round_dataframe(df, digits = 0, type = "round")
```

Arguments

df	A data frame containing a mix of numeric and non-numeric columns.
digits	Integer number of decimal places. Only used with type = "round" and type = "tenth". Default is 0.
type	Character string specifying the transformation to apply to numeric columns:
	"round" Round to digits decimal places using round() (default).
	"ceiling" Round up to the nearest integer using ceiling().
	"floor" Round down to the nearest integer using floor().
	"tenth" Divide each value by 10 then round to digits decimal places — useful for rescaling values that were multiplied by 10 (e.g. converting tenths back to units).

Value

A data frame with the same structure as df where all numeric columns have been rounded or transformed according to type.

Examples

```
round_dataframe(df = change_data_type(df = mtcars, type = "factor"), digits = 0)
round_dataframe(df = change_data_type(df = mtcars, type = "character"), digits = 0)
round_dataframe(df = mtcars, digits = 0)
round_dataframe(df = mtcars, digits = 0, type = "ceiling")
round_dataframe(df = mtcars, digits = 0, type = "floor")
round_dataframe(df = mtcars * 100, digits = 2, type = "tenth")
```

---

score_tirt *Score Multiple Thurstonian IRT Response Patterns (MAP / EBM)*

Description

Scores many respondents at once using Thurstonian IRT parameters and returns MAP (empirical Bayes modal) latent trait estimates for each row in patterns.

In simple terms: this function applies score_tirt_pattern to every respondent, after first checking and aligning item columns so they match lambda row order.

Usage

```
score_tirt(patterns, lambda, theta_diag, tau, Psi, nu = NULL)
```

Arguments

patterns	A matrix or data.frame of response patterns (rows = respondents, columns = pair / items).
lambda	Loading matrix (rows = pair / items, columns = latent traits).
theta_diag	Numeric vector of residual variances aligned to rows of lambda.
tau	Numeric vector of thresholds aligned to rows of lambda.
Psi	Latent covariance matrix (traits x traits).
nu	Optional numeric vector of indicator intercepts aligned to rows of lambda. If NULL, zeros are used in score_tirt_pattern.

Details

Name alignment is the key safeguard: when both rownames(lambda) and colnames(patterns) are present, patterns is reordered to match lambda row order before scoring.

If required lambda names are missing from patterns, the function stops with an informative error. If names are unavailable, positional alignment is assumed and a warning is issued.

Value

A numeric matrix of latent scores: rows correspond to respondents in patterns, columns correspond to traits in colnames(lambda).

Examples

```
library(thurstonianIRT)
data("triplets")
# define the blocks of items
blocks <-
  set_block(c("i1", "i2", "i3"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, 1)) +
  set_block(c("i4", "i5", "i6"), traits = c("t1", "t2", "t3"),
            signs = c(-1, 1, 1)) +
  set_block(c("i7", "i8", "i9"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, -1)) +
  set_block(c("i10", "i11", "i12"), traits = c("t1", "t2", "t3"),
            signs = c(1, -1, 1))
# generate the data to be understood by 'thurstonianIRT'
triplets_long <- make_TIRT_data(
  data = triplets, blocks = blocks, direction = "larger",
  format = "pairwise", family = "bernoulli", range = c(0, 1)
)
# fit the data using lavaan
fit <- fit_TIRT_lavaan(triplets_long)
pars <- extract_tirt_params(fit)
patterns<-as.matrix(triplets)
score_tirt(patterns,lambda=pars$lambda,theta_diag=pars$theta_diag,
           tau=pars$tau,Psi=pars$Psi,nu=NULL)
# Check same scores from thurstonianIRT package
triplets_long <- make_TIRT_data(data = triplets,
```

```
                                blocks = blocks,
                                direction = "larger",
                                format = "pairwise",
                                family = "bernoulli",
                                range = c(0, 1))
scores_thurstonianIRT<-predict(fit)
scores<-score_tirt(patterns,lambda=pars$lambda,theta_diag=pars$theta_diag,
        tau=pars$tau,Psi=pars$Psi,nu=NULL)
head(reshape2::recast(scores_thurstonianIRT,formula=id~trait,id.var=1:2))
head(scores)
```

`score_tirt_pattern` *Score a Single Thurstonian IRT Response Pattern (MAP / EBM)*

Description

Computes the maximum a posteriori (MAP), also called empirical Bayes modal (EBM), estimate of latent traits for one binary / ordinal response pattern under a Thurstonian IRT parameterization.

Missing responses are allowed and are ignored in the likelihood.

Usage

```
score_tirt_pattern(
  pattern,
  lambda,
  theta_diag,
  tau,
  Psi,
  nu = NULL,
  init = NULL,
  control = list()
)
```

Arguments

`pattern`	Numeric vector of observed responses for one person, typically coded 0/1, with optional NA values for missing responses. Its order must match the row order of lambda (or be pre-aligned before calling).
`lambda`	Matrix of factor loadings with rows as observed indicators and columns as latent traits.
`theta_diag`	Numeric vector of residual variances (diagonal of theta), aligned to rows of lambda.
`tau`	Numeric vector of thresholds, aligned to rows of lambda.
`Psi`	Latent covariance matrix (traits x traits), positive definite.
`nu`	Optional numeric vector of indicator intercepts aligned to rows of lambda. If NULL, a zero vector is used.

`init`	Optional numeric vector of starting values for optimization. If NULL, starts at zeros.
`control`	Named list of control arguments passed to optim, merged with defaults reltol = 1e-10 and maxit = 500.

Details

The function maximizes the posterior: likelihood from a probit measurement model plus a multivariate normal prior on traits with covariance Psi. Optimization uses BFGS via optim with an analytic gradient.

Value

Named numeric vector of latent trait MAP estimates, with names taken from colnames(lambda).

Examples

```
library(thurstonianIRT)
data("triplets")
# define the blocks of items
blocks <-
  set_block(c("i1", "i2", "i3"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, 1)) +
  set_block(c("i4", "i5", "i6"), traits = c("t1", "t2", "t3"),
            signs = c(-1, 1, 1)) +
  set_block(c("i7", "i8", "i9"), traits = c("t1", "t2", "t3"),
            signs = c(1, 1, -1)) +
  set_block(c("i10", "i11", "i12"), traits = c("t1", "t2", "t3"),
            signs = c(1, -1, 1))
# generate the data to be understood by 'thurstonianIRT'
triplets_long <- make_TIRT_data(
  data = triplets, blocks = blocks, direction = "larger",
  format = "pairwise", family = "bernoulli", range = c(0, 1)
)
# fit the data using lavaan
fit <- fit_TIRT_lavaan(triplets_long)
pars <- extract_tirt_params(fit)
pattern<-as.numeric(triplets[1,])
score_tirt_pattern(pattern,lambda=pars$lambda,theta_diag=pars$theta_diag,
                   tau=pars$tau,Psi=pars$Psi,nu=NULL,init=NULL,
                   control=list())
```

`simulate_cfa_fit` *Simulate CFA model fit across sample sizes*

Description

Runs a confirmatory factor analysis (CFA) repeatedly across a range of sample sizes and returns fit indices for each. Useful for power analysis and sample size planning in SEM.

Two workflows are supported:

- **Coefficient-based**: supply `model_sim` with fixed loadings - data are generated from those population parameters at each sample size.
- **Correlation-based**: supply `df` - data are bootstrapped from the observed correlation structure of `df` at each sample size.

Iterations run in parallel via `future.apply`. Results are written to a PDF (scatter plots of fit indices vs. sample size) and an Excel file.

Usage

```
simulate_cfa_fit(
  model_sim = NULL,
  model = NULL,
  df = NULL,
  minnobs = 50,
  maxnobs = 1000,
  stepping = 10,
  file = NULL,
  w = 10,
  h = 10
)
```

Arguments

`model_sim`	A `lavaan` model string with fixed coefficients (e.g. `"F =~ 1*x1 + 0.8*x2"`). Used to generate population data. Mutually exclusive with `df`; supply one or the other.
`model`	A `lavaan` model string with free coefficients specifying the CFA structure to be estimated at each sample size (e.g. `"F =~ x1 + x2"`).
`df`	A data frame of observed data. When supplied, each iteration simulates data by sampling from the observed correlation structure. Mutually exclusive with `model_sim`.
`minnobs`	Integer. Smallest sample size to evaluate. Default `50`.
`maxnobs`	Integer. Largest sample size to evaluate. Default `1000`.
`stepping`	Integer. Increment between sample sizes. Default `10`.
`file`	Character. Base name (without extension) for the output PDF and Excel files. If `NULL` no files are written.
`w`	Numeric. Width of the output PDF in inches. Default `10`.
`h`	Numeric. Height of the output PDF in inches. Default `10`.

Value

A list of two elements:

- `[[1]]` - a data frame with one row per sample size and one column per lavaan fit index (CFI, RMSEA, SRMR, etc.).
- `[[2]]` - a named list of ggplot scatter plots, one per fit index, showing how the index changes with sample size.

Examples

```
file = file.path(tempdir(), "report")
model_sim <- "LATENT =~ 1*X1 + 0.5*X2 + 1.5*X3 + 1.5*X4 + X5"
model <- "LATENT =~ X1 + X2 + X3 + X4 + X5"
# Coefficient-based: generate data from known population parameters
result <- simulate_cfa_fit(
  model_sim = model_sim, model = model,
  minnobs = 50, maxnobs = 1000, stepping = 100, file = file)
plot_multiplot(plotlist = result[[2]][1:2], cols = 2)
# Correlation-based: resample from observed data
df <- lavaan::simulateData(
  model = model_sim, model.type = "cfa",
  return.type = "data.frame", sample.nobs = 1000
)
```

---

simulate_correlation_from_sample

Simulate data preserving the correlation structure of an input data frame

---

Description

Estimates the covariance matrix and column means from the input data, then draws multivariate normal samples that reproduce the same correlation structure.

Usage

```
simulate_correlation_from_sample(cordata, nrows = 10)
```

Arguments

`cordata`	A numeric data frame or matrix. The source data from which the covariance matrix and means are estimated. Missing values are handled pairwise.
`nrows`	Integer. Number of observations to simulate. Default is `10`.

Details

Uses `mvrnorm` to draw from a multivariate normal distribution parameterised by the sample covariance matrix and column means of `cordata`. Accuracy of the reproduced correlations improves with larger `nrows`.

Value

A data frame with nrows rows and the same number of columns as cordata, containing simulated values with matching correlation structure.

See Also

generate_correlation_matrix

Examples

```
correlation_matrix <- generate_correlation_matrix()
stats::cor(correlation_matrix)
stats::cor(simulate_correlation_from_sample(correlation_matrix, nrows = 1000))
```

---

stat_word_char *Text similarity measures*

Description

Text similarity measures

Usage

```
stat_word_char(text)
```

Arguments

text	character vector

Examples

```
text<-"There are many variations of passages of Lorem Ipsum available,
but the majority have suffered alteration in some form, by injected humour,
or randomised words which don't look even slightly believable."
stat_word_char(text)
```

`str_aes` *Clean and format string aesthetics*

Description

Replaces a list of separator characters (e.g. `"."`, `"_"`, HTML tags) with spaces, trims leading and trailing whitespace, collapses internal whitespace, and optionally applies proper case.

Usage

```
str_aes(
  vector,
 characterlist = c(".", "_", "-", ",", "$", "<p>", "</p>", "<br>", "<br/>", "<B>",
    "</B>", "<BR/>", "|", "/", " "),
  proper = TRUE
)
```

Arguments

`vector`	Character vector to clean.
`characterlist`	Character vector of strings to treat as separators, each replaced by a single space. Defaults to common punctuation and HTML tags including `"."`, `"_"`, `"-"`, `"<p>"`, `" "`, `" "`, and others.
`proper`	Logical. If `TRUE`, capitalises the first letter and lowercases the rest of each string. Default is `TRUE`.

Value

A character vector of the same length as `vector` with separators replaced, whitespace normalised, and optional proper casing.

Examples

```
vector <- c("TES.T", "TES<p>T", "TES&nbspT")
str_aes(vector = vector)
str_aes(vector = vector, proper = FALSE)
str_aes(vector = vector, proper = TRUE)
```

str_count *Count the number of pattern matches in a string*

Description

Returns the number of times `pattern` appears in each element of `string`. Supports both regular expressions and literal string matching via `fixed()`.

Usage

```
str_count(string, pattern)
```

Arguments

`string`	A character vector.
`pattern`	A regular expression string or a literal string wrapped in `fixed()`.

Value

An integer vector the same length as `string`.

Examples

```
# Count vowels
str_count(c("banana", "apple", "cherry"), "[aeiou]")

# Count literal semicolons (useful for delimited data)
str_count(c("a;b;c", "x;y", "z"), fixed(";"))

# Count digits
str_count(c("abc123", "99bottles", "none"), "[0-9]")
```

str_mgsub *Apply gsub for multiple patterns with a single replacement*

Description

Iterates over a vector of patterns, applying `base::gsub()` sequentially with the same replacement string for each.

Usage

```
str_mgsub(mydata, pattern, replacement, ...)
```

Arguments

mydata	Character vector to search within.
pattern	Character vector of patterns to search for.
replacement	Character. The replacement string applied for all patterns.
...	Additional arguments passed to base::gsub(),

Value

A character vector with all pattern matches replaced.

Examples

```
str_mgsub(mydata = "#$%^&*_+", pattern = c("%", "*"), "REPLACE", fixed = TRUE)
```

---

str_pad *Pad a string to a minimum width*

Description

Pads string with pad characters on the left, right, or both sides until it reaches at least width characters. Strings already at or exceeding width are returned unchanged.

Usage

```
str_pad(string, width, side = "right", pad = " ")
```

Arguments

string	A character vector.
width	Integer. Minimum total width of the output string.
side	One of "right" (default), "left", or "both".
pad	A single character to use for padding. Default " ".

Value

A character vector the same length as string.

Examples

```
# Zero-pad single digit numbers on the left
str_pad(c("1", "10", "100"), width = 3, side = "left", pad = "0")

# Right-pad to align labels
str_pad(c("Name", "Age", "Score"), width = 10)

# Pad on both sides (centers the string)
str_pad("hello", width = 11, side = "both")
```

str_proper *Convert a string to proper case*

Description

Capitalises the first character and lowercases the rest of each element.

Usage

```
str_proper(x)
```

Arguments

x	Character vector.

Value

A character vector of the same length as x.

Examples

```
x <- generate_string(nchar = 10, vector = LETTERS, vector_length = 10)
str_proper(x)
```

str_replace *Replace the first pattern match in a string*

Description

Replaces only the first occurrence of pattern in each element of string. For replacing all occurrences use str_replace_all.

Usage

```
str_replace(string, pattern, replacement)
```

Arguments

string	A character vector.
pattern	A regular expression string or a literal string wrapped in fixed().
replacement	A character string to replace the first match with.

Value

A character vector the same length as string.

Examples

```
# Only the first "o" is replaced
str_replace("hello world", "o", "0")

# Remove leading zero (first match only)
str_replace("007 bond", "^0+", "")

# Fixed match: replace first literal dot
str_replace("a.b.c", fixed("."), "-")
```

`str_replace_all` *Replace all pattern matches in a string*

Description

Replaces every occurrence of `pattern` in `string` with `replacement`. Supports both regular expressions and literal string matching via `fixed()`.

Usage

```
str_replace_all(string, pattern, replacement)
```

Arguments

`string`	A character vector.
`pattern`	A regular expression string, or a literal string wrapped in `fixed()`, or a named character vector where names are regex patterns and values are replacements (applied sequentially).
`replacement`	A character string to replace each match with. Use `""` to delete matches.

Value

A character vector the same length as `string`.

Examples

```
# Regex replacement
str_replace_all("hello world", "o", "0")

# Fixed (literal) replacement
str_replace_all("a.b.c", fixed("."), "-")

# Remove all spaces
str_replace_all("remove all spaces", fixed(" "), "")

# Named vector: multiple replacements applied in order
str_replace_all("aabbcc", c("a" = "X", "b" = "Y"))
```

str_split *Split a string vector into a data frame of parts*

Description

Splits each element of a character vector by a separator and returns theparts as columns of a data frame, one row per input element.

Usage

```
str_split(vector, split = "/", include_original = FALSE)
```

Arguments

vector	Character vector to split.
split	Character. The separator to split on. Default is "/".
include_original	Logical. If TRUE, appends the original input as a final column. Default is FALSE.

Value

A data frame with one row per element of vector and one column per split part. Assumes all elements produce the same number of parts.

Examples

```
string <- paste0(
  1:10, "/",
  generate_string(nchar = 2, vector_length = 10), "/",
  generate_string(nchar = 2, vector_length = 10), "/",
  generate_string(nchar = 2, vector_length = 10)
)
str_split(string, split = "/")
```

str_split_df *Split a string column or row names in a data frame into separate columns*

Description

Splits a delimited string — either from row names or a specified column — and prepends the resulting parts as new columns to the data frame.

Usage

```
str_split_df(df, split = "/", type = "row", index, ...)
```

Arguments

df	A data frame.
split	Character. The separator to split on. Default is "/".
type	Character. Where to read the string from. One of: "row" Splits the row names of df. "collumn" Splits the column specified by index. Default is "row".
index	Integer. Column index to split when type = "collumn".
...	Additional arguments passed to str_split.

Value

A data frame with the split parts prepended as new columns, followed by the original columns of df.

See Also

str_split

Examples

```
df <- generate_correlation_matrix()
string <- paste0(
  1:nrow(df), "/",
  generate_string(nchar = 2, vector_length = nrow(df)), "/",
  generate_string(nchar = 2, vector_length = nrow(df)), "/",
  generate_string(nchar = 2, vector_length = nrow(df))
)
row.names(df) <- string
str_split_df(df, split = "/", type = "row")
df[, 1] <- string
str_split_df(df, split = "/", type = "collumn", index = 1)
```

---

str_split_fixed *Split strings into a fixed-width matrix of pieces*

Description

Splits each element of string by pattern and returns a character matrix with exactly n columns. If a string produces fewer than n pieces the remaining columns are filled with "".

Usage

```
str_split_fixed(string, pattern, n)
```

Arguments

string	A character vector.
pattern	A regular expression string or a literal string wrapped in fixed().
n	Integer. Number of columns in the output matrix.

Value

A character matrix with length(string) rows and n columns.

Examples

```
# Split "trait.method" labels into two columns
str_split_fixed(c("speed.run", "height.jump", "weight.lift"), fixed("."), 2)

# Split on a regex pattern
str_split_fixed(c("a1b", "c2d", "e3f"), "[0-9]", 2)

# Fewer pieces than n: remainder filled with ""
str_split_fixed(c("a.b.c", "x.y"), fixed("."), 3)
```

---

str_squish *Remove leading, trailing, and internal extra whitespace*

Description

Strips leading and trailing whitespace and collapses any internal sequences of whitespace (spaces, tabs, newlines) down to a single space.

Usage

```
str_squish(string)
```

Arguments

string	A character vector.

Value

A character vector the same length as string.

Examples

```
# Remove extra internal spaces
str_squish("  hello   world  ")

# Clean up messy column names or labels
str_squish(c("  first  name ", "last  name", "  age"))

# Handles tabs and newlines too
str_squish("line1\n\nline2\t\tword")
```

str_sub *Extract n characters from the left or right of a string*

Description

Extract n characters from the left or right of a string

Usage

```
str_sub(x, n = 2, type)
```

Arguments

x	Character vector.
n	Integer. Number of characters to extract. Default is 2.
type	Character. One of "left" or "right".

Value

A character vector of the same length as x.

Examples

```
str_sub("12345", n = 2, type = "right")
str_sub("12345", n = 2, type = "left")
```

str_trim_df *Trim whitespace from all character cells in a data frame*

Description

Applies base::strwrap() to every character cell in a data frame, removing leading and trailing whitespace.

Usage

```
str_trim_df(df)
```

Arguments

df	A data frame containing one or more character columns.

Value

A data frame of the same dimensions with whitespace trimmed from all character cells. Non-character cells are unchanged.

Examples

```
string <- data.frame(
  str1 = rep(paste0(sample(c(LETTERS, rep(" ", 10))), collapse = ""), 10),
  str2 = rep(paste0(sample(c(LETTERS, rep(" ", 10))), collapse = ""), 10),
  num1 = rnorm(10),
  stringsAsFactors = FALSE
)
str_trim_df(string)
```

---

str_wrap *Wrap long strings to a specified line width*

Description

Breaks a character string into multiple lines so that no line exceeds `width` characters. Words are kept intact; lines are joined with `"\n"`.

Usage

```
str_wrap(string, width = 80)
```

Arguments

`string`	A character vector.
`width`	Maximum number of characters per line. Default `80`.

Value

A character vector the same length as `string`, with embedded newlines inserted at word boundaries.

Examples

```
# Wrap at 30 characters
cat(str_wrap("The quick brown fox jumped over the lazy dog", width = 30))

# Wrap a vector of strings
labels <- c("Short label", "A much longer label that needs wrapping")
str_wrap(labels, width = 20)
```

swap *Reverse-score a numeric vector*

Description

Reverses the order of values in a vector by mapping each value to its mirror equivalent based on the observed levels. Useful for reverse-scoring Likert scale items.

Usage

```
swap(vector)
```

Arguments

vector	Numeric vector to reverse-score.

Value

A numeric vector of the same length with values reverse-mapped across the observed range.

Examples

```
swap(c(1:10, 1, 2, 3))
```

symmetric_matrix *Make a matrix symmetric by mirroring one triangle*

Description

Copies the lower or upper triangle of a square matrix onto the opposite triangle, producing a symmetric matrix; optionally replaces the diagonal. The row names are set to the column names, so both sides carry the same labels.

Usage

```
symmetric_matrix(matrix, duplicate = "lower", diagonal = NULL)
```

Arguments

matrix	A square matrix. Unlike matrix_triangle, a data frame is not accepted; convert it with as.matrix() first.
duplicate	Character. Use "lower" (default) to copy the lower triangle into the upper, or "upper" to copy the upper triangle into the lower.
diagonal	Value(s) to place on the diagonal: a single value (e.g. NA) or one value per diagonal element. Leave it out to keep the original diagonal; passing NULL explicitly is an error.

Value

A symmetric matrix with the same dimensions as `matrix`, whose row names are the column names of `matrix`.

See Also

`matrix_triangle`, `display_upper_lower_triangle`

Examples

```
m_lower <- matrix_triangle(matrix(1:9, nrow = 3, ncol = 3), type = "lower", diagonal = NA)
m_upper <- matrix_triangle(matrix(11:19, nrow = 3, ncol = 3), type = "upper", diagonal = NA)
symmetric_matrix(matrix = m_lower, duplicate = "lower", diagonal = NA)
symmetric_matrix(matrix = m_upper, duplicate = "upper", diagonal = NA)
```

---

tag_pos *Part of speech tagging*

Description

Part of speech tagging

Usage

```
tag_pos(text)
```

Arguments

`text`	character vector

Examples

```
text1<-"word_one word_two word_three"
text2<-"word_three word_four word_six"
text3<-"All the Lorem Ipsum generators on the Internet tend to repeat predefined
chunks as necessary, making this the first true generator on the Internet."
text4<-"It uses a dictionary of over 200 Latin words, combined with a handful of
model sentence structures, to generate Lorem Ipsum which looks reasonable."
text5<-"The generated Lorem Ipsum is therefore always free from repetition,
injected humour, or non-characteristic words etc."
text<-c(text1,text2,text3,text4,text5)
tag_pos(text)
```

`ts_smoothing` *Time series smoothing with multiple bandwidth levels*

Description

Plots a time series and overlays a family of smoothed curves using a chosen smoothing method. For bandwidth-based methods (`"kernel"`, `"lowess"`, `"splines"`, `"default"`), a sequence of bandwidth or span values from `start` to `stop` is swept and each curve is drawn in a different rainbow colour, making it easy to visually select an appropriate smoothing level. `"polynomial"` and `"linear"` ignore the bandwidth sequence and fit regression-based trend lines instead. The function uses base R graphics and produces a plot as a side effect.

Usage

```
ts_smoothing(
  df,
  start = 0.01,
  stop = 2,
  step = 0.001,
  title = "",
  type = "kernel"
)
```

Arguments

`df`	A `ts` object containing the time series to smooth.
`start`	Numeric. Starting value of the bandwidth or span sequence. Default is `0.01`.
`stop`	Numeric. Ending value of the bandwidth or span sequence. Default is `2`.
`step`	Numeric. Increment between bandwidth values. Smaller values produce more curves. Default is `0.001`.
`title`	Character string appended to the plot title. Default is `""`.
`type`	Character string specifying the smoothing method. One of: `"kernel"` Gaussian kernel smoother via `ksmooth()` — bandwidth controls the smoothing window (default). `"lowess"` Locally weighted regression via `lowess()` — `f` controls the span proportion. `"friedman"` Friedman's super-smoother via `supsmu()` — span must be in (0, 1). `"splines"` Smoothing splines via `smooth.spline()`, where `spar` controls the penalty. `"default"` Running mean filter via `filter()` — bandwidth rounded to an integer window width. `"polynomial"` Fits a centred cubic polynomial trend with and without seasonal (cos / sin) terms via `lm()`. `"linear"` Fits a simple linear trend via `lm()` and draws the regression line.

Value

Invisibly returns NULL. The function is called for its side effect of producing a base R plot.

Examples

```
ts_data <- ts(UKDriverDeaths, start = 1969, end = 1984, frequency = 12)
par(mfrow = c(2, 2))
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "default"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "polynomial"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "linear"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "kernel"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "lowess"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "friedman"
)
ts_smoothing(ts_data,
  start = .01, stop = 2, step = .01,
  title = "Driver Deaths in UK", type = "splines"
)
```

---

write_txt *Print an object and optionally save output to a log file*

Description

Prints input to the console. When file is supplied, the printed output is also captured to a .log file using sink(), and the file contents are echoed back to the console after writing, so the output is visible in both places.

Usage

```
write_txt(input, file = NULL)
```

Arguments

input	Any R object to print.
file	Character string naming the output log file without extension. A .log extension is appended automatically. When NULL (default) output is printed to the console only and no file is written.

Value

Invisibly returns NULL. Called for its side effects of printing and optionally writing to a log file.

Examples

```
write_txt(mtcars)
write_txt(mtcars, file = file.path(tempdir(), "mtcars"))
```

Index

∗ **ANOVA**
compute_aov_es, 21
compute_friedman_test, 32
compute_kruskal_wallis_test, 37
compute_one_way_test, 42
compute_posthoc, 45
plot_interaction, 129
plot_oneway, 137
plot_oneway_diagnostics, 139
report_factorial_anova, 165
report_manova, 173
report_oneway, 176
∗ **CFA**
compute_efa_cfa, 29
∗ **EFA**
compute_efa_cfa, 29
compute_residual_stats, 48
model_loadings, 113
plot_loadings, 131
plot_scree, 145
report_efa, 164
∗ **IRT**
compute_info_1pl, 35
compute_info_2pl, 36
compute_info_3pl, 36
compute_scores, 49
compute_se_theta, 50
compute_unidimensional_ability, 57
compute_unidimensional_theta, 58
plot_irt_onefactor, 130
report_irt, 170
∗ **ML**
plot_trees_xgboost, 146
report_lda, 170
report_xgboost, 187
∗ **NLP**
clear_stopwords, 14
clear_text, 15
compute_text_similarity, 55
stat_word_char, 199
tag_pos, 211
∗ **SEM**
report_cfa, 160
simulate_cfa_fit, 196
∗ **Thurstonian**
compute_scores, 49
∗ **algebra**
compute_solve, 52
∗ **assumptions**
outlier_summary, 116
plot_boxplot, 120
plot_histogram, 127
plot_normality_diagnostics, 136
plot_outlier, 140
plot_qq, 141
remove_outliers, 156
report_normality_tests, 175
∗ **cfa**
check_heywood, 13
∗ **cohens-d**
compute_cohens_d, 23
∗ **correlation**
compute_power_r, 46
compute_power_r_matrix, 47
deg2rad, 65
plot_corrplot, 124
rad2deg, 151
report_choric_serial, 160
report_correlation, 162
∗ **dataframe**
cdf, 8
cdff, 10
∗ **datasets**
df_admission, 66
df_automotive_data, 67
df_blood_pressure, 68
df_co2, 69
df_crop_yield, 70

df_difficile, 70
df_insurance, 71
df_ocean, 72
df_personality, 75
df_responses, 77
df_responses_state, 82
df_sexual_comp, 82
df_titanic, 83
* **descriptives**
compute_aggregate, 19
compute_crosstable, 25
compute_descriptives, 26
compute_frequencies, 31
plot_crosstable, 125
plot_mosaic, 133
plot_response_frequencies, 141
response_frequency, 190
* **diagnostics**
cdf, 8
cdff, 10
check_heywood, 13
* **distance**
compute_tversky_index, 56
* **ebm**
score_tirt, 193
score_tirt_pattern, 195
* **effect-size**
compute_cohens_d, 23
compute_wilcoxon_effect_size, 59
report_ttests, 182
report_wtests, 185
* **friedman**
compute_friedman_test, 32
* **functions**
c_bind, 6
change_data_type, 12
combine_files, 15
comparison_combinations, 16
compute_confidence_inteval, 25
confusion, 61
confusion_matrix_percent, 62
data_frame_index, 63
detach_package, 66
display_upper_lower_triangle, 84
dotnames, 85
environment_options, 87
excel_confusion_matrix, 88
get_fwd, 105
get_script_directory, 106
install_all_packages, 108
install_load, 108
k_fold, 109
k_sample, 110
matrix_triangle, 111
min_max_index, 112
off_diagonal_index, 115
padNA, 118
plot_confusion, 122
plot_duplicate_y_axis, 126
plot_hinvert_title_grob, 127
plot_roc, 142
plot_separability, 145
proportion_accurate, 148
rbind_all, 153
recode_scale_dummy, 154
remove_nc, 155
remove_user_packages, 157
replace_na_with_previous, 157
report_dataframe, 163
result_confusion_performance, 191
round_dataframe, 192
symmetric_matrix, 210
write_txt, 213
* **generate**
generate_correlation_matrix, 99
generate_data, 100
generate_factor, 101
generate_missing, 103
simulate_correlation_from_sample, 198
* **heywood**
check_heywood, 13
* **inference**
report_ttests, 182
report_wtests, 185
* **irt**
cfa_icc_index, 11
compute_ability, 17
compute_dummy_comparisons, 28
compute_icc_thurstonian, 34
compute_map, 40
extract_tirt_params, 96
generate_comparisons_matrix, 98
generate_matrix_A, 102
generate_matrix_lambda_hat, 102
generate_unique_comparisons_index,

105
icc_cfa, 107
increase_index, 107
plot_icc_thurstonian, 128
rank3_to_triplets, 153
rank_df_to_binary, 151
rank_to_binary, 152
response_dimension, 189
score_tirt, 193
score_tirt_pattern, 195
* **kruskal**
compute_kruskal_wallis_test, 37
* **labels**
name_triplet_pairs, 114
* **lavaan**
check_heywood, 13
extract_tirt_params, 96
* **linear-equations**
compute_solve, 52
* **logistic**
compute_y_logistic, 60
output_compare_model_logistic, 116
plot_logistic_model, 132
report_logistic, 171
* **map**
score_tirt, 193
score_tirt_pattern, 195
* **matrix**
compute_solve, 52
display_upper_lower_triangle, 84
matrix_triangle, 111
off_diagonal_index, 115
symmetric_matrix, 210
* **missing-values**
cdf, 8
cdff, 10
* **nonparametric**
compute_friedman_test, 32
compute_kruskal_wallis_test, 37
compute_wilcoxon_effect_size, 59
report_wtests, 185
* **pairs**
name_triplet_pairs, 114
* **pairwise**
report_ttests, 182
report_wtests, 185
* **parametric**
compute_cohens_d, 23
* **plot**
plot_duplicate_y_axis, 126
plot_hinvert_title_grob, 127
* **recode**
drop_levels, 86
dummy_arrange, 86
flatten_list, 98
swap, 210
* **regression**
compute_y_logistic, 60
output_compare_model_logistic, 116
plot_logistic_model, 132
plot_scatterplot, 143
report_logistic, 171
report_regression, 180
* **reliability**
compute_alpha_diagnostics, 20
compute_mean_sd_alpha, 41
compute_raw_alpha, 48
compute_shrout, 50
extract_components, 95
key_to_cfa_model, 111
plot_mtmm, 134
report_alpha, 158
* **reporting**
report_ttests, 182
report_wtests, 185
* **scoring**
extract_tirt_params, 96
score_tirt, 193
score_tirt_pattern, 195
* **sem**
check_heywood, 13
* **series**
compute_moving_average, 41
plot_acf, 119
plot_ts, 147
ts_smoothing, 212
* **set**
compute_tversky_index, 56
* **similarity**
compute_tversky_index, 56
* **statistical**
compute_confidence_inteval, 25
* **strings**
call_to_string, 7
fixed, 97
output_separator, 117

str_aes, 200
str_count, 201
str_mgsub, 201
str_pad, 202
str_proper, 203
str_replace, 203
str_replace_all, 204
str_split, 205
str_split_df, 205
str_split_fixed, 206
str_squish, 207
str_sub, 208
str_trim_df, 208
str_wrap, 209
∗ **summary**
cdf, 8
cdff, 10
∗ **t-test**
report_ttests, 182
∗ **timestamp**
convert_excel_unix_timestamp, 63
decompose_datetime, 64
∗ **time**
compute_moving_average, 41
plot_acf, 119
plot_ts, 147
ts_smoothing, 212
∗ **tirt**
cfa_icc_index, 11
compute_ability, 17
compute_dummy_comparisons, 28
compute_icc_thurstonian, 34
compute_map, 40
extract_tirt_params, 96
generate_comparisons_matrix, 98
generate_matrix_A, 102
generate_matrix_lambda_hat, 102
generate_unique_comparisons_index, 105
icc_cfa, 107
increase_index, 107
plot_icc_thurstonian, 128
rank3_to_triplets, 153
rank_df_to_binary, 151
rank_to_binary, 152
response_dimension, 189
score_tirt, 193
score_tirt_pattern, 195
∗ **triplets**
name_triplet_pairs, 114
∗ **unidimensional**
compute_info_1pl, 35
compute_info_2pl, 36
compute_info_3pl, 36
compute_se_theta, 50
compute_unidimensional_ability, 57
compute_unidimensional_theta, 58
∗ **utility**
name_triplet_pairs, 114
∗ **wilcoxon**
compute_wilcoxon_effect_size, 59
report_wtests, 185

c_bind, 6, *85*, *118*
call_to_string, 7
cdf, 8, *10, 11*
cdff, 10
cfa_icc_index, 11
change_data_type, 12
check_heywood, 13
clear_stopwords, 14
clear_text, 15
combine_files, 15
comparison_combinations, 16
compute_ability, 17
compute_adjustment, 18
compute_aggregate, 19
compute_alpha_diagnostics, 20
compute_aov_es, 21
compute_cohens_d, 23
compute_confidence_inteval, 25
compute_crosstable, *16*, 25
compute_descriptives, *19*, 26
compute_dissatenuation, 27
compute_dummy_comparisons, 28
compute_efa_cfa, 29
compute_frequencies, 31
compute_friedman_test, 32
compute_icc_thurstonian, 34
compute_info_1pl, 35
compute_info_2pl, 36
compute_info_3pl, 36
compute_kruskal_wallis_test, 37, *178*
compute_kurtosis, 39
compute_map, 40
compute_mean_sd_alpha, 41
compute_moving_average, 41

compute_one_way_test, 42, *178*
compute_posthoc, 45, *178*
compute_power_r, 46
compute_power_r_matrix, 47
compute_raw_alpha, *20*, 48
compute_residual_stats, 48
compute_scores, 49
compute_se_theta, 50
compute_shrout, 50
compute_skewness, 51
compute_solve, 52
compute_standard, 53
compute_standard_error, 55
compute_text_similarity, 55
compute_tversky_index, 56
compute_unidimensional_ability, 57
compute_unidimensional_theta, 58
compute_wilcoxon_effect_size, 59
compute_y_logistic, 60
confusion, 61
confusion_matrix_percent, 62
convert_excel_unix_timestamp, 63

data_frame_index, 63
decompose_datetime, 64
deg2rad, 65
detach_package, 66
df_admission, 66
df_automotive_data, 67
df_blood_pressure, 68
df_co2, 69
df_crop_yield, 70
df_difficile, 70
df_insurance, 71
df_ocean, 72
df_personality, 75
df_responses, 77
df_responses_state, 82
df_sexual_comp, 82
df_titanic, 83
display_upper_lower_triangle, 84, *112*, *211*
dotnames, 85
drop_levels, 86
dummy_arrange, 86

environment_options, 87
excel_confusion_matrix, 88
excel_critical_value, 89
excel_generic_format, *89*, *90*, 91, *93*, *94*
excel_matrix, *90*, 93
extract_components, *50*, 95
extract_tirt_params, 96

fixed, 97
flatten_list, 98

generate_comparisons_matrix, 98
generate_correlation_matrix, 99, *199*
generate_data, *99*, 100
generate_factor, 101
generate_matrix_A, 102
generate_matrix_lambda_hat, 102
generate_missing, 103
generate_multiple_response_vector, *87*, 103
generate_string, 104
generate_unique_comparisons_index, 105
get_fwd, 105
get_script_directory, 106

icc_cfa, 107
increase_index, 107
install_all_packages, 108
install_load, 108

k_fold, 109
k_sample, 110
key_to_cfa_model, 111

matrix_triangle, *85*, 111, *115*, *210*, *211*
min_max_index, 112
model_loadings, 113

name_triplet_pairs, 114

off_diagonal_index, 115
outlier_summary, 116
output_compare_model_logistic, 116
output_separator, 117

padNA, 118
plot_acf, 119
plot_boxplot, 120
plot_cfa, 120
plot_cfa_gg, 121
plot_confusion, 122
plot_confusion_extended, 123
plot_corrplot, 124

plot_crosstable, *16*, 125
plot_duplicate_y_axis, 126
plot_hinvert_title_grob, 127
plot_histogram, 127
plot_icc_thurstonian, 128
plot_interaction, 129
plot_irt_onefactor, 130
plot_loadings, 131
plot_logistic_model, 132
plot_mosaic, 133
plot_mtmm, 134
plot_multiplot, 135
plot_normality_diagnostics, 136
plot_oneway, 137
plot_oneway_diagnostics, 139
plot_outlier, 140
plot_qq, 141
plot_response_frequencies, 141
plot_roc, 142
plot_scatterplot, 143
plot_scree, 145
plot_separability, 145
plot_trees_xgboost, 146
plot_ts, 147
proportion_accurate, 148

questions_by_keys, 149, *150*
questions_dimensions_dataframe, 150

rad2deg, 151
rank3_to_triplets, 153
rank_df_to_binary, 151
rank_to_binary, 152
rbind_all, 153
recode_scale_dummy, 154
remove_nc, 155
remove_outliers, 156
remove_user_packages, 157
replace_na_with_previous, 157
report_alpha, *111*, 158
report_cfa, 160
report_choric_serial, 160
report_correlation, 162
report_dataframe, 163
report_efa, 164
report_factorial_anova, 165
report_hlr, 169
report_irt, 170
report_lda, 170
report_logistic, 171
report_manova, 173
report_normality_tests, 175
report_oneway, 176
report_pdf, 179
report_regression, 180
report_ttests, 182
report_wtests, 185
report_xgboost, 187
response_dimension, 189
response_frequency, 190
result_confusion_performance, 191
round_dataframe, 192

score_tirt, 193
score_tirt_pattern, 195
simulate_cfa_fit, 196
simulate_correlation_from_sample, 198
stat_word_char, 199
str_aes, 200
str_count, 201
str_mgsub, 201
str_pad, 202
str_proper, 203
str_replace, 203
str_replace_all, 204
str_split, 205, *206*
str_split_df, 205
str_split_fixed, 206
str_squish, 207
str_sub, 208
str_trim_df, 208
str_wrap, 209
swap, 210
symmetric_matrix, *85*, *99*, *112*, 210

tag_pos, 211
ts_smoothing, 212

write_txt, 213