Package ‘nuggets’

January 8, 2025

Title Extensible Data Pattern Searching Framework

Version 1.4.0

Date 2025-01-08

Maintainer Michal Burda <michal.burda@osu.cz>

Description Extensible framework for
subgroup discovery (Atzmueller (2015) <doi:10.1002/widm.1144>),
contrast patterns (Chen (2022) <doi:10.48550/arXiv.2209.13556>),
emerging patterns (Dong (1999) <doi:10.1145/312129.312191>),
association rules (Agrawal (1994) <https://www.vldb.org/conf/1994/P487.PDF>) and
conditional correlations (Hájek (1978) <doi:10.1007/978-3-642-66943-9>).
Both crisp (Boolean, binary) and fuzzy data are supported.
It generates conditions in the form of elementary conjunctions, evaluates
them on a dataset and checks the induced sub-data for interesting statistical
properties. A user-defined function may be defined to evaluate on each generated
condition to search for custom patterns.

URL https://beerda.github.io/nuggets/,
https://github.com/beerda/nuggets

BugReports https://github.com/beerda/nuggets/issues

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

Imports cli, lifecycle, methods, purrr, Rcpp, rlang, stats, stringr,
tibble, tidyr, tidyselect

LinkingTo Rcpp, testthat

SystemRequirements C++17

Suggests arules, dplyr, testthat (>= 3.0.0), xml2, withr, knitr,
rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Michal Burda [aut, cre] (<`https://orcid.org/0000-0002-4182-4407`>)

Repository CRAN

Date/Publication 2025-01-08 10:20:01 UTC

Contents

`dig` *Search for patterns of custom type*

Description

[Experimental]

A general function for searching for patterns of custom type. The function allows for the selection of columns of `x` to be used as condition predicates. The function enumerates all possible conditions in the form of elementary conjunctions of selected predicates, and for each condition, a user-defined callback function `f` is executed. The callback function is intended to perform some analysis and return an object representing a pattern or patterns related to the condition. `dig()` returns a list of these returned objects.

The callback function `f` may have some arguments that are listed in the `f` argument description. The algorithm provides information about the generated condition based on the present arguments.

Additionally to `condition`, the function allows for the selection of the so-called *focus* predicates. The focus predicates, a.k.a. *foci*, are predicates that are evaluated within each condition and some additional information is provided to the callback function about them.

`dig()` allows to specify some restrictions on the generated conditions, such as:

- the minimum and maximum length of the condition (`min_length` and `max_length` arguments).

- the minimum support of the condition (`min_support` argument). Support of the condition is the relative frequency of the condition in the dataset `x`.
- the minimum support of the focus (`min_focus_support` argument). Support of the focus is the relative frequency of rows such that all condition predicates AND the focus are TRUE on it. Foci with support lower than `min_focus_support` are filtered out.

Usage

```
dig(
  x,
  f,
  condition = everything(),
  focus = NULL,
  disjoint = var_names(colnames(x)),
  min_length = 0,
  max_length = Inf,
  min_support = 0,
  min_focus_support = min_support,
  min_conditional_focus_support = 0,
  max_support = 1,
  filter_empty_foci = FALSE,
  t_norm = "goguen",
  max_results = Inf,
  verbose = FALSE,
  threads = 1L,
 error_context = list(arg_x = "x", arg_f = "f", arg_condition = "condition", arg_focus =
   "focus", arg_disjoint = "disjoint", arg_min_length = "min_length", arg_max_length =
    "max_length", arg_min_support = "min_support", arg_min_focus_support =
    "min_focus_support", arg_min_conditional_focus_support =
    "min_conditional_focus_support", arg_max_support = "max_support",
   arg_filter_empty_foci = "filter_empty_foci", arg_t_norm = "t_norm", arg_max_results =
    "max_results", arg_verbose = "verbose", arg_threads = "threads",
     call =
    current_env())
)
```

Arguments

`x`	a matrix or data frame. The matrix must be numeric (double) or logical. If `x` is a data frame then each column must be either numeric (double) or logical.
`f`	the callback function executed for each generated condition. This function may have some of the following arguments. Based on the present arguments, the algorithm would provide information about the generated condition: • `condition` - a named integer vector of column indices that represent the predicates of the condition. Names of the vector correspond to column names; • `support` - a numeric scalar value of the current condition's support; • `indices` - a logical vector indicating the rows satisfying the condition;

- `weights` - (similar to indices) weights of rows to which they satisfy the current condition;
- `pp` - a value of a contingency table, `condition & focus`. `pp` is a named numeric vector where each value is a support of conjunction of the condition with a foci column (see the `focus` argument to specify, which columns). Names of the vector are foci column names.
- `pn` - a value of a contingency table, `condition & neg focus`. `pn` is a named numeric vector where each value is a support of conjunction of the condition with a negated foci column (see the `focus` argument to specify, which columns are foci) - names of the vector are foci column names.
- `np` - a value of a contingency table, `neg condition & focus`. `np` is a named numeric vector where each value is a support of conjunction of the negated condition with a foci column (see the `focus` argument to specify, which columns are foci) - names of the vector are foci column names.
- `nn` - a value of a contingency table, `neg condition & neg focus`. `nn` is a named numeric vector where each value is a support of conjunction of the negated condition with a negated foci column (see the `focus` argument to specify, which columns are foci) - names of the vector are foci column names.
- `foci_supports` - (deprecated, use `pp` instead) a named numeric vector of supports of foci columns (see `focus` argument to specify, which columns are foci) - names of the vector are foci column names.

`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates
`focus`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as focus predicates
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.
`max_length`	The maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.
`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`min_focus_support`	the minimum support of a focus, for the focus to be passed to the callback function. The support of the focus is the relative frequency of rows such that all condition predicates AND the focus are TRUE on it. For numerical (double)

input, the support is computed as the mean (over all rows) of multiplications of predicate values.

`min_conditional_focus_support`
: the minimum relative support of a focus within a condition. The conditional support of the focus is the relative frequency of rows with focus being TRUE within rows where the condition is TRUE.

`max_support`
: the maximum support of a condition to trigger the callback

`filter_empty_foci`
: a logical scalar indicating whether to skip conditions, for which no focus remains available after filtering by `min_focus_support`. If `TRUE`, the condition is passed to the callback function only if at least one focus remains after filtering. If `FALSE`, the condition is passed to the callback function regardless of the number of remaining foci.

`t_norm`
: a t-norm used to compute conjunction of weights. It must be one of `"goedel"` (minimum t-norm), `"goguen"` (product t-norm), or `"lukas"` (Lukasiewicz t-norm).

`max_results`
: the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.

`verbose`
: a logical scalar indicating whether to print progress messages.

`threads`
: the number of threads to use for parallel computation.

`error_context`
: a list of details to be used in error messages. This argument is useful when `dig()` is called from another function to provide error messages, which refer to arguments of the calling function. The list must contain the following elements:

- `arg_x` - the name of the argument `x` as a character string
- `arg_f` - the name of the argument `f` as a character string
- `arg_condition` - the name of the argument `condition` as a character string
- `arg_focus` - the name of the argument `focus` as a character string
- `arg_disjoint` - the name of the argument `disjoint` as a character string
- `arg_min_length` - the name of the argument `min_length` as a character string
- `arg_max_length` - the name of the argument `max_length` as a character string
- `arg_min_support` - the name of the argument `min_support` as a character string
- `arg_min_focus_support` - the name of the argument `min_focus_support` as a character string
- `arg_max_support` - the name of the argument `max_support` as a character
- `arg_filter_empty_foci` - the name of the argument `filter_empty_foci` as a character string
- `arg_t_norm` - the name of the argument `t_norm` as a character string
- `arg_threads` - the name of the argument `threads` as a character string
- `call` - an environment in which to evaluate the error messages.

Value

A list of results provided by the callback function `f`.

Author(s)

Michal Burda

See Also

`partition()`, `var_names()`, `dig_grid()`

Examples

```
library(tibble)

# Prepare iris data for use with dig()
d <- partition(iris, .breaks = 2)

# Call f() for each condition with support >= 0.5. The result is a list
# of strings representing the conditions.
dig(x = d,
    f = function(condition) {
        format_condition(names(condition))
    },
    min_support = 0.5)

# Create a more complex pattern object - a list with some statistics
res <- dig(x = d,
           f = function(condition, support) {
               list(condition = format_condition(names(condition)),
                    support = support)
           },
           min_support = 0.5)
print(res)

# Format the result as a data frame
do.call(rbind, lapply(res, as_tibble))

# Within each condition, evaluate also supports of columns starting with
# "Species"
res <- dig(x = d,
           f = function(condition, support, pp) {
               c(list(condition = format_condition(names(condition))),
                 list(condition_support = support),
                 as.list(pp / nrow(d)))
           },
           condition = !starts_with("Species"),
           focus = starts_with("Species"),
           min_support = 0.5,
           min_focus_support = 0)

# Format the result as a tibble
```

```
do.call(rbind, lapply(res, as_tibble))

# For each condition, create multiple patterns based on the focus columns
res <- dig(x = d,
           f = function(condition, support, pp) {
               lapply(seq_along(pp), function(i) {
                   list(condition = format_condition(names(condition)),
                        condition_support = support,
                        focus = names(pp)[i],
                        focus_support = pp[[i]] / nrow(d))
               })
           },
           condition = !starts_with("Species"),
           focus = starts_with("Species"),
           min_support = 0.5,
           min_focus_support = 0)

# As res is now a list of lists, we need to flatten it before converting to
# a tibble
res <- unlist(res, recursive = FALSE)

# Format the result as a tibble
do.call(rbind, lapply(res, as_tibble))
```

`dig_associations` *Search for association rules*

Description

[Experimental]

Association rules identify conditions (*antecedents*) under which a specific feature (*consequent*) is present very often.

Scheme: `A => C`

> If condition `A` is satisfied, then the feature `C` is present very often.

Example: `university_edu & middle_age & IT_industry => high_income`

> People in *middle age* with *university education* working in IT industry have very likely a *high income*.

Antecedent `A` is usually a set of predicates, and consequent `C` is a single predicate.

For the following explanations we need a mathematical function $supp(I)$, which is defined for a set I of predicates as a relative frequency of rows satisfying all predicates from I. For logical data, $supp(I)$ equals to the relative frequency of rows, for which all predicates $i_1, i_2, \ldots, i_n$ from I are TRUE. For numerical (double) input, $supp(I)$ is computed as the mean (over all rows) of truth degrees of the formula `i_1 AND i_2 AND ... AND i_n`, where `AND` is a triangular norm selected by the `t_norm` argument.

Association rules are characterized with the following quality measures.

Length of a rule is the number of elements in the antecedent.

Coverage of a rule is equal to $supp(A)$.

Consequent support of a rule is equal to $supp(\{c\})$.

Support of a rule is equal to $supp(A \cup \{c\})$.

Confidence of a rule is the fraction $supp(A)/supp(A \cup \{c\})$.

Usage

```
dig_associations(
  x,
  antecedent = everything(),
  consequent = everything(),
  disjoint = var_names(colnames(x)),
  min_length = 0L,
  max_length = Inf,
  min_coverage = 0,
  min_support = 0,
  min_confidence = 0,
  contingency_table = FALSE,
  measures = NULL,
  t_norm = "goguen",
  max_results = Inf,
  verbose = FALSE,
  threads = 1
)
```

Arguments

`x`	a matrix or data frame with data to search in. The matrix must be numeric (double) or logical. If `x` is a data frame then each column must be either numeric (double) or logical.
`antecedent`	a tidyselect expression (see tidyselect syntax) specifying the columns to use in the antecedent (left) part of the rules
`consequent`	a tidyselect expression (see tidyselect syntax) specifying the columns to use in the consequent (right) part of the rules
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`min_length`	the minimum length, i.e., the minimum number of predicates in the antecedent, of a rule to be generated. Value must be greater or equal to 0. If 0, rules with empty antecedent are generated in the first place.
`max_length`	The maximum length, i.e., the maximum number of predicates in the antecedent, of a rule to be generated. If equal to Inf, the maximum length is limited only by the number of available predicates.

`min_coverage`	the minimum coverage of a rule in the dataset `x`. (See Description for the definition of *coverage*.)
`min_support`	the minimum support of a rule in the dataset `x`. (See Description for the definition of *support*.)
`min_confidence`	the minimum confidence of a rule in the dataset `x`. (See Description for the definition of *confidence*.)
`contingency_table`	a logical value indicating whether to provide a contingency table for each rule. If `TRUE`, the columns `pp`, `pn`, `np`, and `nn` are added to the output table. These columns contain the number of rows satisfying the antecedent and the consequent, the antecedent but not the consequent, the consequent but not the antecedent, and neither the antecedent nor the consequent, respectively.
`measures`	a character vector specifying the additional quality measures to compute. If `NULL`, no additional measures are computed. Possible values are `"lift"`, `"conviction"`, `"added_value"`. See `https://mhahsler.github.io/arules/docs/measures` for a description of the measures.
`t_norm`	a t-norm used to compute conjunction of weights. It must be one of `"goedel"` (minimum t-norm), `"goguen"` (product t-norm), or `"lukas"` (Lukasiewicz t-norm).
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical value indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.

Value

A tibble with found patterns and computed quality measures.

Author(s)

Michal Burda

See Also

`partition()`, `var_names()`, `dig()`

Examples

```
d <- partition(mtcars, .breaks = 2)
dig_associations(d,
                 antecedent = !starts_with("mpg"),
                 consequent = starts_with("mpg"),
                 min_support = 0.3,
                 min_confidence = 0.8,
                 measures = c("lift", "conviction"))
```

dig_baseline_contrasts

Search for conditions that yield in statistically significant one-sample test in selected variables.

Description

[Experimental]

Baseline contrast patterns identify conditions under which a specific feature is significantly different from a given value by performing a one-sample statistical test.

Scheme: `var != 0 | C`

Variable `var` is (in average) significantly different from 0 under the condition `C`.

Example: `(measure_error != 0 | measure_tool_A`

If measuring with measure tool *A*, the average measure error is significantly different from 0.

The baseline contrast is computed using a one-sample statistical test, which is specified by the `method` argument. The function computes the contrast between all variables specified by the `vars` argument. Baseline contrasts are computed in sub-data corresponding to conditions generated from the `condition` columns. Function `dig_baseline_contrasts()` supports crisp conditions only, i.e., the condition columns in `x` must be logical.

Usage

```
dig_baseline_contrasts(
  x,
  condition = where(is.logical),
  vars = where(is.numeric),
  disjoint = var_names(colnames(x)),
  min_length = 0L,
  max_length = Inf,
  min_support = 0,
  max_support = 1,
  method = "t",
  alternative = "two.sided",
  h0 = 0,
  conf_level = 0.95,
  max_p_value = 0.05,
  wilcox_exact = FALSE,
  wilcox_correct = TRUE,
  wilcox_tol_root = 1e-04,
  wilcox_digits_rank = Inf,
  max_results = Inf,
  verbose = FALSE,
  threads = 1
)
```

Arguments

`x`	a matrix or data frame with data to search the patterns in.
`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates
`vars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of contrasts
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.
`max_length`	The maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.
`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`max_support`	the maximum support of a condition to trigger the callback function for it. See argument `min_support` for details of what is the support of a condition.
`method`	a character string indicating which contrast to compute. One of `"t"`, for parametric, or `"wilcox"`, for non-parametric test on equality in position.
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. `"greater"` corresponds to positive association, `"less"` to negative association.
`h0`	a numeric value specifying the null hypothesis for the test. For the `"t"` method, it is the value of the mean. For the `"wilcox"` method, it is the value of the median. The default value is 0.
`conf_level`	a numeric value specifying the level of the confidence interval. The default value is 0.95.
`max_p_value`	the maximum p-value of a test for the pattern to be considered significant. If the p-value of the test is greater than `max_p_value`, the pattern is not included in the result.
`wilcox_exact`	(used for the `"wilcox"` method only) a logical value indicating whether the exact p-value should be computed. If `NULL`, the exact p-value is computed for sample sizes less than 50. See `wilcox.test()` and its `exact` argument for more information. Contrary to the behavior of `wilcox.test()`, the default value is `FALSE`.

`wilcox_correct`	(used for the `"wilcox"` method only) a logical value indicating whether the continuity correction should be applied in the normal approximation for the p-value, if `wilcox_exact` is `FALSE`. See `wilcox.test()` and its `correct` argument for more information.
`wilcox_tol_root`	(used for the `"wilcox"` method only) a numeric value specifying the tolerance for the root-finding algorithm used to compute the exact p-value. See `wilcox.test()` and its `tol.root` argument for more information.
`wilcox_digits_rank`	(used for the `"wilcox"` method only) a numeric value specifying the number of digits to round the ranks to. See `wilcox.test()` and its `digits.rank` argument for more information.
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical scalar indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.

Value

A tibble with found patterns in rows. The following columns are always present:

`condition`	the condition of the pattern as a character string in the form `{p1 & p2 & ... & pn}` where `p1`, `p2`, ..., `pn` are `x`'s column names.
`support`	the support of the condition, i.e., the relative frequency of the condition in the dataset `x`.
`var`	the name of the contrast variable.
`estimate`	the estimated mean or median of variable `var`.
`statistic`	the statistic of the selected test.
`p_value`	the p-value of the underlying test.
`n`	the number of rows in the sub-data corresponding to the condition.
`conf_int_lo`	the lower bound of the confidence interval of the estimate.
`conf_int_hi`	the upper bound of the confidence interval of the estimate.
`alternative`	a character string indicating the alternative hypothesis. The value must be one of `"two.sided"`, `"greater"`, or `"less"`.
`method`	a character string indicating the method used for the test.
`comment`	a character string with additional information about the test (mainly error messages on failure).

For the `"t"` method, the following additional columns are also present (see also `t.test()`):

`df`	the degrees of freedom of the t test.
`stderr`	the standard error of the mean.

Author(s)

Michal Burda

See Also

`dig_paired_baseline_contrasts()`, `dig_complement_contrasts()`, `dig()`, `dig_grid()`, `stats::t.test()`, `stats::wilcox.test()`

`dig_complement_contrasts` *Search for conditions that provide significant differences in selected variables to the rest of the data table*

Description

[Experimental]

Complement contrast patterns identify conditions under which there is a significant difference in some numerical variable between elements that satisfy the identified condition and the rest of the data table.

Scheme: `(var | C) != (var | not C)`

There is a statistically significant difference in variable `var` between group of elements that satisfy condition `C` and a group of elements that do not satisfy condition `C`.

Example: `(life_expectancy | smoker) < (life_expectancy | non-smoker)`

The life expectancy in people that smoke cigarettes is in average significantly lower than in people that do not smoke.

The complement contrast is computed using a two-sample statistical test, which is specified by the `method` argument. The function computes the complement contrast in all variables specified by the `vars` argument. Complement contrasts are computed based on sub-data corresponding to conditions generated from the `condition` columns and the rest of the data table. Function #' `dig_complement_contrasts()` supports crisp conditions only, i.e., the condition columns in `x` must be logical.

Usage

```
dig_complement_contrasts(
  x,
  condition = where(is.logical),
  vars = where(is.numeric),
  disjoint = var_names(colnames(x)),
  min_length = 0L,
  max_length = Inf,
  min_support = 0,
  max_support = 1 - min_support,
  method = "t",
  alternative = "two.sided",
  h0 = if (method == "var") 1 else 0,
  conf_level = 0.95,
  max_p_value = 0.05,
  t_var_equal = FALSE,
  wilcox_exact = FALSE,
  wilcox_correct = TRUE,
  wilcox_tol_root = 1e-04,
  wilcox_digits_rank = Inf,
  max_results = Inf,
  verbose = FALSE,
  threads = 1L
)
```

Arguments

`x`	a matrix or data frame with data to search the patterns in.
`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates
`vars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of contrasts
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.
`max_length`	The maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.
`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`max_support`	the maximum support of a condition to trigger the callback function for it. See argument `min_support` for details of what is the support of a condition.
`method`	a character string indicating which contrast to compute. One of `"t"`, for parametric, or `"wilcox"`, for non-parametric test on equality in position, and `"var"` for F-test on comparison of variances of two populations.
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. `"greater"` corresponds to positive association, `"less"` to negative association.

`h0`	a numeric value specifying the null hypothesis for the test. For the `"t"` method, it is the difference in means. For the `"wilcox"` method, it is the difference in medians. For the `"var"` method, it is the hypothesized ratio of the population variances. The default value is 1 for `"var"` method, and 0 otherwise.
`conf_level`	a numeric value specifying the level of the confidence interval. The default value is 0.95.
`max_p_value`	the maximum p-value of a test for the pattern to be considered significant. If the p-value of the test is greater than `max_p_value`, the pattern is not included in the result.
`t_var_equal`	(used for the `"t"` method only) a logical value indicating whether the variances of the two samples are assumed to be equal. If `TRUE`, the pooled variance is used to estimate the variance in the t-test. If `FALSE`, the Welch (or Satterthwaite) approximation to the degrees of freedom is used. See `t.test()` and its `var.equal` argument for more information.
`wilcox_exact`	(used for the `"wilcox"` method only) a logical value indicating whether the exact p-value should be computed. If `NULL`, the exact p-value is computed for sample sizes less than 50. See `wilcox.test()` and its `exact` argument for more information. Contrary to the behavior of `wilcox.test()`, the default value is `FALSE`.
`wilcox_correct`	(used for the `"wilcox"` method only) a logical value indicating whether the continuity correction should be applied in the normal approximation for the p-value, if `wilcox_exact` is `FALSE`. See `wilcox.test()` and its `correct` argument for more information.
`wilcox_tol_root`	(used for the `"wilcox"` method only) a numeric value specifying the tolerance for the root-finding algorithm used to compute the exact p-value. See `wilcox.test()` and its `tol.root` argument for more information.
`wilcox_digits_rank`	(used for the `"wilcox"` method only) a numeric value specifying the number of digits to round the ranks to. See `wilcox.test()` and its `digits.rank` argument for more information.
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical scalar indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.

Value

A tibble with found patterns in rows. The following columns are always present:

`condition`	the condition of the pattern as a character string in the form `{p1 & p2 & ... & pn}` where `p1`, `p2`, ..., `pn` are `x`'s column names.
`support`	the support of the condition, i.e., the relative frequency of the condition in the dataset `x`.

`var`	the name of the contrast variable.
`estimate`	the estimate value (see the underlying test.
`statistic`	the statistic of the selected test.
`p_value`	the p-value of the underlying test.
`n_x`	the number of rows in the sub-data corresponding to the condition.
`n_y`	the number of rows in the sub-data corresponding to the negation of the condition.
`conf_int_lo`	the lower bound of the confidence interval of the estimate.
`conf_int_hi`	the upper bound of the confidence interval of the estimate.
`alternative`	a character string indicating the alternative hypothesis. The value must be one of `"two.sided"`, `"greater"`, or `"less"`.
`method`	a character string indicating the method used for the test.
`comment`	a character string with additional information about the test (mainly error messages on failure).

For the `"t"` method, the following additional columns are also present (see also `t.test()`):

`df`	the degrees of freedom of the t test.
`stderr`	the standard error of the mean difference.

Author(s)

Michal Burda

See Also

`dig_baseline_contrasts()`, `dig_paired_baseline_contrasts()`, `dig()`, `dig_grid()`, `stats::t.test()`, `stats::wilcox.test()`, `stats::var.test()`

`dig_correlations` *Search for conditional correlations*

Description

[Experimental]

Conditional correlations are patterns that identify strong relationships between pairs of numeric variables under specific conditions.

Scheme: `xvar ~ yvar | C`

`xvar` and `yvar` highly correlates in data that satisfy the condition `C`.

Example: `study_time ~ test_score | hard_exam`

For *hard exams*, the amount of *study time* is highly correlated with the obtained exam's *test score*.

The function computes correlations between all combinations of `xvars` and `yvars` columns of `x` in multiple sub-data corresponding to conditions generated from `condition` columns.

Usage

```
dig_correlations(
  x,
  condition = where(is.logical),
  xvars = where(is.numeric),
  yvars = where(is.numeric),
  disjoint = var_names(colnames(x)),
  method = "pearson",
  alternative = "two.sided",
  exact = NULL,
  min_length = 0L,
  max_length = Inf,
  min_support = 0,
  max_support = 1,
  max_results = Inf,
  verbose = FALSE,
  threads = 1
)
```

Arguments

`x`	a matrix or data frame with data to search in.
`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates
`xvars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of correlations
`yvars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of correlations
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`method`	a character string indicating which correlation coefficient is to be used for the test. One of `"pearson"`, `"kendall"`, or `"spearman"`
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. `"greater"` corresponds to positive association, `"less"` to negative association.
`exact`	a logical indicating whether an exact p-value should be computed. Used for Kendall's *tau* and Spearman's *rho*. See `stats::cor.test()` for more information.
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.

`max_length`	The maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.
`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`max_support`	the maximum support of a condition to trigger the callback function for it. See argument `min_support` for details of what is the support of a condition.
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical scalar indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.

Value

A tibble with found patterns.

Author(s)

Michal Burda

See Also

`dig()`, `stats::cor.test()`

Examples

```
# convert iris$Species into dummy logical variables
d <- partition(iris, Species)

# find conditional correlations between all pairs of numeric variables
dig_correlations(d,
                 condition = where(is.logical),
                 xvars = Sepal.Length:Petal.Width,
                 yvars = Sepal.Length:Petal.Width)

# With `condition = NULL`, dig_correlations() computes correlations between
# all pairs of numeric variables on the whole dataset only, which is an
# alternative way of computing the correlation matrix
dig_correlations(iris,
                 condition = NULL,
                 xvars = Sepal.Length:Petal.Width,
                 yvars = Sepal.Length:Petal.Width)
```

dig_grid	*Search for grid-based rules*

Description

[Experimental]

This function creates a grid column names specified by xvars and yvars (see var_grid()). After that, it enumerates all conditions created from data in x (by calling dig()) and for each such condition and for each row of the grid of combinations, a user-defined function f is executed on each sub-data created from x by selecting all rows of x that satisfy the generated condition and by selecting the columns in the grid's row.

Function is useful for searching for patterns that are based on the relationships between pairs of columns, such as in dig_correlations().

Usage

```
dig_grid(
  x,
  f,
  condition = where(is.logical),
  xvars = where(is.numeric),
  yvars = where(is.numeric),
  disjoint = var_names(colnames(x)),
  allow = "all",
  na_rm = FALSE,
  type = "crisp",
  min_length = 0L,
  max_length = Inf,
  min_support = 0,
  max_support = 1,
  max_results = Inf,
  verbose = FALSE,
  threads = 1L,
 error_context = list(arg_x = "x", arg_f = "f", arg_condition = "condition", arg_xvars =
   "xvars", arg_yvars = "yvars", arg_disjoint = "disjoint", arg_allow = "allow",
  arg_na_rm = "na_rm", arg_type = "type", arg_min_length = "min_length", arg_max_length
  = "max_length", arg_min_support = "min_support", arg_max_support = "max_support",
  arg_max_results = "max_results", arg_verbose = "verbose", arg_threads = "threads",
   call = current_env())
)
```

Arguments

x	a matrix or data frame with data to search in.
f	the callback function to be executed for each generated condition. The arguments of the callback function differ based on the value of the type argument (see below):

- If `type = "crisp"` (that is, boolean), the callback function `f` must accept a single argument `pd` of type `data.frame` with single (if `yvars == NULL`) or two (if `yvars != NULL`) columns, accessible as `pd[[1]]` and `pd[[2]]`. Data frame `pd` is a subset of the original data frame `x` with all rows that satisfy the generated condition. Optionally, the callback function may accept an argument `nd` that is a subset of the original data frame `x` with all rows that do not satisfy the generated condition.
- If `type = "fuzzy"`, the callback function `f` must accept an argument `d` of type `data.frame` with single (if `yvars == NULL`) or two (if `yvars != NULL`) columns, accessible as `d[[1]]` and `d[[2]]`, and a numeric argument `weights` with the same length as the number of rows in `d`. The `weights` argument contains the truth degree of the generated condition for each row of `d`. The truth degree is a number in the interval $[0, 1]$ that represents the degree of satisfaction of the condition in the original data row.

In all cases, the function must return a list of scalar values, which will be converted into a single row of result of final tibble.

`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates. The selected columns must be logical or numeric. If numeric, fuzzy conditions are considered.
`xvars`	a tidyselect expression (see tidyselect syntax) specifying the columns of `x`, whose names will be used as a domain for combinations use at the first place (xvar)
`yvars`	`NULL` or a tidyselect expression (see tidyselect syntax) specifying the columns of `x`, whose names will be used as a domain for combinations use at the second place (yvar)
`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`allow`	a character string specifying which columns are allowed to be selected by `xvars` and `yvars` arguments. Possible values are: • `"all"` - all columns are allowed to be selected • `"numeric"` - only numeric columns are allowed to be selected
`na_rm`	a logical value indicating whether to remove rows with missing values from sub-data before the callback function `f` is called
`type`	a character string specifying the type of conditions to be processed. The `"crisp"` type accepts only logical columns as condition predicates. The `"fuzzy"` type accepts both logical and numeric columns as condition predicates where numeric data are in the interval $[0, 1]$. The callback function `f` differs based on the value of the `type` argument (see the description of `f` above).
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.
`max_length`	the maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.

`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`max_support`	the maximum support of a condition to trigger the callback function for it. See argument `min_support` for details of what is the support of a condition.
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical scalar indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.
`error_context`	a list of details to be used in error messages. This argument is useful when `dig_grid()` is called from another function to provide error messages, which refer to arguments of the calling function. The list must contain the following elements: • `arg_x` - the name of the argument `x` as a character string • `arg_condition` - the name of the argument `condition` as a character string • `arg_xvars` - the name of the argument `xvars` as a character string • `arg_yvars` - the name of the argument `yvars` as a character string • `call` - an environment in which to evaluate the error messages.

Value

A tibble with found patterns. Each row represents a single call of the callback function `f`.

Author(s)

Michal Burda

See Also

`dig()`, `var_grid()`; see also `dig_correlations()` and `dig_paired_baseline_contrasts()`, as they are using this function internally.

Examples

```
# *** Example of crisp (boolean) patterns:
# dichotomize iris$Species
crispIris <- partition(iris, Species)

# a simple callback function that computes mean difference of `xvar` and `yvar`
f <- function(pd) {
    list(m = mean(pd[[1]] - pd[[2]]),
         n = nrow(pd))
    }
```

```
# call f() for each condition created from column `Species`
dig_grid(crispIris,
         f,
         condition = starts_with("Species"),
         xvars = starts_with("Sepal"),
         yvars = starts_with("Petal"),
         type = "crisp")

# *** Example of fuzzy patterns:
# create fuzzy sets from Sepal columns
fuzzyIris <- partition(iris,
                       starts_with("Sepal"),
                       .method = "triangle",
                       .breaks = 3)

# a simple callback function that computes a weighted mean of a difference of
# `xvar` and `yvar`
f <- function(d, weights) {
    list(m = weighted.mean(d[[1]] - d[[2]], w = weights),
         w = sum(weights))
}

# call f() for each fuzzy condition created from column fuzzy sets whose
# names start with "Sepal"
dig_grid(fuzzyIris,
         f,
         condition = starts_with("Sepal"),
         xvars = Petal.Length,
         yvars = Petal.Width,
         type = "fuzzy")
```

`dig_paired_baseline_contrasts` *Search for conditions that provide significant differences between paired variables*

Description

[Experimental]

Paired baseline contrast patterns identify conditions under which there is a significant difference in some statistical feature between two paired numeric variables.

Scheme: `(xvar - yvar) != 0 | C`

There is a statistically significant difference between paired variables `xvar` and `yvar` under the condition `C`.

Example: `(daily_ice_cream_income - daily_tea_income) > 0 | sunny`

> Under the condition of *sunny weather*, the paired test shows that *daily ice-cream income* is significantly higher than the *daily tea income*.

The paired baseline contrast is computed using a paired version of a statistical test, which is specified by the `method` argument. The function computes the paired contrast between all pairs of variables, where the first variable is specified by the `xvars` argument and the second variable is specified by the `yvars` argument. Paired baseline contrasts are computed in sub-data corresponding to conditions generated from the `condition` columns. Function `dig_paired_baseline_contrasts()` supports crisp conditions only, i.e., the condition columns in `x` must be logical.

Usage

```
dig_paired_baseline_contrasts(
  x,
  condition = where(is.logical),
  xvars = where(is.numeric),
  yvars = where(is.numeric),
  disjoint = var_names(colnames(x)),
  min_length = 0L,
  max_length = Inf,
  min_support = 0,
  max_support = 1,
  method = "t",
  alternative = "two.sided",
  h0 = 0,
  conf_level = 0.95,
  max_p_value = 1,
  t_var_equal = FALSE,
  wilcox_exact = FALSE,
  wilcox_correct = TRUE,
  wilcox_tol_root = 1e-04,
  wilcox_digits_rank = Inf,
  max_results = Inf,
  verbose = FALSE,
  threads = 1
)
```

Arguments

`x`	a matrix or data frame with data to search the patterns in.
`condition`	a tidyselect expression (see tidyselect syntax) specifying the columns to use as condition predicates
`xvars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of contrasts
`yvars`	a tidyselect expression (see tidyselect syntax) specifying the columns to use for computation of contrasts

`disjoint`	an atomic vector of size equal to the number of columns of `x` that specifies the groups of predicates: if some elements of the `disjoint` vector are equal, then the corresponding columns of `x` will NOT be present together in a single condition. If `x` is prepared with `partition()`, using the `var_names()` function on `x`'s column names is a convenient way to create the `disjoint` vector.
`min_length`	the minimum size (the minimum number of predicates) of the condition to be generated (must be greater or equal to 0). If 0, the empty condition is generated in the first place.
`max_length`	The maximum size (the maximum number of predicates) of the condition to be generated. If equal to Inf, the maximum length of conditions is limited only by the number of available predicates.
`min_support`	the minimum support of a condition to trigger the callback function for it. The support of the condition is the relative frequency of the condition in the dataset `x`. For logical data, it equals to the relative frequency of rows such that all condition predicates are TRUE on it. For numerical (double) input, the support is computed as the mean (over all rows) of multiplications of predicate values.
`max_support`	the maximum support of a condition to trigger the callback function for it. See argument `min_support` for details of what is the support of a condition.
`method`	a character string indicating which contrast to compute. One of `"t"`, for parametric, or `"wilcox"`, for non-parametric test on equality in position.
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. `"greater"` corresponds to positive association, `"less"` to negative association.
`h0`	a numeric value specifying the null hypothesis for the test. For the `"t"` method, it is the difference in means. For the `"wilcox"` method, it is the difference in medians. The default value is 0.
`conf_level`	a numeric value specifying the level of the confidence interval. The default value is 0.95.
`max_p_value`	the maximum p-value of a test for the pattern to be considered significant. If the p-value of the test is greater than `max_p_value`, the pattern is not included in the result.
`t_var_equal`	(used for the `"t"` method only) a logical value indicating whether the variances of the two samples are assumed to be equal. If `TRUE`, the pooled variance is used to estimate the variance in the t-test. If `FALSE`, the Welch (or Satterthwaite) approximation to the degrees of freedom is used. See `t.test()` and its `var.equal` argument for more information.
`wilcox_exact`	(used for the `"wilcox"` method only) a logical value indicating whether the exact p-value should be computed. If `NULL`, the exact p-value is computed for sample sizes less than 50. See `wilcox.test()` and its `exact` argument for more information. Contrary to the behavior of `wilcox.test()`, the default value is `FALSE`.
`wilcox_correct`	(used for the `"wilcox"` method only) a logical value indicating whether the continuity correction should be applied in the normal approximation for the p-value, if `wilcox_exact` is `FALSE`. See `wilcox.test()` and its `correct` argument for more information.

`wilcox_tol_root`	(used for the `"wilcox"` method only) a numeric value specifying the tolerance for the root-finding algorithm used to compute the exact p-value. See `wilcox.test()` and its `tol.root` argument for more information.
`wilcox_digits_rank`	(used for the `"wilcox"` method only) a numeric value specifying the number of digits to round the ranks to. See `wilcox.test()` and its `digits.rank` argument for more information.
`max_results`	the maximum number of generated conditions to execute the callback function on. If the number of found conditions exceeds `max_results`, the function stops generating new conditions and returns the results. To avoid long computations during the search, it is recommended to set `max_results` to a reasonable positive value. Setting `max_results` to `Inf` will generate all possible conditions.
`verbose`	a logical scalar indicating whether to print progress messages.
`threads`	the number of threads to use for parallel computation.

Value

A tibble with found patterns in rows. The following columns are always present:

`condition`	the condition of the pattern as a character string in the form `{p1 & p2 & ... & pn}` where `p1`, `p2`, ..., `pn` are `x`'s column names.
`support`	the support of the condition, i.e., the relative frequency of the condition in the dataset `x`.
`xvar`	the name of the first variable in the contrast.
`yvar`	the name of the second variable in the contrast.
`estimate`	the estimated difference of variable `var`.
`statistic`	the statistic of the selected test.
`p_value`	the p-value of the underlying test.
`n`	the number of rows in the sub-data corresponding to the condition.
`conf_int_lo`	the lower bound of the confidence interval of the estimate.
`conf_int_hi`	the upper bound of the confidence interval of the estimate.
`alternative`	a character string indicating the alternative hypothesis. The value must be one of `"two.sided"`, `"greater"`, or `"less"`.
`method`	a character string indicating the method used for the test.
`comment`	a character string with additional information about the test (mainly error messages on failure).

For the `"t"` method, the following additional columns are also present (see also `t.test()`):

`df`	the degrees of freedom of the t test.
`stderr`	the standard error of the mean difference.

Author(s)

Michal Burda

See Also

dig_baseline_contrasts(), dig_complement_contrasts(), dig(), dig_grid(), stats::t.test(), stats::wilcox.test()

Examples

```
# Compute ratio of sepal and petal length and width for iris dataset
crispIris <- iris
crispIris$Sepal.Ratio <- iris$Sepal.Length / iris$Sepal.Width
crispIris$Petal.Ratio <- iris$Petal.Length / iris$Petal.Width

# Create predicates from the Species column
crispIris <- partition(crispIris, Species)

# Compute paired contrasts for ratios of sepal and petal length and width
dig_paired_baseline_contrasts(crispIris,
                              condition = where(is.logical),
                              xvars = Sepal.Ratio,
                              yvars = Petal.Ratio,
                              method = "t",
                              min_support = 0.1)
```

format_condition *Format a vector of predicates into a string with a condition*

Description

Function takes a character vector of predicates and returns a formatted condition. The format of the condition is a string with predicates separated by commas and enclosed in curly braces.

Usage

```
format_condition(condition)
```

Arguments

condition	a character vector of predicates to be formatted

Value

a character scalar with a formatted condition

Author(s)

Michal Burda

Examples

```
format_condition(NULL)             # returns {}
format_condition(c("a", "b", "c")) # returns {a,b,c}
```

`is_degree` *Tests whether the given argument is a numeric value from the interval* $[0, 1]$

Description

Tests whether the given argument is a numeric value from the interval $[0, 1]$

Usage

```
is_degree(x, na_rm = FALSE)
```

Arguments

`x`	the value to be tested
`na_rm`	whether to ignore `NA` values

Value

`TRUE` if `x` is a numeric vector, matrix or array with values between 0 and 1, otherwise, `FALSE` is returned. If `na_rm` is `TRUE`, `NA` values are treated as valid values. If `na_rm` is `FALSE` and `x` contains `NA` values, `FALSE` is returned.

Author(s)

Michal Burda

`is_subset` *Determine whether the first vector is a subset of the second vector*

Description

Determine whether the first vector is a subset of the second vector

Usage

```
is_subset(x, y)
```

Arguments

`x`	the first vector
`y`	the second vector

Value

`TRUE` if `x` is a subset of `y`, or `FALSE` otherwise. `x` is considered a subset of `y` if all elements of `x` are also in `y`, i.e., if `setdiff(x, y)` is a vector of length 0.

Author(s)

Michal Burda

`partition` *Convert columns of data frame to Boolean or fuzzy sets*

Description

Convert the selected columns of the data frame into either dummy logical columns, or into membership degrees of fuzzy sets, while leaving the remaining columns untouched. Each column selected for transformation typically yields in multiple columns in the output.

Usage

```
partition(
  .data,
  .what = everything(),
  ...,
  .breaks = NULL,
  .labels = NULL,
  .na = TRUE,
  .keep = FALSE,
  .method = "crisp",
  .right = TRUE
)
```

Arguments

`.data`	the data frame to be processed
`.what`	a tidyselect expression (see tidyselect syntax) specifying the columns to be transformed
`...`	optional other tidyselect expressions selecting additional columns to be processed
`.breaks`	for numeric columns, this has to be either an integer scalar or a numeric vector. If `.breaks` is an integer scalar, it specifies the number of resulting intervals to break the numeric column to (for `.method="crisp"`) or the number of target fuzzy sets (for `.method="triangle"` or `.method="raisedcos`). If `.breaks` is a vector, the values specify the borders of intervals (for `.method="crisp"`) or the breaking points of fuzzy sets.
`.labels`	character vector specifying the names used to construct the newly created column names. If `NULL`, the labels are generated automatically.

`.na`	if `TRUE`, an additional logical column is created for each source column that contains `NA` values. For column named `x`, the newly created column's name will be `x=NA`.
`.keep`	if `TRUE`, the original columns being transformed remain present in the resulting data frame.
`.method`	The method of transformation for numeric columns. Either `"crisp"`, `"triangle"`, or `"raisedcos"` is required.
`.right`	If `.method="crisp"`, this argument specifies if the intervals should be closed on the right (and open on the left) or vice versa.

Details

Transformations performed by this function are typically useful as a preprocessing step before using the `dig()` function or some of its derivatives (`dig_correlations()`, `dig_paired_baseline_contrasts()`, `dig_associations()`).

The transformation of selected columns differ based on the type. Concretely:

- **logical** column `x` is transformed into pair of logical columns, `x=TRUE` and`x=FALSE`;
- **factor** column `x`, which has levels `l1`, `l2`, and `l3`, is transformed into three logical columns named `x=l1`, `x=l2`, and `x=l3`;
- **numeric** column`x` is transformed accordingly to `.method` argument:
  - if `.method="crisp"`, the column is first transformed into a factor with intervals as factor levels and then it is processed as a factor (see above);
  - for other `.method` (`triangle` or `raisedcos`), several new columns are created, where each column has numeric values from the interval $[0, 1]$ and represents a certain fuzzy set (either triangular or raised-cosinal). Details of transformation of numeric columns can be specified with additional arguments (`.breaks`, `.labels`, `.right`).

Value

A tibble created by transforming `.data`.

Author(s)

Michal Burda

Examples

```
# transform logical columns and factors
d <- data.frame(a = c(TRUE, TRUE, FALSE),
                b = factor(c("A", "B", "A")),
                c = c(1, 2, 3))
partition(d, a, b)

# transform numeric columns to logical columns (crisp transformation)
partition(CO2, conc:uptake, .method = "crisp", .breaks = 3)

# transform numeric columns to fuzzy sets (triangle transformation)
```

```
partition(CO2, conc:uptake, .method = "triangle", .breaks = 3)

# complex transformation with different settings for each column
CO2 |>
    partition(Plant:Treatment) |>
    partition(conc,
              .method = "raisedcos",
              .breaks = c(-Inf, 95, 175, 350, 675, 1000, Inf)) |>
    partition(uptake,
              .method = "triangle",
              .breaks = c(-Inf, 7.7, 28.3, 45.5, Inf),
              .labels = c("low", "medium", "high"))
```

var_grid — *Create a tibble of combinations of selected column names*

Description

xvars and yvars arguments are tidyselect expressions (see tidyselect syntax) that specify the columns of x whose names will be used as a domain for combinations.

If yvars is NULL, the function creates a tibble with one column var enumerating all column names specified by the xvars argument.

If yvars is not NULL, the function creates a tibble with two columns, xvar and yvar, whose rows enumerate all combinations of column names specified by the xvars and yvars argument.

It is allowed to specify the same column in both xvars and yvars arguments. In such a case, the combinations of the same column with itself are removed from the result.

In other words, the function creates a grid of all possible pairs (xx, yy) where $xx \in xvars$, $yy \in yvars$, and $xx \neq yy$.

Usage

```
var_grid(
  x,
  xvars = everything(),
  yvars = everything(),
  allow = "all",
  xvar_name = if (quo_is_null(enquo(yvars))) "var" else "xvar",
  yvar_name = "yvar",
 error_context = list(arg_x = "x", arg_xvars = "xvars", arg_yvars = "yvars", arg_allow =
    "allow", arg_xvar_name = "xvar_name", arg_yvar_name = "yvar_name", call =
    current_env())
)
```

Arguments

`x`	either a data frame or a matrix
`xvars`	a tidyselect expression (see tidyselect syntax) specifying the columns of `x`, whose names will be used as a domain for combinations use at the first place (xvar)
`yvars`	`NULL` or a tidyselect expression (see tidyselect syntax) specifying the columns of `x`, whose names will be used as a domain for combinations use at the second place (yvar)
`allow`	a character string specifying which columns are allowed to be selected by `xvars` and `yvars` arguments. Possible values are: • `"all"` - all columns are allowed to be selected • `"numeric"` - only numeric columns are allowed to be selected
`xvar_name`	the name of the first column in the resulting tibble.
`yvar_name`	the name of the second column in the resulting tibble. The column does not exist if `yvars` is `NULL`.
`error_context`	A list of details to be used in error messages. This argument is useful when `var_grid()` is called from another function to provide error messages, which refer to arguments of the calling function. The list must contain the following elements: • `arg_x` - the name of the argument `x` as a character string • `arg_xvars` - the name of the argument `xvars` as a character string • `arg_yvars` - the name of the argument `yvars` as a character string • `arg_allow` - the name of the argument `allow` as a character string • `arg_xvar_name` - the name of the `xvar` column in the output tibble • `arg_yvar_name` - the name of the `yvar` column in the output tibble • `call` - an environment in which to evaluate the error messages.

Value

if `yvars` is `NULL`, the function returns a tibble with a single column (`var`). If `yvars` is a non-`NULL` expression, the function returns two columns (`xvar` and `yvar`) with rows enumerating all combinations of column names specified by tidyselect expressions in `xvars` and `yvars` arguments.

Author(s)

Michal Burda

Examples

```
# Create a grid of combinations of all pairs of columns in the CO2 dataset:
var_grid(CO2)

# Create a grid of combinations of all pairs of columns in the CO2 dataset
# such that the first, i.e., `xvar` column is `Plant`, `Type`, or
# `Treatment`, and the second, i.e., `yvar` column is `conc` or `uptake`:
var_grid(CO2, xvars = Plant:Treatment, yvars = conc:uptake)
```

var_names *Extract variable names from predicates*

Description

The function assumes that `x` is a vector of predicate names, i.e., a character vector with elements compatible with pattern `<varname>=<value>`. The function returns the `<varname>` part of these elements. If the string does not correspond to the pattern `<varname>=<value>`, i.e., if the equal sign (=) is missing in the string, the whole string is returned.

Usage

```
var_names(x)
```

Arguments

`x`	A character vector of predicate names.

Value

A `<varname>` part of predicate names in `x`.

Author(s)

Michal Burda

Examples

```
var_names(c("a=1", "a=2", "b=x", "b=y")) # returns c("a", "a", "b", "b")
```

which_antichain *Return indices of first elements of the list, which are incomparable with preceding elements.*

Description

The function returns indices of elements from the given list `x`, which are incomparable (i.e., it is neither subset nor superset) with any preceding element. The first element is always selected. The next element is selected only if it is incomparable with all previously selected elements.

Usage

```
which_antichain(x, distance = 0)
```

Arguments

`x`	a list of integerish vectors
`distance`	a non-negative integer, which specifies the allowed discrepancy between compared sets

Value

an integer vector of indices of selected (incomparable) elements.

Author(s)

Michal Burda

Index