Package ‘clinpubr’

June 23, 2025

Title Clinical Publication

Version 1.0.1

Description Accelerate the process from clinical data to medical publication, including clinical data cleaning, significant result screening, and the generation of publish-ready tables and figures.

Maintainer Yue Niu <niuyuesam@163.com>

URL https://github.com/yotasama/clinpubr, https://gitee.com/yotasama/clinpubr

BugReports https://github.com/yotasama/clinpubr/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Collate 'answer_check.r' 'baseline_table.r' 'calculate_index.r' 'check_nonnum.r' 'classif_model_compare.r' 'combine_files.r' 'cut_by.r' 'extract_num.r' 'get_valid.r' 'get_valid_subset.r' 'importance_plot.r' 'interactions.r' 'mark_outlier.r' 'utils.R' 'misc.r' 'rcs_plot.r' 'regressions.r' 'split_multichoice.r' 'subgroup_forest.r' 'subject_standardize.r' 'to_date.r' 'unit_standardize.r' 'value_initial_cleaning.r'

Imports broom, car, caret, dcurves, DescTools, dplyr, fBasics, forestploter, ggplot2, pROC, ResourceSelection, rlang, rms, rstatix, stringi, stringr, survival, survminer, tableone, tidyr

Suggests dtplyr, testthat (>= 3.0.0), vdiffr, withr

Config/testthat/edition 3

NeedsCompilation no

Author Yue Niu [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-6843-3548>), Keyun Wang [aut]

Repository CRAN

Date/Publication 2025-06-23 10:20:14 UTC

Contents

`add_lists` *Adding lists element-wise*

Description

Combine lists by adding element-wise.

Usage

```
add_lists(l1, l2)
```

Arguments

`l1, l2`	A pair of lists.

Value

A list.

Examples

```
l1 <- list(a = 1, b = 2)
l2 <- list(a = 3, b = 4, c = 5)
add_lists(l1, l2)
```

`answer_check` *Check answers of multiple choice questions*

Description

Check answers of multiple choice questions by matching the answers with the correct sequence.

Usage

```
answer_check(dat, seq, multi_column = FALSE)
```

Arguments

`dat`	A data frame of answers.
`seq`	A vector of correct answers, one element for each question.
`multi_column`	Logical, whether the multi-answers are in multiple columns.

Details

If `multi_column` is TRUE, the answers for Multiple-Answer Questions should be in multiple columns of logicals, with each column representing a choice. The `seq` should be a string of `"T"` and `"F"`. If `multi_column` is `FALSE`, the answers for Multiple-Answer Questions should be in one column, and the function would expect an exact match of `seq`.

Value

A data frame of boolean values, with ncol equals the number of questions.

Examples

```
dat <- data.frame(Q1 = c("A", "B", "C"), Q2 = c("AD", "AE", "ABF"))
seq <- c("A", "AE")
answer_check(dat, seq)
dat <- data.frame(
  Q1 = c("A", "B", "C"), Q2.A = c(TRUE, TRUE, FALSE),
  Q2.B = c(TRUE, FALSE, TRUE), Q2.C = c(FALSE, TRUE, FALSE)
)
seq <- c("A", "TFT")
answer_check(dat, seq, multi_column = TRUE)
```

`baseline_table` *Create a baseline table for a dataset.*

Description

Create a baseline table and a table of missing values. If the strata variable has more than 2 levels, a pairwise comparison table will also be created.

Usage

```
baseline_table(
  data,
  var_types = NULL,
  strata = NULL,
  vars = NULL,
  factor_vars = NULL,
  exact_vars = NULL,
  nonnormal_vars = NULL,
  seed = NULL,
  omit_missing_strata = FALSE,
  save_table = FALSE,
  filename = NULL,
  multiple_comparison_test = TRUE,
  p_adjust_method = "BH",
  smd = FALSE,
  ...
)
```

Arguments

`data`	A data frame.
`var_types`	An object from class `var_types` returned by `get_var_types` function.
`strata`	A variable to stratify the table. Overwrites the strata variable in `var_types`.
`vars`	A vector of variables to include in the table.
`factor_vars`	A vector of factor variables. Overwrites the factor variables in `var_types`.
`exact_vars`	A vector of variables to test for exactness. Overwrites the exact variables in `var_types`.
`nonnormal_vars`	A vector of variables to test for normality. Overwrites the nonnormal variables in `var_types`.
`seed`	A seed for the random number generator. This seed can be set for consistent simulation when performing fisher exact tests.
`omit_missing_strata`	A logical value indicating whether to omit missing values in the strata variable.
`save_table`	A logical value indicating whether to save the result tables.
`filename`	The name of the file to save the table. The file names for accompanying tables will be the same as the main table, but with "_missing" and "_pairwise" appended.
`multiple_comparison_test`	A logical value indicating whether to perform multiple comparison tests. Variables in `factor_vars` and `exact_vars` are tested with pairwise `chisq.test` or `fisher.test`, and other variables are tested with `rstatix::dunn_test` or `rstatix::games_howell_test`.
`p_adjust_method`	The method to use for p-value adjustment for pairwise comparison. Default is "BH". See `?p.adjust.methods`.
`smd`	A logical value indicating whether to include SMD in the table. Passed to `tableone::print.TableOne`.
`...`	Additional arguments passed to `tableone::print.TableOne`.

Value

A list containing the baseline table and accompanying tables.

Examples

```
withr::with_tempdir(
  {
    data(cancer, package = "survival")
    var_types <- get_var_types(cancer, strata = "sex")
    baseline_table(cancer, var_types = var_types, filename = "baseline.csv")

    # baseline table with pairwise comparison
    cancer$ph.ecog_cat <- factor(cancer$ph.ecog,
      levels = c(0:3),
      labels = c("0", "1", ">=2", ">=2")
    )
    var_types <- get_var_types(cancer, strata = "ph.ecog_cat")
    baseline_table(cancer, var_types = var_types, filename = "baselineV2.csv")
    print(paste0("files saved to: ", getwd()))
  },
  clean = FALSE
)
```

`break_at` *Generate breaks for histogram*

Description

Generate breaks for histogram that covers xlim and includes a ref_val.

Usage

```
break_at(xlim, breaks, ref_val = NULL)
```

Arguments

`xlim`	A vector of length 2.
`breaks`	The number of breaks.
`ref_val`	The reference value to include in breaks.

Value

A vector of breaks of length `breaks + 1`.

Examples

```
break_at(xlim = c(0, 10), breaks = 12, ref_val = 3.12)
```

`calculate_index` *Calculate index based on conditions*

Description

Calculate an index based on multiple conditions. Each condition is evaluated and the result is weighted and summed to produce the final index.

Usage

```
calculate_index(.df, ..., .weight = 1, .na_replace = 0)
```

Arguments

`.df`	A data frame
`...`	Conditions to evaluate. See examples for more details.
`.weight`	Weight for each condition, should be of length 1 or equal to the number of conditions.
`.na_replace`	Value to replace `NA`, should be of length 1 or equal to the number of conditions.

Value

A numeric vector of index scores

Examples

```
df <- data.frame(x = c(1, 2, 3, 4, 5), y = c(1, 2, NA, 4, NA))
calculate_index(df, x > 3, y < 3, .weight = c(1, 2), .na_replace = 0)
```

`check_nonnum` *Check elements that are not numeric*

Description

Finds the elements that cannot be converted to numeric in a character vector. Useful when setting the strategy to clean numeric values.

Usage

```
check_nonnum(
  x,
  return_idx = FALSE,
  show_unique = TRUE,
  max_count = NULL,
  random_sample = FALSE,
  fix_len = FALSE
)
```

Arguments

`x`	A string vector that stores numerical values.
`return_idx`	A logical value. If `TRUE`, return the index of the elements that are not numeric.
`show_unique`	A logical value. If `TRUE`, return the unique elements that are not numeric. Omitted if `return_idx` is `TRUE`.
`max_count`	An integer. The maximum number of elements to show. If `NULL` or `0`, show all elements. Omitted if `return_idx` is `TRUE`.
`random_sample`	A logical value. If `TRUE`, randomly sample the elements to show. Only works if `max_count` is not `NULL` or `0`.
`fix_len`	A logical value. If `TRUE`, fill the vector with `NA` to fix the length to `max_count`.

Details

The function uses the `as.numeric()` function to try to convert the elements to numeric. If the conversion fails, the element is considered non-numeric.

Value

The (unique) elements that cannot be converted to numeric, and their indexes if `return_idx` is `TRUE`.

Examples

```
check_nonnum(c("\uFF11\uFF12\uFF13", "11..23", "3.14", "2.131", "35.2."))
```

`classif_model_compare` *Performance comparison of classification models*

Description

Compare the performance of classification models by commonly used metrics, and generate commonly used plots including receiver operating characteristic curve plot, decision curve analysis plot, and calibration plot.

Usage

```
classif_model_compare(
  data,
  target_var,
  model_names,
  colors = NULL,
  save_output = FALSE,
  figure_type = "png",
  output_prefix = "model_compare",
  as_probability = FALSE,
  auto_order = TRUE
)
```

Arguments

`data`	A data frame containing the target variable and the predicted values.
`target_var`	A string specifying the name of the target variable in the data frame.
`model_names`	A vector of strings specifying the names of the models to compare.
`colors`	A vector of colors to use for the plots. The last 2 colors are used for the "Treat all" and "Treat none" lines in the DCA plot.
`save_output`	A logical value indicating whether to output the results to files.
`figure_type`	A character string of the figure type. Can be `"png"`, `"pdf"`, and other types that `ggplot2::ggsave()` support.

`output_prefix`	A string specifying the prefix for the output files.
`as_probability`	A logical or a vector of variable names. The logical value indicates whether to convert variables not in range 0 to 1 into this range. The vector of variable names means to convert these variables to the range of 0 to 1.
`auto_order`	A logical value indicating whether to automatically order the models by their AUCs. If `TRUE`, the models will be ordered by their AUCs in descending order. If `FALSE`, the order in `model_names` will be retained.

Value

A list of various results. If the output files are not in desired format, these results can be modified for further use.

- metric_table: A data frame containing the performance metrics for each model.
- roc_plot: A `ggplot` object of ROC curves.
- dca_plot: A `ggplot` object of decision curve analysis plots.
- calibration_plot: A `ggplot` object of calibration plots.

Metrics

- AUC: Area Under the Receiver Operating Characteristic Curve
- Accuracy: Overall accuracy
- Sensitivity: True positive rate
- Specificity: True negative rate
- Pos Pred Value: Positive predictive value
- Neg Pred Value: Negative predictive value
- F1: F1 score
- Kappa: Cohen's kappa
- Brier: Brier score
- cutoff: Optimal cutoff for classification, metrics that require a cutoff are based on this value.
- Youden: Youden's J statistic
- HosLem: Hosmer-Lemeshow test p-value

Examples

```
data(cancer, package = "survival")
df <- kidney
df$dead <- ifelse(df$time <= 100 & df$status == 0, NA, df$time <= 100)
df <- na.omit(df[, -c(1:3)])

model0 <- glm(dead ~ age + frail, family = binomial(), data = df)
model <- glm(dead ~ ., family = binomial(), data = df)
df$base_pred <- predict(model0, type = "response")
df$full_pred <- predict(model, type = "response")

classif_model_compare(df, "dead", c("base_pred", "full_pred"), save_output = FALSE)
```

combine_files *combine multiple data files into a single data frame*

Description

combine multiple data files into a single data frame

Usage

```
combine_files(
  path = ".",
  pattern = NULL,
  add_file_name = FALSE,
  unique_only = TRUE,
  reader_fun = read.csv,
  ...
)
```

Arguments

path	A string as the path to find the data files.
pattern	A file pattern to filter the required data files.
add_file_name	A logical value to indicate whether to add the file name as a column. Note that the added file name will affect the uniqueness of the data.
unique_only	A logical value to indicate whether to remove the duplicated rows.
reader_fun	A function to read the data files. Can be `read.csv`, `openxlsx::read.xlsx`, etc.
...	Other parameters passed to the `reader_fun`.

Value

A data frame. If no data files found, return `NULL`.

Examples

```
library(withr)
with_tempdir({
  write.csv(data.frame(x = 1:3, y = 4:6), "file1.csv", row.names = FALSE)
  write.csv(data.frame(x = 7:9, y = 10:12), "file2.csv", row.names = FALSE)
  dat <- combine_files(pattern = "file")
})
print(dat)
```

`cut_by` *Convert Numeric to Factor*

Description

Divide numeric data into different groups. Easier to use than `base::cut()`.

Usage

```
cut_by(
  x,
  breaks,
  breaks_as_quantiles = FALSE,
  labels = NULL,
  label_type = "ori",
  ...
)
```

Arguments

`x`	A numeric vector.
`breaks`	A numeric vector of internal cut points. If `breaks_as_quantiles` is TRUE, this is a proportion of the data. See `Details`.
`breaks_as_quantiles`	If TRUE, `breaks` is interpreted as a proportion of the data.
`labels`	A vector of labels for the resulting factor levels.
`label_type`	If `labels` is `NULL`, this sets the label type. `"ori"` for original labels, `"LMH"` for "Low Medium High" style. `"combined"` labels that combine `"LMH"` type or provided `labels` with the original range labels.
`...`	Other arguments passed to `base::cut()`.

Details

`cut_by()` is a wrapper for `base::cut()`. Compared with the argument `breaks` in `base::cut()`, `breaks` here automatically sets the minimum and maximum to `-Inf` and `Inf`.

Value

A factor.

Note

The argument `right` in `base::cut()` is always set to `FALSE`, which means the levels follow the left closed right open convention.

Examples

```
set.seed(123)
cut_by(rnorm(100), c(0, 1, 2))
cut_by(rnorm(100), c(1 / 3, 2 / 3), breaks_as_quantiles = TRUE, label_type = "LMH")
```

---

df_view_nonnum *Show non-numeric elements in a data frame*

Description

Shows the non-numeric elements in a data frame. Useful when setting the strategy to clean numeric values.

Usage

```
df_view_nonnum(
  df,
  max_count = 20,
  random_sample = FALSE,
  long_df = FALSE,
  subject_col = NULL,
  value_col = NULL
)
```

Arguments

df	A data frame.
max_count	An integer. The maximum number of elements to show for each column. If `NULL` or `0`, show all elements, not recommended due to huge memory waste.
random_sample	A logical value. If `TRUE`, randomly sample the elements to show.
long_df	A logical value. If `TRUE`, the input `df` is provided in a long format.
subject_col	A character string. The name of the column that contains the subject identifier. Used when `long_df` is `TRUE`. If `NULL`, the subject column is assumed to be the first column.
value_col	A character string. The name of the column that contains the values. Used when `long_df` is `TRUE`. If `NULL`, the value column is assumed to be the second column.

Value

A data frame of the non-numeric elements.

Examples

```
df <- data.frame(
  x = c("1", "2", "3..3", "4", "6a"),
  y = c("1", "ss", "aa.a", "4", "xx"),
  z = c("1", "2", "3", "4", "6")
)
df_view_nonnum(df)
```

`emp_colors` *default color palette for* `clinpubr` *plots*

Description

default color palette for `clinpubr` plots

Usage

```
emp_colors
```

Format

An object of class `character` of length 10.

`extract_num` *Extract numbers from string.*

Description

Extract numerical values from strings. Can be used to filter out the unwanted information coming along with the numbers.

Usage

```
extract_num(
  x,
  res_type = c("first", "range"),
  multimatch2na = FALSE,
  leq_1 = FALSE,
  allow_neg = TRUE,
  zero_regexp = NULL,
  max_regexp = NULL,
  max_quantile = 0.95
)
```

Arguments

`x`	A character vector.
`res_type`	The type of the result. Can be `"first"` or `"range"`. If `"first"`, the first number in the string is extracted. If `"range"`, the mean of the range in the string is extracted.
`multimatch2na`	If `TRUE`, multiple matches will be converted to `NA`. Only works when `res_type` is `"first"`.
`leq_1`	If `TRUE`, numbers greater than 1 will be converted to `NA`. Only works when `res_type` is `"first"`.
`allow_neg`	If `TRUE`, negative numbers are allowed. Otherwise, only positive numbers are allowed.
`zero_regexp`	A regular expression to match the string that indicates zero.
`max_regexp`	A regular expression to match the string that indicates the maximum value.
`max_quantile`	The quantile of values to set the maximum value to.

Details

The function uses regular expressions to extract numbers from strings. The regular expression used is `"-?[0-9]+\\.?[0-9]*|-?\\.[0-9]+"`, which matches any number that may have a decimal point and may have a negative sign.

Value

A numeric vector.

Examples

```
x <- c("1.2(XXX)", "5-8POS", "NS", "FULL", "5.5", "4.2")
extract_num(x)
extract_num(x,
  res_type = "first", multimatch2na = TRUE, zero_regexp = "NEG|NS",
  max_regexp = "FULL"
)
extract_num(x, res_type = "range", allow_neg = FALSE, zero_regexp = "NEG|NS", max_regexp = "FULL")
```

`fill_with_last` *Fill NA values with the last valid value*

Description

Fill NA values with the last valid value. Can be used to fill excel combined cells.

Usage

```
fill_with_last(x)
```

Arguments

`x`	A vector.

Value

A vector.

Examples

```
fill_with_last(c(1, 2, NA, 4, NA, 6))
```

`filter_rcs_predictors` *Filter predictors for RCS*

Description

Filter predictors that can be used to fit for RCS models.

Usage

```
filter_rcs_predictors(data, predictors = NULL)
```

Arguments

`data`	A data frame.
`predictors`	A vector of predictor names to be filtered.

Value

A vector of predictor names. These variables are numeric and have more than 5 unique values.

Examples

```
filter_rcs_predictors(mtcars)
```

`first_mode` *Calculate the first mode*

Description

Calculate the first mode of a vector. Ignore NA values. Can be used if any mode is acceptable.

Usage

```
first_mode(x, empty_return = NA)
```

Arguments

`x`	A vector.
`empty_return`	The value to return if the vector is empty.

Value

The first mode of the vector.

Examples

```
first_mode(c(1, 1, 2, 2, 3, 3, 3, NA, NA, NA))
```

`format_pval` *Format p-value for publication*

Description

Format p-value with modified default settings suitable for publication.

Usage

```
format_pval(
  p,
  text_ahead = NULL,
  digits = 1,
  nsmall = 2,
  eps = 0.001,
  na_empty = TRUE
)
```

Arguments

`p`	The numerical p values to be formatted.
`text_ahead`	A string to be added before the p value. If not `NULL`, this string will be connected to the formatted p value with `"="` or `"<"`.
`digits`	The number of digits to be used. Same as in `base::format.pval`.
`nsmall`	The number of digits after the decimal point. Same as in `base::format.pval`.
`eps`	The threshold for rounding p values to 0. Same as in `base::format.pval`.
`na_empty`	If `TRUE`, replace `"NA"` in result with an empty string.

Value

A string vector of formatted p values.

Examples

```
format_pval(c(0.001, 0.0001, 0.05, 0.1123456))
format_pval(c(0.001, 0.0001, 0.05, 0.1123456), text_ahead = "p value")
```

`formula_add_covs` *Add covariates to a formula*

Description

Add covariates to a formula. Support both formula and character string.

Usage

```
formula_add_covs(formula, covars)
```

Arguments

`formula`	A formula. Should be a formula or a character string of formula.
`covars`	A vector of covariates.

Value

A formula.

Examples

```
formula_add_covs("y ~ a + b", c("c", "d"))
```

`get_samples` *Generate a sample of values from a vector and collapse them.*

Description

Generate a string summary of a vector by picking samples.

Usage

```
get_samples(x, unique_only = FALSE, n_samples = 10, collapse = "\n")
```

Arguments

`x`	A vector of values.
`unique_only`	A logical value indicating whether to return unique values only.
`n_samples`	The number of samples to return.
`collapse`	The separator to use for collapsing the values.

Value

A character string.

Examples

```
get_samples(c(1, 2, 3, 4, 5))
get_samples(c(1, 2, 3, 4, 5), n_samples = 2)
get_samples(c(1, 2, 3, 3, 3), n_samples = 2, unique_only = TRUE)
get_samples(c(1, 2, 3, 4, 5), collapse = ", ")
```

`get_valid` *Get one valid value from vector.*

Description

Extract one valid (non-NA) value from a vector.

Usage

```
get_valid(x, mode = c("first", "mid", "last"), disjoint = FALSE)
```

Arguments

`x`	A vector.
`mode`	The mode of the valid value to extract. `"first"` extracts the first valid value, `"last"` extracts the last valid value, and `"mid"` extracts the middle valid value.
`disjoint`	If TRUE, the values extracted by the three modes are forced to be different. This behavior might be desired when trying to extract different values with different modes. The three modes extract values in the sequence: "first", "last", "mid".

Value

A single valid value from the vector. `NA` if all values are invalid.

Examples

```
get_valid(c(NA, 1, 2, NA, 3, NA, 4))
get_valid(c(NA, 1, NA), mode = "last", disjoint = TRUE)
```

`get_valid_subset` *Get the subset that satisfies the missing rate condition.*

Description

Get the subset of a data frame that satisfies the missing rate condition using a greedy algorithm.

Usage

```
get_valid_subset(
  df,
  row_na_ratio = 0.5,
  col_na_ratio = 0.2,
  row_priority = 1,
  speedup_ratio = 0,
  return_index = FALSE
)
```

Arguments

`df`	A data frame.
`row_na_ratio`	The maximum acceptable missing rate of rows.
`col_na_ratio`	The maximum acceptable missing rate of columns.
`row_priority`	A positive numerical, the priority to keep rows. The higher the value, the higher the priority, with `1` indicating equal priority for rows and columns.
`speedup_ratio`	A positive numerical, the ratio of speedup. The higher the value, the greedier the algorithm.
`return_index`	A logical, whether to return only the row and column indices of the subset.

Details

The function is based on a greedy algorithm. It iteratively removes the row or column with the highest excessive missing rate weighted by the inverse of `row_priority` until the missing rates of all rows and columns are below the specified threshold. Then it reversely tries to add rows and columns that do not break the conditions back and finalize the subset. The result depends on the `row_priority` parameter drastically, so it's recommended to try different `row_priority` values to find the most satisfying one.

Value

The subset data frame, or a list that contains the row and column indices of the subset.

Examples

```
data(cancer, package = "survival")
dim(cancer)
max_missing_rates(cancer)

cancer_valid <- get_valid_subset(cancer, row_na_ratio = 0.2, col_na_ratio = 0.1, row_priority = 1)
dim(cancer_valid)
max_missing_rates(cancer_valid)
```

`get_var_types` *Get variable types for baseline table*

Description

Automatic variable type and method determination for baseline table.

Usage

```
get_var_types(
  data,
  strata = NULL,
  norm_test_by_group = TRUE,
  omit_factor_above = 20,
  num_to_factor = 5,
  save_qqplots = FALSE,
  folder_name = "qqplots"
)
```

Arguments

`data`	A data frame.
`strata`	A character string indicating the column name of the strata variable.
`norm_test_by_group`	A logical value indicating whether to perform normality tests by group.

`omit_factor_above`	An integer indicating the maximum number of levels for a variable to be considered a factor.
`num_to_factor`	An integer. Numerical variables with number of unique values below or equal to this value would be considered a factor.
`save_qqplots`	A logical value indicating whether to save QQ plots. Sometimes the normality tests do not work well for some variables, and the QQ plots can be used to check the distribution.
`folder_name`	A character string indicating the folder name for saving QQ plots.

Value

An object from class `var_types`, which is just list containing the following elements:

`factor_vars`	A character vector of variables that are factors.
`exact_vars`	A character vector of variables that require fisher exact test.
`nonnormal_vars`	A character vector of variables that are nonnormal.
`omit_vars`	A character vector of variables that are excluded form the baseline table.
`strata`	A character vector of the strata variable.

Note

This function performs normality tests on the variables in the data frame and determines whether they are normal. This is done by performing Shapiro-Wilk, Lilliefors, Anderson-Darling, Jarque-Bera, and Shapiro-Francia tests. If at least two of these tests indicate that the variable is nonnormal, then it is considered nonnormal. To alleviate the problem that normality tests become too sensitive when sample size gets larger, the alpha level is determined by an experience formula that decrease with sample size.

This function also marks the factor variables that require fisher exact tests if any cell haves expected frequency less than or equal to 5. Note that this criterion less strict than the commonly used one.

Examples

```
data(cancer, package = "survival")
get_var_types(cancer, strata = "sex") # set save_qqplots = TRUE to check the QQ plots

var_types <- get_var_types(cancer, strata = "sex")
# for some reason we want the variable "pat.karno" ro be considered normal.
var_types$nonnormal_vars <- setdiff(var_types$nonnormal_vars, "pat.karno")
```

`importance_plot` *Importance plot*

Description

Creates an importance plot from a named vector of values.

Usage

```
importance_plot(
  x,
  x_lab = "Importance",
  top_n = NULL,
  color = c("#56B1F7", "#132B43"),
  show_legend = FALSE,
  split_at = NULL,
  show_labels = TRUE,
  digits = 2,
  nsmall = 3,
  scientific = TRUE,
  label_color = "black",
  label_size = 3,
  label_hjust = max(x)/10,
  save_plot = FALSE,
  filename = "importance.png"
)
```

Arguments

`x`	A named vector of values, typically importance scores from models.
`x_lab`	A character string for the x-axis label.
`top_n`	The number of top values to show. If NULL, all values are shown.
`color`	A length-2 vector of low and high colors, or a single color for the bars.
`show_legend`	A logical value indicating whether to show the legend.
`split_at`	The index at which to split the plot into two halves, usually used to illustrate variable selection. If NULL, no split is made.
`show_labels`	A logical value indicating whether to show the value labels on the bars.
`digits`, `nsmall`, `scientific`	Controls the formatting of labels. Passed to `format()`.
`label_color`	The color of the labels.
`label_size`	The size of the labels.
`label_hjust`	The horizontal justification of the labels.
`save_plot`	A logical value indicating whether to save the plot.
`filename`	The filename to save the plot as.

Details

The importance plot is a bar plot that shows the importance of each variable in a model. The variables are sorted in descending order of importance, and the top_n variables are shown. If top_n is NULL, all variables are shown. The plot can be split into two halves at a specified index, which is useful for illustrating variable selection.

Value

A `ggplot` object

Examples

```
set.seed(1)
dummy_importance <- runif(20)^5
names(dummy_importance) <- paste0("var", 1:20)
importance_plot(dummy_importance, top_n = 15, split_at = 10, save_plot = FALSE)
```

`interaction_plot` *Plot interactions*

Description

Plot interactions between variables. Both logistic and Cox proportional hazards regression models are supported. The predictor variables in the model are can be used both in linear form or in restricted cubic spline form.

Usage

```
interaction_plot(
  data,
  y,
  predictor,
  group_var,
  time = NULL,
  covars = NULL,
  group_colors = NULL,
  save_plot = FALSE,
  filename = NULL,
  height = 4,
  width = 4,
  xlab = predictor,
  ylab = NULL,
  show_n = TRUE,
  group_title = group_var,
  ...
)
```

Arguments

`data`	A data frame.
`y`	A character string of the outcome variable.
`predictor`	A character string of the predictor variable.
`group_var`	A character string of the group variable. The variable should be categorical. If a numeric variable is provided, it will be split by the median value.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`covars`	A character vector of covariate names.
`group_colors`	A character vector of colors for the plot. If `NULL`, the default colors are used.
`save_plot`	A logical value indicating whether to save the plot.
`filename`	The name of the file to save the plot. Support both `.png` and `.pdf` formats.
`height`	The height of the saved plot.
`width`	The width of the saved plot.
`xlab`	The label of the x-axis.
`ylab`	The label of the y-axis.
`show_n`	A logical value indicating whether to show the number of observations in the plot.
`group_title`	The title of the group variable.
`...`	Additional arguments passed to the `ggplot` function.

Value

A `ggplot` object.

Examples

```
data(cancer, package = "survival")
interaction_plot(cancer,
  y = "status", time = "time", predictor = "age", group_var = "sex",
  save_plot = FALSE
)
interaction_plot(cancer,
  y = "status", predictor = "age", group_var = "sex",
  save_plot = FALSE
)
interaction_plot(cancer,
  y = "wt.loss", predictor = "age", group_var = "sex",
  save_plot = FALSE
)
```

`interaction_p_value` *Calculate interaction p-value*

Description

This function calculates the interaction p-value between a predictor and a group variable in a linear, logistic, or Cox proportional hazards model.

Usage

```
interaction_p_value(
  data,
  y,
  predictor,
  group_var,
  time = NULL,
  covars = NULL,
  rcs_knots = NULL
)
```

Arguments

`data`	A data frame.
`y`	A character string of the outcome variable. The variable should be binary or numeric and determines the type of model to be used. If the variable is binary, logistic or Cox regression is used. If the variable is numeric, linear regression is used.
`predictor`	A character string of the predictor variable.
`group_var`	A character string of the group variable. The variable should be categorical. If a numeric variable is provided, it will be split by the median value.
`time`	A character string of the time variable. If `NULL`, linear or logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`covars`	A character vector of covariate names.
`rcs_knots`	The number of rcs knots. If `NULL`, a linear model would be fitted instead.

Value

A numerical, the interaction p-value

Examples

```
data(cancer, package = "survival")
interaction_p_value(
  data = cancer, y = "status", predictor = "age", group_var = "sex",
  time = "time", rcs_knots = 4
)
```

interaction_scan *Scan for interactions between variables*

Description

Scan for interactions between variables and output results. Both logistic and Cox proportional hazards regression models are supported. The predictor variables in the model are can be used both in linear form or in restricted cubic spline form.

Usage

```
interaction_scan(
  data,
  y,
  time = NULL,
  predictors = NULL,
  group_vars = NULL,
  covars = NULL,
  try_rcs = TRUE,
  p_adjust_method = "BH",
  save_table = FALSE,
  filename = NULL
)
```

Arguments

`data`	A data frame.
`y`	A character string of the outcome variable.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`predictors`	The predictor variables to be scanned for interactions. If `NULL`, all variables except `y` and `time` are taken as predictors.
`group_vars`	The group variables to be scanned for interactions. If `NULL`, all variables except `y` and `time` are taken as group variables. The group variables should be categorical. If a numeric variable is included, it will be split by the median value.
`covars`	A character vector of covariate names.
`try_rcs`	A logical value indicating whether to perform restricted cubic spline interaction analysis.
`p_adjust_method`	The method to use for p-value adjustment for pairwise comparison. Default is "BH". See `?p.adjust.methods`.
`save_table`	A logical value indicating whether to save the results as a table.
`filename`	The name of the file to save the results. File will be saved in `.csv` format.

Value

A data frame containing the results of the interaction analysis.

Examples

```
data(cancer, package = "survival")
interaction_scan(cancer, y = "status", time = "time", save_table = FALSE)
```

---

mad_outlier *Mark possible outliers with MAD.*

Description

Mark possible outliers using the median absolute deviation (MAD) method.

Usage

```
mad_outlier(x, constant = 1.4826 * 3)
```

Arguments

`x`	A numeric vector.
`constant`	The constant multiplier for the MAD. Default is 1.4826 * 3.

Details

The function calculates the median absolute deviation of the input vector and uses it to identify possible outliers. The default constant multiplier is 1.4826 * 3, which gives approximately the 3σ of the normal distribution.

Value

A logical vector.

See Also

`mad`

Examples

```
x <- c(1, 2, 3, 4, 5, 100)
mad_outlier(x)
```

`max_missing_rates` *Get the maximum missing rate of rows and columns.*

Description

Get the maximum missing rate of rows and columns.

Usage

```
max_missing_rates(df)
```

Arguments

`df`	A data frame.

Value

A list that contains the maximum missing rate of rows and columns.

Examples

```
data(cancer, package = "survival")
max_missing_rates(cancer)
```

`merge_ordered_vectors` *Merging vectors while maintaining order*

Description

Merge multiple vectors into one while trying to maintain the order of elements in each vector. The relative order of elements is compared by their first occurrence in the vectors in the list. This function is useful when merging slightly different vectors, such as questionnaires of different versions.

Usage

```
merge_ordered_vectors(vectors)
```

Arguments

`vectors`	A list of vectors to be merged.

Value

A vector.

Examples

```
merge_ordered_vectors(list(c(1, 3, 4, 5, 7, 10), c(2, 5, 6, 7, 8), c(1, 7, 5, 10)))
```

`na2false` *Replace NA values with FALSE*

Description

Replace `NA` values with `FALSE` in logical vectors. For other vectors, the behavior relies on R's automatic conversion rules.

Usage

```
na2false(x)
```

Arguments

`x`	A vector.

Value

A vector with `NA` values replaced by `FALSE`.

Examples

```
na2false(c(TRUE, FALSE, NA, TRUE, NA))
na2false(c(1, 2, NA))
```

`qq_show` *QQ plot*

Description

QQ plot for a sample.

Usage

```
qq_show(
  x,
  title = NULL,
  save = FALSE,
  filename = "QQplot.png",
  width = 2,
  height = 2
)
```

Arguments

`x`	A sample.
`title`	Title of the plot.
`save`	If TRUE, save the plot.
`filename`	Filename of the plot.
`width`	Width of the plot.
`height`	Height of the plot.

Value

A plot.

Examples

```
qq_show(rnorm(100))
```

`rcs_plot` *Plot restricted cubic spline*

Description

Plot restricted cubic spline based on package `rms`. Support both logistic and cox model.

Usage

```
rcs_plot(
  data,
  x,
  y,
  time = NULL,
  covars = NULL,
  knot = 4,
  add_hist = TRUE,
  ref = "x_median",
  ref_digits = 3,
  group_by_ref = TRUE,
  group_title = NULL,
  group_labels = NULL,
  group_colors = NULL,
  breaks = 20,
  rcs_color = "#e23e57",
  print_p_ph = TRUE,
  trans = "identity",
  save_plot = FALSE,
  filename = NULL,
  y_lim = NULL,
  hist_max = NULL,
  xlim = NULL,
  height = 6,
  width = 6,
  return_details = FALSE
)
```

Arguments

data	A data frame.
x	A character string of the predictor variable.
y	A character string of the outcome variable.
time	A character string of the time variable. If NULL, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
covars	A character vector of covariate names.
knot	The number of knots. If NULL, the number of knots is determined by AIC minimum.
add_hist	A logical value. If TRUE, add histogram to the plot.
ref	The reference value for the plot. Could be "x_median", "x_mean", "ratio_min", or a numeric value. If "x_median", the median of the predictor variable is used. If "ratio_min", the value of the predictor variable that has the minium predicted risk is used. If a numeric value, that value is used.
ref_digits	The number of digits for the reference value.
group_by_ref	A logical value. If TRUE, split the histogram at the reference value from ref into two groups.
group_title	A character string of the title for the group. Ignored if group_by_ref is FALSE.
group_labels	A character vector of the labels for the group. If NULL, the labels are generated automatically. Ignored if group_by_ref is FALSE.
group_colors	A character vector of colors for the plot. If NULL, the default colors are used. If group_by_ref is FALSE, the first color is used as fill color.
breaks	The number of breaks for the histogram.
rcs_color	The color for the restricted cubic spline.
print_p_ph	A logical value. If TRUE, print the p-value of the proportional hazards test (survival::cox.zph()) in the plot.
trans	The transformation for the y axis in the plot. Passed to ggplot2::scale_y_continuous(transform = trans).
save_plot	A logical value indicating whether to save the plot.
filename	A character string specifying the filename for the plot. If NULL, a default filename is used.
y_lim	The range of effect value of the plot. If NULL, the numbers are determined automatically.

hist_max	The maximum value for the histogram. If NULL, the maximum value is determined automatically.
xlim	The x-axis limits for the plot. If NULL, the limits are the 0.025 and 0.975 quantiles. The actual plot range might be slightly larger than this range to fit the histogram.
height	The height of the saved plot.
width	The width of the saved plot.
return_details	A logical value indicating whether to return the details of the plot.

Value

A ggplot object, or a list containing the ggplot object and other details if return_details is TRUE.

Examples

```
data(cancer, package = "survival")
# coxph model with time assigned
rcs_plot(cancer, x = "age", y = "status", time = "time", covars = "ph.karno", save_plot = FALSE)

# logistic model with time not assigned
cancer$dead <- cancer$status == 2
rcs_plot(cancer, x = "age", y = "dead", covars = "ph.karno", save_plot = FALSE)
```

---

regression_basic_results
Basic results of logistic or Cox regression.

Description

Generate the result table of logistic or Cox regression with different settings of the predictor variable and covariates. Also generate KM curves for Cox regression.

Usage

```
regression_basic_results(
  data,
  x,
  y,
  time = NULL,
  model_covs = NULL,
  pers = c(0.1, 10, 100),
  factor_breaks = NULL,
  factor_labels = NULL,
  quantile_breaks = NULL,
  quantile_labels = NULL,
  label_with_range = FALSE,
  save_output = FALSE,
  figure_type = "png",
  ref_levels = "lowest",
  est_nsmall = 2,
  p_nsmall = 3,
  pval_eps = 0.001,
  median_nsmall = 0,
  colors = NULL,
  xlab = NULL,
  legend_title = x,
  legend_pos = c(0.8, 0.8),
  pval_pos = NULL,
  n_y_pos = 0.9,
  height = 6,
  width = 6,
  ...
)
```

Arguments

`data`	A data frame.
`x`	A character string of the predictor variable.
`y`	A character string of the outcome variable.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`model_covs`	A character vector or a named list of covariates for different models. If `NULL`, only the crude model is used.
`pers`	A numeric vector of the denominators of variable `x`. Set this denominator to obtain a reasonable OR or HR.
`factor_breaks`	A numeric vector of the breaks to factorize the `x` variable.
`factor_labels`	A character vector of the labels for the factor levels.
`quantile_breaks`	A numeric vector of the quantile breaks to factorize the `x` variable.
`quantile_labels`	A character vector of the labels for the quantile levels.
`label_with_range`	A logical value indicating whether to add the range of the levels to the labels.
`save_output`	A logical value indicating whether to save the results.
`figure_type`	A character string of the figure type. Can be `"png"`, `"pdf"`, and other types that `ggplot2::ggsave()` support.
`ref_levels`	A vector of strings of the reference levels of the factor variable. You can use `"lowest"` or `"highest"` to select the lowest or highest level as the reference level. Otherwise, any level that matches the provided strings will be used as the reference level.
`est_nsmall`	An integer specifying the precision for the estimates in the plot.

p_nsmall	An integer specifying the number of decimal places for the p-values.
pval_eps	The threshold for rounding p values to 0.
median_nsmall	The minimum number of digits to the right of the decimal point for the median survival time.
colors	A vector of colors for the KM curves.
xlab	A character string of the x-axis label.
legend_title	A character string of the title of the legend.
legend_pos	A numeric vector of the position of the legend.
pval_pos	A numeric vector of the position of the p-value.
n_y_pos	A numerical of range 0 to 1 to assign the y position of total sample count. `NULL` to hide.
height	The height of the plot.
width	The width of the plot.
...	Additional arguments passed to the `survminer::ggsurvplot` function for KM curve.

Details

The function `regression_basic_results` generates the result table of logistic or Cox regression with different settings of the predictor variable and covariates. The setting of the predictor variable includes the original `x`, the standardized `x`, the log of `x`, and `x` divided by denominators in `pers` as continuous variables, and the factorization of the variable including split by median, by quartiles, and by `factor_breaks` and `quantile_breaks`. The setting of the covariates includes different models with different covariates.

Value

A list of results, including the regression table and the KM curve plots.

Note

For factor variables with more than 2 levels, p value for trend is also calculated.

Examples

```
data(cancer, package = "survival")
# coxph model with time assigned
regression_basic_results(cancer,
  x = "age", y = "status", time = "time",
  model_covs = list(Crude = c(), Model1 = c("ph.karno"), Model2 = c("ph.karno", "sex")),
  save_output = FALSE,
 ggtheme = survminer::theme_survminer(font.legend = c(14, "plain", "black")) # theme for KM
)

# logistic model with time not assigned
cancer$dead <- cancer$status == 2
regression_basic_results(cancer,
  x = "age", y = "dead", ref_levels = c("Q3", "High"),
  model_covs = list(Crude = c(), Model1 = c("ph.karno"), Model2 = c("ph.karno", "sex")),
  save_output = FALSE
)
```

`regression_fit` *Obtain regression results*

Description

This function fit the regression of a predictor in a linear, logistic, or Cox proportional hazards model.

Usage

```
regression_fit(
  data,
  y,
  predictor,
  time = NULL,
  covars = NULL,
  rcs_knots = NULL,
  returned = c("full", "predictor_split", "predictor_combined")
)
```

Arguments

`data`	A data frame.
`y`	A character string of the outcome variable. The variable should be binary or numeric and determines the type of model to be used. If the variable is binary, logistic or cox regression is used. If the variable is numeric, linear regression is used.
`predictor`	A character string of the predictor variable.
`time`	A character string of the time variable. If `NULL`, linear or logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`covars`	A character vector of covariate names.
`rcs_knots`	The number of rcs knots. If `NULL`, a linear model would be fitted instead.
`returned`	The return mode of this function. • `"full"`: return the full regression result. • `"predictor_split"`: return the regression parameter of the predictor, could have multiple lines. • `"predictor_combined"`: return the regression parameter of the predictor, test the predictor as a whole and takes only one line.

Value

A list containing the regression ratio and p-value of the predictor. If `rcs_knots` is not `NULL`, the list contains the overall p-value and the nonlinear p-value of the rcs model. If `return_full_result` is `TRUE`, the complete result of the regression model is returned.

Examples

```
data(cancer, package = "survival")
regression_fit(data = cancer, y = "status", predictor = "age", time = "time", rcs_knots = 4)
```

`regression_forest` *Forest plot of regression results*

Description

Generate the forest plot of logistic or Cox regression with different models.

Usage

```
regression_forest(
  data,
  model_vars,
  y,
  time = NULL,
  as_univariate = FALSE,
  est_nsmall = 2,
  p_nsmall = 3,
  show_vars = NULL,
  save_plot = FALSE,
  filename = NULL,
  ...
)
```

Arguments

`data`	A data frame.
`model_vars`	A character vector or a named list of predictor variables for different models.
`y`	A character string of the outcome variable.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`as_univariate`	A logical value indicating whether to treat the model_vars as univariate.
`est_nsmall`	An integer specifying the precision for the estimates in the plot.
`p_nsmall`	An integer specifying the number of decimal places for the p-values.
`show_vars`	A character vector of variable names to be shown in the plot. If `NULL`, all variables are shown.

`save_plot`	A logical value indicating whether to save the plot.
`filename`	A character string specifying the filename for the plot. If `NULL`, a default filename is used.
`...`	Additional arguments passed to the `forestploter::forest` function.

Value

A `gtable` object.

Examples

```
data(cancer, package = "survival")
cancer$ph.ecog_cat <- factor(cancer$ph.ecog, levels = c(0:3), labels = c("0", "1", ">=2", ">=2"))
regression_forest(cancer,
  model_vars = c("age", "sex", "wt.loss", "ph.ecog_cat", "meal.cal"), y = "status", time = "time",
   as_univariate = TRUE, save_plot = FALSE
)

regression_forest(cancer,
  model_vars = c("age", "sex", "wt.loss", "ph.ecog_cat", "meal.cal"), y = "status", time = "time",
   show_vars = c("age", "sex", "ph.ecog_cat", "meal.cal"), save_plot = FALSE
)

regression_forest(cancer,
   model_vars = list(
     M0 = c("age"),
     M1 = c("age", "sex", "wt.loss", "ph.ecog_cat", "meal.cal"),
     M2 = c("age", "sex", "wt.loss", "ph.ecog_cat", "meal.cal", "pat.karno")
   ),
   y = "status", time = "time",
   show_vars = c("age", "sex", "ph.ecog_cat", "meal.cal"), save_plot = FALSE
)
```

`regression_scan` *Scan for significant regression predictors*

Description

Scan for significant regression predictors and output results. Both logistic and Cox proportional hazards regression models are supported. The predictor variables in the model are can be used both in linear form or in restricted cubic spline form.

Usage

```
regression_scan(
  data,
  y,
  time = NULL,
  predictors = NULL,
  covars = NULL,
  num_to_factor = 5,
  p_adjust_method = "BH",
  save_table = FALSE,
  filename = NULL
)
```

Arguments

`data`	A data frame.
`y`	A character string of the outcome variable.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`predictors`	The predictor variables to be scanned for relationships. If `NULL`, all variables except `y` and `time` are taken as predictors.
`covars`	A character vector of covariate names.
`num_to_factor`	An integer. Numerical variables with number of unique values below or equal to this value would be considered a factor.
`p_adjust_method`	The method to use for p-value adjustment for pairwise comparison. Default is "BH". See `?p.adjust.methods`. Note that the p-value adjustment is only applied column wise, not applied among all available p-values in the table.
`save_table`	A logical value indicating whether to save the results as a table.
`filename`	The name of the file to save the results. File will be saved in `.csv` format.

Details

The function first determines the type of each predictor variable (`numerical`, `factor`, `num_factor` (numerical but with less unique values than or equal to `num_to_factor`), or `other`). Then, it performs regression analysis for available transforms of each predictor variable and saves the results.

Value

A data frame containing the results of the regression analysis.

The available transforms for each predictor type are

- `numerical`: `original`, `logarithm`, `categorized`, `rcs`
- `num_factor`: `original`, `categorized`
- `factor`: `original`
- `other`: none

The transforms are applied as follows

- `original`: Fit the regression model with the original variable. Provide HR/OR and p-values in results.
- `logarithm`: If the `numerical` variable is all greater than 0, fit the regression model with the log-transformed variable. Provide HR/OR and p-values in results.
- `categorized`: For `numerical` variables, fit the regression model with the binarized variable split at the median value. For `num_factor` variables, fit the regression model with the variable after `as.factor()`. Provide HR/OR and p-values in results. If the number of levels is greater than 2, no single HR/OR is provided, but the p-value of the overall test can be provided with TYPE-2 ANOVA from `car::Anova()`.
- `rcs`: Fit the regression model with the restricted cubic spline variable. The overall and nonlinear p-values are provided in results. These p-vals are calculated by `anova()` of `rms::cph()` or `rms::Glm`.

Examples

```
data(cancer, package = "survival")
regression_scan(cancer, y = "status", time = "time", save_table = FALSE)
```

`replace_elements` *Replacing elements in a vector*

Description

Replacing elements in a vector

Usage

```
replace_elements(x, from, to)
```

Arguments

`x`	A vector.
`from`	A vector of elements to be replaced.
`to`	A vector of elements to replace the original ones.

Value

A vector.

Examples

```
replace_elements(c("a", "x", "1", NA, "a"), c("a", "b", NA), c("A", "B", "XX"))
```

split_multichoice *Split multi-choice data into columns*

Description

Split multi-choice data into columns, each new column consists of booleans whether a choice is presented.

Usage

```
split_multichoice(
  df,
  quest_cols,
  split = "",
  remove_space = TRUE,
  link = "_",
  remove_cols = TRUE
)
```

Arguments

`df`	A data frame.
`quest_cols`	A vector of column names that contain multi-choice data.
`split`	A string to split the data. Default is `""`.
`remove_space`	If `TRUE`, remove space in the data.
`link`	A string to link the column name and the option. Default is `"_"`.
`remove_cols`	If `TRUE`, remove the original columns.

Value

A data frame with additional columns.

Examples

```
df <- data.frame(q1 = c("ab", "c da", "b a", NA), q2 = c("a b", "a c", "d", "ab"))
split_multichoice(df, quest_cols = c("q1", "q2"))
```

`str_match_replace` *Match string and replace with corresponding value*

Description

Partially match a string and replace with corresponding value. This function is useful to recover the original names of variables after legalized using `make.names` or modified by other functions.

Usage

```
str_match_replace(x, to_match, to_replace)
```

Arguments

`x`	A vector.
`to_match`	A vector of strings to be matched.
`to_replace`	A vector of strings to replace the matched ones, must have the same length as `to_match`.

Value

A vector.

Examples

```
ori_names <- c("xx (mg/dl)", "b*x", "Covid-19")
modified_names <- c("v1", "v2", "v3")
x <- c("v1.v2", "v3.yy", "v4")
str_match_replace(x, modified_names, ori_names)
```

`subgroup_forest` *Create subgroup forest plot.*

Description

Create subgroup forest plot with `glm` or coxph models. The interaction p-values are calculated using likelihood ratio tests.

Usage

```
subgroup_forest(
  data,
  subgroup_vars,
  x,
  y,
  time = NULL,
  standardize_x = FALSE,
  covars = NULL,
  est_nsmall = 2,
  p_nsmall = 3,
  group_cut_quantiles = 0.5,
  save_plot = FALSE,
  filename = NULL,
  ...
)
```

Arguments

`data`	A data frame.
`subgroup_vars`	A character vector of variable names to be used as subgroups. It's recommended that the variables are categorical. If the variables are continuous, they will be cut into groups.
`x`	A character string of the predictor variable.
`y`	A character string of the outcome variable.
`time`	A character string of the time variable. If `NULL`, logistic regression is used. Otherwise, Cox proportional hazards regression is used.
`standardize_x`	A logical value. If `TRUE`, the predictor variable will be standardized.
`covars`	A character vector of covariate names. If duplicated with `subgroup_vars`, the duplicated covariates will be temporarily removed during the corresponding subgroup analysis.
`est_nsmall`	An integer specifying the precision for the estimates in the plot.
`p_nsmall`	An integer specifying the number of decimal places for the p-values.
`group_cut_quantiles`	A vector of numerical values between 0 and 1, specifying the quantile to use for cutting continuous subgroup variables.
`save_plot`	A logical value indicating whether to save the plot.
`filename`	A character string specifying the filename for the plot. If `NULL`, a default filename is used.
`...`	Additional arguments passed to the `forestploter::forest` function.

Value

A `gtable` object.

Examples

```
data(cancer, package = "survival")
# coxph model with time assigned
subgroup_forest(cancer,
  subgroup_vars = c("age", "sex", "wt.loss"), x = "ph.ecog", y = "status",
  time = "time", covars = "ph.karno", ticks_at = c(1, 2), save_plot = FALSE
)

# logistic model with time not assigned
cancer$dead <- cancer$status == 2
subgroup_forest(cancer,
  subgroup_vars = c("age", "sex", "wt.loss"), x = "ph.ecog", y = "dead",
  covars = "ph.karno", ticks_at = c(1, 2), save_plot = FALSE
)

cancer$ph.ecog_cat <- factor(cancer$ph.ecog, levels = c(0:3), labels = c("0", "1", ">=2", ">=2"))
subgroup_forest(cancer,
  subgroup_vars = c("sex", "wt.loss"), x = "ph.ecog_cat", y = "dead",
  covars = "ph.karno", ticks_at = c(1, 2), save_plot = FALSE
)
```

`subject_view` *Get an overview of different subjects in data.*

Description

Get a table of subject details for the clinical data. This table could be labeled and used for subject name standardization.

Usage

```
subject_view(
  df,
  subject_col,
  info_cols,
  value_col = NULL,
  info_n_samples = 10,
  info_collapse = "\n",
  info_unique = FALSE,
  save_table = FALSE,
  filename = NULL
)
```

Arguments

`df`	A data frame of medical records that contains test subject, value, and unit cols.
`subject_col`	The name of the subject column.
`info_cols`	The names of the columns to get detailed information.

`value_col`	The name of the column that contains values. This column must be numerical.
`info_n_samples`	The number of samples to show in the detailed information columns.
`info_collapse`	The separator to use for collapsing the detailed information.
`info_unique`	A logical value indicating whether to show unique values only.
`save_table`	A logical value indicating whether to save the table to a csv file.
`filename`	The name of the csv file to be saved.

Value

A data frame of subject details.

Examples

```
df <- data.frame(subject = sample(c("a", "b"), 1000, replace = TRUE), value = runif(1000))
df$unit <- NA
df$unit[df$subject == "a"] <- sample(c("mg/L", "g/l", "g/L"),
  sum(df$subject == "a"),
  replace = TRUE
)
df$value[df$subject == "a" & df$unit == "mg/L"] <-
  df$value[df$subject == "a" & df$unit == "mg/L"] * 1000
df$unit[df$subject == "b"] <- sample(c(NA, "g", "mg"), sum(df$subject == "b"), replace = TRUE)
df$value[df$subject == "b" & df$unit %in% "mg"] <-
  df$value[df$subject == "b" & df$unit %in% "mg"] * 1000
df$value[df$subject == "b" & is.na(df$unit)] <- df$value[df$subject == "b" & is.na(df$unit)] *
  sample(c(1, 1000), size = sum(df$subject == "b" & is.na(df$unit)), replace = TRUE)
subject_view(
  df = df, subject_col = "subject", info_cols = c("value", "unit"), value_col = "value",
  save_table = FALSE
)
```

`to_date` *Convert numerical or character date to date.*

Description

Convert numerical (especially Excel date) or character date to date. Can deal with common formats and allow different formats in one vector.

Usage

```
to_date(
  x,
  from_excel = TRUE,
  verbose = TRUE,
  try_formats = c("%Y-%m-%d", "%Y/%m/%d", "%Y%m%d", "%Y.%m.%d")
)
```

Arguments

`x`	A vector that stores dates in numerical or character types.
`from_excel`	If TRUE, treat numerical values as Excel dates.
`verbose`	If TRUE, print the values that cannot be converted.
`try_formats`	A character vector of date formats to try. Same as `tryFormats` in `as.Date`.

Value

A single valid value from the vector. `NA` if all values are invalid.

Examples

```
to_date(c(43562, "2020-01-01", "2020/01/01", "20200101", "2020.01.01"))
```

`unit_standardize` *Standardize units of numeric data.*

Description

Standardize units of numeric data, especially for data of medical records with different units.

Usage

```
unit_standardize(df, subject_col, value_col, unit_col, change_rules)
```

Arguments

`df` A data frame of medical records that contains test subject, value, and unit cols.

`subject_col` The name of the subject column.

`value_col` The name of the value column.

`unit_col` The name of the unit column.

`change_rules` A data frame or a list of lists. If a data frame, it must contain the following columns:

- `subject`: The subject to be standardized.
- `unit`: The units of the subject.
- `label`: The role of the unit, the rule is as follows:
  - `"t"`: the target unit to be standardized to. If not specified, the function will use the most common unit in the data (retrieved by `first_mode()`).
  - `"r"`: The units to be removed, and the corresponding values be set to `NA`. Set this when data with this unit cannot be used.
  - A number: Set the multiplier of this unit, the standardized value will be `value * multiplier`. And `NA` and `""` is considered the same as `1`.

If a list of lists, each list contains the following elements:

- `subject`: The subject to be standardized.
- `target_unit`: The target unit to be standardized to. If not specified, the function will use the most common unit in the data (retrieved by `first_mode()`).
- `units2change`: The units to be changed. If not specified, the function will use all units except the target unit. Must be specified to apply different `coeffs`.
- `coeffs`: The coefficients to be used for the conversion. If not specified, the function will use 1 for all units to be changed.
- `units2remove`: The units to be removed, and the corresponding values be set to `NA`. Set this when data with this unit cannot be used.

It's recommended to use the labeled result from `unit_view()` as the input.

Value

A data frame with subject units standardized.

Examples

```
# Example 1: Using the list as change_rules is more convenient for small datasets.
df <- data.frame(
  subject = c("a", "a", "b", "b", "b", "c", "c"), value = c(1, 2, 3, 4, 5, 6, 7),
  unit = c(NA, "x", "x", "x", "y", "a", "b")
)
change_rules <- list(
  list(subject = "a", target_unit = "x", units2change = c(NA), coeffs = c(20)),
  list(subject = "b"),
  list(subject = "c", target_unit = "b")
)
unit_standardize(df,
  subject_col = "subject", value_col = "value", unit_col = "unit",
  change_rules = change_rules
)

# Example 2: Using the labeled result from `unit_view()` as the input
# is more robust for large datasets.
df <- data.frame(subject = sample(c("a", "b"), 1000, replace = TRUE), value = runif(1000))
df$unit <- NA
df$unit[df$subject == "a"] <- sample(c("mg/L", "g/l", "g/L"),
  sum(df$subject == "a"),
  replace = TRUE
)
df$value[df$subject == "a" & df$unit == "mg/L"] <-
  df$value[df$subject == "a" & df$unit == "mg/L"] * 1000
df$unit[df$subject == "b"] <- sample(c(NA, "m.g", "mg"), sum(df$subject == "b"),
  prob = c(0.3, 0.05, 0.65), replace = TRUE
)
df$value[df$subject == "b" & df$unit %in% "mg"] <-
  df$value[df$subject == "b" & df$unit %in% "mg"] * 1000
df$value[df$subject == "b" & is.na(df$unit)] <- df$value[df$subject == "b" & is.na(df$unit)] *
  sample(c(1, 1000), size = sum(df$subject == "b" & is.na(df$unit)), replace = TRUE)
```

```
unit_table <- unit_view(
  df = df, subject_col = "subject",
  value_col = "value", unit_col = "unit", save_table = FALSE
)
unit_table$label <- c("t", NA, 1e-3, NA, NA, "r") # labeling the units

df_standardized <- unit_standardize(
  df = df, subject_col = "subject", value_col = "value",
  unit_col = "unit", change_rules = unit_table
)
unit_view(
  df = df_standardized, subject_col = "subject", value_col = "value", unit_col = "unit",
  save_table = FALSE, conflicts_only = FALSE
)
```

unit_view *Generate a table of conflicting units.*

Description

Get a table of conflicting units for the clinical data, along with the some useful information, this table could be labeled and used for unit standardization.

Usage

```
unit_view(
  df,
  subject_col,
  value_col,
  unit_col,
  quantiles = c(0.025, 0.975),
  save_table = FALSE,
  filename = NULL,
  conflicts_only = TRUE
)
```

Arguments

`df`	A data frame of medical records that contains test subject, value, and unit cols.
`subject_col`	The name of the subject column.
`value_col`	The name of the value column.
`unit_col`	The name of the unit column.
`quantiles`	A vector of quantiles to be shown in the table.
`save_table`	A logical value indicating whether to save the table to a csv file.
`filename`	The name of the csv file to be saved.
`conflicts_only`	A logical value indicating whether to only show the conflicting units.

Value

A data frame of conflicting units.

Examples

```
df <- data.frame(subject = sample(c("a", "b"), 1000, replace = TRUE), value = runif(1000))
df$unit <- NA
df$unit[df$subject == "a"] <- sample(c("mg/L", "g/l", "g/L"),
  sum(df$subject == "a"),
  replace = TRUE
)
df$value[df$subject == "a" & df$unit == "mg/L"] <-
  df$value[df$subject == "a" & df$unit == "mg/L"] * 1000
df$unit[df$subject == "b"] <- sample(c(NA, "g", "mg"), sum(df$subject == "b"), replace = TRUE)
df$value[df$subject == "b" & df$unit %in% "mg"] <-
  df$value[df$subject == "b" & df$unit %in% "mg"] * 1000
df$value[df$subject == "b" & is.na(df$unit)] <- df$value[df$subject == "b" & is.na(df$unit)] *
  sample(c(1, 1000), size = sum(df$subject == "b" & is.na(df$unit)), replace = TRUE)
unit_view(
  df = df, subject_col = "subject",
  value_col = "value", unit_col = "unit", save_table = FALSE
)
```

---

unmake_names *Unmake names*

Description

Inverse function of `make.names`. You can use `make.names` to make colnames legal for subsequent processing and analysis in R. Then use this function to switch back for publication.

Usage

```
unmake_names(x, ori_names)
```

Arguments

`x`	A vector of names generated by `base::make.names()`.
`ori_names`	A vector of original names.

Details

The function will try to match the names in `x` with the names in `ori_names`. If the names in `x` are not in `ori_names`, the function will return `NA`.

Value

A vector of original names.

Examples

```
ori_names <- c("xx (mg/dl)", "b*x", "Covid-19")
x <- c(make.names(ori_names), "aa")
unmake_names(x, ori_names)
```

`value_initial_cleaning` *Preliminarily cleaning numerical string vectors*

Description

Cleaning illegal characters in string vectors that store numerical values. The function is useful for cleaning electrical health records in Chinese.

Usage

```
value_initial_cleaning(x)
```

Arguments

`x`	A string vector that stores numerical values.

Details

The function will convert full-width characters to half-width characters, remove spaces and extra dots, and replace empty strings with `NA`.

Value

A string vector that stores cleaner numerical values.

Examples

```
x <- c("\uFF11\uFF12\uFF13", "11..23", "\uff41\uff42\uff41\uff4e\uff44\uff4f\uff4e")
value_initial_cleaning(x)
```

`vec2code` *Generate code from string vector Generate the code that can be used to generate the string vector.*

Description

Generate code from string vector Generate the code that can be used to generate the string vector.

Usage

```
vec2code(x)
```

Arguments

`x`	A string vector.

Value

A string that contains the code to generate the vector.

Examples

```
vec2code(colnames(mtcars))
```

Index