Package: RCLabels (via r-universe)

August 27, 2024

Title Manipulate Matrix Row and Column Labels with Ease

Version 0.1.10

Date 2024-01-26

Description Functions to assist manipulation of matrix row and column labels for all types of matrix mathematics where row and column labels are to be respected.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports assertthat, Hmisc, magrittr, purrr

Suggests dplyr, knitr, rmarkdown, spelling, stringr, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first notation

Depends R (>= 2.10)

LazyData true

VignetteBuilder knitr

Language en-US

URL `https://matthewheun.github.io/RCLabels/`

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

RemoteUrl https://github.com/matthewheun/rclabels

RemoteRef HEAD

RemoteSha 83416e89bbecf580439eb534ad30eafda8171d44

Contents

`arrow_notation` *Arrow notation*

Description

A description of arrow notation.

Usage

```
arrow_notation
```

Format

A vector of notational symbols that provides an arrow separator ("a -> b") between prefix and suffix.

Examples

```
arrow_notation
```

`bracket_arrow_notation` *Bracket arrow notation*

Description

A description of bracket arrow notation.

Usage

```
bracket_arrow_notation
```

Format

A vector of notational symbols that provides bracket arrow ("a [-> b]") notation.

Examples

```
bracket_arrow_notation
```

`bracket_notation` *Bracket notation*

Description

A description of bracket notation.

Usage

```
bracket_notation
```

Format

A vector of notational symbols that provides bracket ("a [b]") notation.

Examples

```
bracket_notation
```

`dash_notation` *A description of dash notation.*

Description

A description of dash notation.

Usage

```
dash_notation
```

Format

A vector of notational symbols that provides an dash separator ("a - b") between prefix and suffix.

Examples

```
dash_notation
```

`first_dot_notation` *First dot notation*

Description

A description of first dot notation. Note that "a.b.c" splits into prefix ("a") and suffix ("b.c").

Usage

```
first_dot_notation
```

Format

A vector of notational symbols that provides first dot ("a.b") notation.

Examples

```
first_dot_notation
```

from_notation *From notation*

Description

A description of from notation.

Usage

```
from_notation
```

Format

A vector of notational symbols that provides from ("a [from b]") notation.

Examples

```
from_notation
```

get_nouns *Extract nouns from row and column labels*

Description

Nouns are the first part of a row-column label, "a" in "a [b]". Internally, this function calls `get_pref_suff(which = "pref")`.

Usage

```
get_nouns(
  labels,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = TRUE
)
```

Arguments

`labels`	A list or vector of labels from which nouns are to be extracted.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting nouns. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `TRUE`.

Value

A list of nouns from row and column labels.

Examples

```
get_nouns("a [b]", notation = bracket_notation)
# Also works with vectors and lists.
get_nouns(c("a [b]", "c [d]"))
get_nouns(list("a [b]", "c [d]"))
```

get_objects *Extract objects of prepositional phrases in row and column labels*

Description

This function extracts the objects of prepositional phrases from row and column labels. The format of the output is a list of named items, one name for each preposition encountered in labels. Objects are `NA` if there is no prepositional phrase starting with that preposition.

Usage

```
get_objects(
  labels,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE,
  prepositions = RCLabels::prepositions_list
)
```

Arguments

`labels`	The row and column labels from which prepositional phrases are to be extracted.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.
`prepositions`	A vector of strings to be treated as prepositions. Note that a space is appended to each word internally, so, e.g., "to" becomes "to ". Default is `RCLabels::prepositions_list`.

Value

A list of objects of prepositional phrases, with names being prepositions, and values being objects.

Examples

```
get_objects(c("a [of b into c]", "d [of Coal from e -> f]"))
```

---

get_piece *Get a piece of a label*

Description

This is a wrapper function for `get_pref_suff()`, `get_nouns()`, and `get_objects()`. It returns a `piece` of a row or column label.

Usage

```
get_piece(
  labels,
  piece = "all",
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE,
  prepositions = RCLabels::prepositions_list
)
```

Arguments

`labels`	The row and column labels from which prepositional phrases are to be extracted.
`piece`	The name of the item to return.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.
`prepositions`	A vector of strings to be treated as prepositions. Note that a space is appended to each word internally, so, e.g., "to" becomes "to ". Default is `RCLabels::prepositions_list`.

Details

`piece` is typically one of

- "all" (which returns `labels` directly),
- "pref" (for the prefixes),
- "suff" (for the suffixes),
- "noun" (returns the noun),
- "pps" (prepositional phrases, returns prepositional phrases in full),
- "prepositions" (returns a list of prepositions),
- "objects" (returns a list of objects with prepositions as names), or
- a preposition in `prepositions` (as a string), which will return the object of that preposition named by the preposition itself.

`piece` must be a character vector of length 1. If a `piece` is missing in a label, "" (empty string) is returned.

If specifying more than one `notation`, be sure the notations are in a list. `notation = c(RCLabels::bracket_notation, RCLabels::arrow_notation)` is unlikely to produce the desired result, because the notations are concatenated together to form a long string vector. Rather say `notation = list(RCLabels::bracket_notation, RCLabels::arrow_notation)`.

Value

A `piece` of `labels`.

Examples

```
labs <- c("a [from b in c]", "d [of e in f]", "Export [of Coal from USA to MEX]")
get_piece(labs, "pref")
get_piece(labs, "suff")
get_piece(labs, piece = "noun")
get_piece(labs, piece = "pps")
get_piece(labs, piece = "prepositions")
get_piece(labs, piece = "objects")
get_piece(labs, piece = "from")
get_piece(labs, piece = "in")
get_piece(labs, piece = "of")
get_piece(labs, piece = "to")
```

`get_pps` *Extract prepositional phrases of row and column labels*

Description

This function extracts prepositional phrases from suffixes of row and column labels of the form "a [preposition b]", where "preposition b" is the prepositional phrase.

Usage

```
get_pps(
  labels,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE,
  prepositions = RCLabels::prepositions_list
)
```

Arguments

`labels`	A list or vector of labels from which prepositional phrases are to be extracted.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositional phrases. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.
`prepositions`	A list of prepositions for which to search. Default is `RCLabels::prepositions_list`.

Value

All prepositional phrases in a suffix.

Examples

```
get_pps(c("a [in b]", "c [of d]"))
get_pps(c("a [of b in c]", "d [-> e of f]"))
```

`get_prepositions` *Extract prepositions from row and column labels*

Description

This function extracts prepositions from a list of row and column labels. The list has outer structure of the number of labels and an inner structure of each prepositional phrase in the specific label.

Usage

```
get_prepositions(
  labels,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE,
  prepositions = RCLabels::prepositions_list
)
```

Arguments

`labels`	The row and column labels from which prepositional phrases are to be extracted.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.
`prepositions`	A vector of strings to be treated as prepositions. Note that a space is appended to each word internally, so, e.g., "to" becomes "to ". Default is `RCLabels::prepositions_list`.

Value

A list of prepositions.

Examples

```
get_prepositions(c("a [of b into c]", "d [-> e of f]"))
```

`infer_notation` *Infer the notation(s) for a row or column label*

Description

It is convenient to know which notation is applicable to row or column labels. This function infers which `notations` are appropriate for `x`.

Usage

```
infer_notation(
  x,
  inf_notation = TRUE,
  notations = RCLabels::notations_list,
  allow_multiple = FALSE,
  retain_names = FALSE,
  choose_most_specific = TRUE,
  must_succeed = TRUE
)
```

Arguments

`x`	A row or column label (or vector of labels).
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`.
`notations`	A list of notations from which matches will be inferred. This function might not work as expected if `notation` is not a list. If `notation` is not a list, `notations` is returned in full. Default is `RCLabels::notations_list`.
`allow_multiple`	A boolean that tells whether multiple notation matches are allowed. If `FALSE` (the default), multiple matches give an error.
`retain_names`	A boolean that tells whether to retain names from `notations` on the outgoing matches. Default is `FALSE`. If `TRUE`, the return value is *always* a named list. If only one of `notations` is returned (for example, because `choose_most_specific = TRUE`), names are never supplied.
`choose_most_specific`	A boolean that indicates whether the most-specific notation will be returned when more than one of `notations` matches `x` and `allow_multiple = FALSE`. When `FALSE`, the first matching notation in `notations` is returned when `allow_multiple = FALSE`. Default is `TRUE`. See details.
`must_succeed`	A boolean that if `TRUE` (the default), causes an error to be thrown if a matching notation is not found for any label in `x`. When `FALSE`, an unsuccessful notation inference will return `NULL`.

Details

This function is vectorized. Thus, `x` can be a vector, in which case the output is a list of notations.

`notations` is treated as a store from which matches for each label in `x` can be determined. `notations` should be a named list of notations. When `retain_names = TRUE`, the names on `notations` will be retained, and the return value is *always* a list.

By default (`allow_multiple = FALSE`), a single notation object is returned for each item in `x` if only one notation in `notations` is appropriate for `x`. If `allow_multiple = FALSE` (the default) and more than one `notation` is applicable to `x`, an error is thrown. Multiple matches can be returned when `allow_multiple = TRUE`.

If multiple notations are matched, the return value is a list.

When `choose_most_specific = TRUE` (the default), the most specific notation in `notations` is returned. "Most specific" is defined as the matching notation whose sum of characters in the `pref_start`, `pref_end`, `suff_start` and `suff_end` elements is greatest. If `choose_most_specific = TRUE` and two matching notations in `notations` have the same number of characters, only the first match is returned. When `choose_most_specific = TRUE`, the value of `allow_multiple` no longer matters. `allow_multiple = FALSE` is implied and at most one of the `notations` will be returned.

When `inf_notation = FALSE` (default is `TRUE`), `notations` are returned unmodified, essentially disabling this function. Although calling with `inf_notation = FALSE` seems daft, this behavior enables cleaner code elsewhere.

Value

A single notation object (if `x` is a single row or column label) or a list of notation objects (if `x` is a vector or a list). If no `notations` match `x`, `NULL` is returned, either alone or in a list.

Examples

```
# Does not match any notations in RCLabels::notations_list
# and throws an error, because the default value for `must_succeed`
# is `TRUE`.
## Not run:
infer_notation("abc")

## End(Not run)
# This returns `NULL`, because `must_succeed = FALSE`.
infer_notation("abc", must_succeed = FALSE)
# This succeeds, because the label is in the form of a
# notation in `RCLabels::notation_list`,
# the default value of the `notation` argument.
infer_notation("a -> b")
# Names of the notations can be retained, in which case
# the return value is always a list.
infer_notation("a -> b", retain_names = TRUE)
# This function is vectorized.
# The list of labels matches
# all known notations in `RCLabels::notations_list`.
infer_notation(c("a -> b", "a (b)", "a [b]", "a [from b]", "a [of b]",
                 "a [to b]", "a [in b]", "a [-> b]", "a.b"),
                 retain_names = TRUE)
# By default, the most specific notation is returned.
# But when two or more matches are present,
# multiple notations can be returned, too.
infer_notation("a [from b]",
               allow_multiple = TRUE, retain_names = TRUE,
               choose_most_specific = FALSE)
infer_notation(c("a [from b]", "c [to d]"),
               allow_multiple = TRUE, retain_names = TRUE,
               choose_most_specific = FALSE)
# As shown above, "a \[from b\]" matches 2 notations:
# `RCLabels::bracket_notation` and `RCLabels::from_notation`.
# The default value for the notation argument is
# RCLabels::notations_list,
# which includes `RCLabels::bracket_notation`
# and `RCLabels::from_notation` in that order.
# Thus, there is some flexibility to how this function works
# if the value of the `notation` argument is a list of notations
# ordered from least specific to most specific,
# as `RCLabels::notations_list` is ordered.
# To review, the next call returns both `RCLabels::bracket_notation` and
# `RCLabels::from_notation`, because `allow_multiple = TRUE` and
# `choose_most_specific = FALSE`, neither of which are default.
infer_notation("a [from b]",
               allow_multiple = TRUE,
               choose_most_specific = FALSE,
               retain_names = TRUE)
# The next call returns `RCLabels::from_notation`, because
# the most specific notation is requested, and
# `RCLabels::from_notation` has more characters in its specification than
# `RCLabels::bracket_notation`.
infer_notation("a [from b]",
               choose_most_specific = TRUE,
               retain_names = TRUE)
# The next call returns the `RCLabels::bracket_notation`, because
# `choose_most_specific = FALSE`, and the first matching
# notation in `RCLabels::notations_list` is `RCLabels::bracket_notation`.
infer_notation("a [from b]",
               choose_most_specific = FALSE,
               retain_names = TRUE)
```

`infer_notation_for_one_label` *Infer the notation from one row or column label*

Description

This is a non-public helper function for vectorized `infer_notation()`.

Usage

```
infer_notation_for_one_label(
  x,
  inf_notation = TRUE,
  notations = RCLabels::notations_list,
  allow_multiple = FALSE,
  retain_names = FALSE,
  choose_most_specific = TRUE,
  must_succeed = TRUE
)
```

Arguments

`x`	A single row or column label.
`inf_notation`	A boolean that tells whether to infer notation for `x`.

`notations`	A list of notations from which matches will be inferred This function might not work as expected if `notation` is not a list. If `notation` is not a list, `notations` is returned in full. Default is `RCLabels::notations_list`.
`allow_multiple`	A boolean that tells whether multiple notation matches are allowed. If `FALSE` (the default), multiple matches give an error.
`retain_names`	A boolean that tells whether to retain names on the outgoing matches. Default is `FALSE`. If `TRUE`, the return value is a named list. If only one of `notations` is returned, names are never supplied.
`choose_most_specific`	A boolean that indicates if the most-specific notation will be returned when more than one of `notations` matches `x`. Default is `TRUE`.
`must_succeed`	A boolean that if `TRUE` (the default), causes an error to be thrown if a matching notation is not found for any label in `x`. When `FALSE`, an unsuccessful label inference will return `NULL`.

Value

A single matching notation object (if `allow_multiple = FALSE`, the default) or possibly multiple matching notation objects (if `allow_multiple = TRUE`). If no `notations` match `x`, `NULL`.

in_notation *In notation*

Description

A description of in notation.

Usage

```
in_notation
```

Format

A vector of notational symbols that provides to ("a [in b]") notation.

Examples

```
in_notation
```

make_list — *Make a list of items in x, regardless of x's type*

Description

Repeats x as necessary to make n of them. Does not try to simplify x.

Usage

```
make_list(x, n, lenx = ifelse(is.vector(x), length(x), 1))
```

Arguments

x	The object to be duplicated.
n	The number of times to be duplicated.
lenx	The length of item x. Be default, lenx is taken to be length(x),

Details

If x is itself a vector or list, you may want to override the default value for lenx. For example, if x is a list that should be duplicated several times, set lenx = 1.

Value

A list of x duplicated n times

Examples

```
m <- matrix(c(1:6), nrow=3, dimnames = list(c("r1", "r2", "r3"), c("c2", "c1")))
make_list(m, n = 1)
make_list(m, n = 2)
make_list(m, n = 5)
make_list(list(c(1,2), c(1,2)), n = 4)
m <- matrix(1:4, nrow = 2)
l <- list(m, m+100)
make_list(l, n = 4)
make_list(l, n = 1) # Warning because l is trimmed.
make_list(l, n = 5) # Warning because length(l) (i.e., 2) not evenly divisible by 5
make_list(list(c("r10", "r11"), c("c10", "c11")), n = 2) # Confused by x being a list
make_list(list(c("r10", "r11"), c("c10", "c11")), n = 2, lenx = 1) # Fix by setting lenx = 1
```

`make_or_pattern` *Create "or" regex patterns*

Description

This function makes "or" regex patterns from vectors or lists of strings. This function can be used with the `matsbyname::select_rows_byname()` and `matsbyname::select_cols_byname` functions. `make_or_pattern()` correctly escapes special characters in `strings`, such as `(` and `)`, as needed. Thus, it is highly recommended that `make_or_pattern` be used when constructing patterns for row and column selections with `matsbyname::select_rows_byname()` and `matsbyname::select_cols_byname()`.

Usage

```
make_or_pattern(
  strings,
  pattern_type = c("exact", "leading", "trailing", "anywhere", "literal")
)
```

Arguments

`strings`	A vector of row and column names.
`pattern_type`	One of "exact", "leading", "trailing", "anywhere", or "literal". Default is "exact".

Details

`pattern_type` controls the type of pattern created:

- `exact` produces a regex pattern that selects row or column names by exact match.
- `leading` produces a regex pattern that selects row or column names if the item in `strings` matches the beginnings of row or column names.
- `trailing` produces a regex pattern that selects row or column names if the item in `strings` matches the ends of row or column names.
- `anywhere` produces a regex pattern that selects row or column names if the item in `strings` matches any substring of row or column names.
- `literal` returns `strings` unmodified, and it is up to the caller to formulate a correct regex.

Value

An "or" regex pattern suitable for selecting row and column names. Amenable for use with `matsbyname::select_rows_byna` or `matsbyname::select_cols_byname`.

Examples

```
make_or_pattern(strings = c("a", "b"), pattern_type = "exact")
make_or_pattern(strings = c("a", "b"), pattern_type = "leading")
make_or_pattern(strings = c("a", "b"), pattern_type = "trailing")
make_or_pattern(strings = c("a", "b"), pattern_type = "anywhere")
make_or_pattern(strings = c("a", "b"), pattern_type = "literal")
```

modify_label_pieces *Modify pieces of row and column labels*

Description

Typical `pieces` include "noun" or a preposition, such as "in" or "from". See `RCLabels::prepositions` for additional examples. This argument may be a single string or a character vector.

Usage

```
modify_label_pieces(
  labels,
  piece,
  mod_map,
  prepositions = RCLabels::prepositions_list,
  inf_notation = TRUE,
  notation = RCLabels::bracket_notation,
  choose_most_specific = FALSE
)
```

Arguments

`labels`	A vector of row or column labels in which pieces will be modified.
`piece`	The piece (or pieces) of the row or column label that will be modified.
`mod_map`	A modification map. See details.
`prepositions`	A list of prepositions, used to detect prepositional phrases. Default is `RCLabels::prepositions_list`.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether the most specific notation is selected when more than one notation match. Default is `FALSE`.

Details

This function modifies pieces of row and column labels according to `label_map` that defines "one or many to one" relationships. This function is useful for aggregations. For example, replacing nouns can be done by `modify_label_pieces(labels, piece = "noun", label_map = list(new_noun = c("a", "b", "c"))`. The string "new_noun" will replace any of "a", "b", or "c" when they appear as nouns in a row or column label. See examples for details.

The `mod_map` argument should consist of a named list of character vectors in which names indicate strings to be inserted and values indicate values that should be replaced. The sense is `new = old` or `new = olds`, where "new" is the new name (the replacement) and "old"/"olds" is/are a string/vector of strings, any one of which will be replaced by "new".

Note `piece` can be "pref"/"suff" or "noun"/"prepositions" If any `piece` is "pref" or "suff", all pieces are assumed to be a prefix or a suffix. If non of the `pieces` are "pref" or "suff", all `pieces` are assumed to be nouns or prepositions, such as "in" or "from". See `RCLabels::prepositions` for additional examples. This argument may be a single string or a character vector.

Value

`labels` with replacements according to `piece` and `mod_map`.

Examples

```
# Simple case
modify_label_pieces("a [of b in c]",
                    piece = "noun",
                    mod_map = list(new_noun = c("a", "b")))
# Works with a vector or list of labels
modify_label_pieces(c("a [of b in c]", "d [-> e in f]"),
                    piece = "noun",
                    mod_map = list(new_noun = c("d", "e")))
# Works with multiple items in the mod_map
modify_label_pieces(c("a [of b in c]", "d [-> e in f]"),
                    piece = "noun",
                    mod_map = list(new_noun1 = c("a", "b", "c"),
                                   new_noun2 = c("d", "e", "f")))
# Works with multiple pieces to be modified
modify_label_pieces(c("a [of b in c]", "d [-> e in f]"),
                    piece = c("noun", "in"),
                    mod_map = list(new_noun = c("a", "b", "c"),
                                   new_in   = c("c", "f")))
```

`modify_nouns` *Modify nouns in labels*

Description

This function modifies the nouns of row and column labels. The length of `new_nouns` must be the same as the length of `labels`.

Usage

```
modify_nouns(
  labels,
  new_nouns,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE
)
```

Arguments

`labels`	The row and column labels in which the nouns will be modified.
`new_nouns`	The new nouns to be set in `labels`. Must be same length as `labels`.
`inf_notation`	A boolean that tells whether to infer notation for `labels`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.

Value

A character vector of same length as labels with nouns modified to be `new_nouns`.

Examples

```
labels <- c("a [of b in c]", "d [of e in USA]")
modify_nouns(labels, c("a_plus", "g"))
```

`notations_list` *Notations list*

Description

A list of all bundled notations. This list is organized by least specific to most specific, thereby enabling some unique behaviors in `infer_notation()`. See the examples for `infer_notation()`.

Usage

```
notations_list
```

Format

A list of bundled notations.

Examples

```
notations_list
```

`of_notation` *Of notation*

Description

A description of of notation.

Usage

```
of_notation
```

Format

A vector of notational symbols that provides of ("a [of b]") notation.

Examples

```
of_notation
```

`paren_notation` *Parenthetical notation*

Description

A description of parenthetical notation.

Usage

```
paren_notation
```

Format

A vector of notational symbols that provides a parenthetical ("a (b)") notation.

Examples

```
paren_notation
```

`paste_noun_pp` *Recombine row and column labels*

Description

This function recombines (unsplits) row or column labels that have been separated by `split_noun_pp()`.

Usage

```
paste_noun_pp(
  splt_labels,
  notation = RCLabels::bracket_notation,
  squish = TRUE
)
```

Arguments

`splt_labels`	A vector of split row or column labels, probably created by `split_noun_pp()`.
`notation`	The notation object that describes the labels. Default is `RCLabels::bracket_notation`.
`squish`	A boolean that tells whether to remove extra spaces in the output of `paste_*()` functions. Default is `TRUE`.

Value

Recombined row and column labels.

Examples

```
labs <- c("a [of b in c]", "d [from Coal mines in USA]")
labs
split <- split_noun_pp(labs)
split
paste_noun_pp(split)
# Also works in a data frame
df <- tibble::tibble(labels = c("a [in b]", "c [of d into USA]",
                                "e [of f in g]", "h [-> i in j]"))
recombined <- df %>%
  dplyr::mutate(
    splits = split_noun_pp(labels),
    recombined = paste_noun_pp(splits)
  )
all(recombined$labels == recombined$recombined)
```

`prepositions` *Prepositions*

Description

This constant is deprecated. Please use `prepositiions_list` instead.

Usage

```
prepositions
```

Format

A vector of prepositions used in row and column labels.

`prepositions_list` *Prepositions*

Description

Prepositions used in row and column labels.

Usage

```
prepositions_list
```

Format

A vector of prepositions used in row and column labels.

Examples

```
prepositions_list
```

regex_funcs — *Find or replace row or column labels that match a regular expression*

Description

`match_by_pattern()` tells whether row or column labels match a regular expression. Internally, `grepl()` decides whether a match occurs. `replace_by_pattern()` replaces portions of row of column labels when a regular expression is matched. Internally, `gsub()` performs the replacements.

Usage

```
match_by_pattern(
  labels,
  regex_pattern,
  pieces = "all",
  prepositions = RCLabels::prepositions_list,
  notation = RCLabels::bracket_notation,
  ...
)

replace_by_pattern(
  labels,
  regex_pattern,
  replacement,
  pieces = "all",
  prepositions = RCLabels::prepositions_list,
  notation = RCLabels::bracket_notation,
  ...
)
```

Arguments

`labels`	The row and column labels to be modified.
`regex_pattern`	The regular expression pattern to determine matches and replacements. Consider using `Hmisc::escapeRegex()` to escape `regex_pattern` before calling this function.
`pieces`	The pieces of row or column labels to be checked for matches or replacements. See details.
`prepositions`	A vector of strings that count as prepositions. Default is `RCLabels::prepositions_list`. Used to detect prepositional phrases if `pieces` are to be interpreted as prepositions.
`notation`	The notation used in `labels`. Default is `RCLabels::bracket_notation`.
`...`	Other arguments passed to `grepl()` or `gsub()`, such as `ignore.case`, `perl`, `fixed`, or `useBytes`. See examples.
`replacement`	For `replace_by_pattern()`, the string that replaces all matches to `regex_pattern`.

Details

By default (pieces = "all"), complete labels (as strings) are checked for matches and replacements. If pieces == "pref" or pieces == "suff", only the prefix or the suffix is checked for matches and replacements. Alternatively, pieces = "noun" or pieces = <<preposition>> indicate that only specific pieces of labels are to be checked for matches and replacements. When pieces = <<preposition>>, only the object of <<preposition>> is checked for matches and replacement.

pieces can be a vector, indicating multiple pieces to be checked for matches and replacements. But if any of the pieces are "all", all pieces are checked and replaced. If pieces is "pref" or "suff", only one can be specified.

Value

A logical vector of same length as labels, where TRUE indicates a match was found and FALSE indicates otherwise.

Examples

```
labels <- c("Production [of b in c]", "d [of Coal in f]", "g [of h in USA]")
# With default `pieces` argument, matching is done for whole labels.
match_by_pattern(labels, regex_pattern = "Production")
match_by_pattern(labels, regex_pattern = "Coal")
match_by_pattern(labels, regex_pattern = "USA")
# Check beginnings of labels
match_by_pattern(labels, regex_pattern = "^Production")
# Check at ends of labels: no match.
match_by_pattern(labels, regex_pattern = "Production$")
# Can match on nouns or prepositions.
match_by_pattern(labels, regex_pattern = "Production", pieces = "noun")
# Gives FALSE, because "Production" is a noun.
match_by_pattern(labels, regex_pattern = "Production", pieces = "in")
```

---

remove_label_pieces *Remove a prepositional phrase in a row or column label*

Description

This function removes pieces from row and column labels.

Usage

```
remove_label_pieces(
  labels,
  pieces_to_remove,
  prepositions = RCLabels::prepositions_list,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE
)
```

Arguments

`labels`	The row and column labels from which prepositional phrases will be removed.
`pieces_to_remove`	The names of pieces of the label to be removed, typically "noun" or a preposition such as "of" or "in" See `RCLabels::prepositions_list` for a list of known prepositions.
`prepositions`	A list of prepositions, used to detect prepositional phrases. Default is `RCLabels::prepositions_list`.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether the most specific notation is selected when more than one notation match. Default is `FALSE`.

Value

`labels` with pieces removed.

Examples

```
labs <- c("a [of b in c]", "d [-> e in f]")
remove_label_pieces(labs, pieces_to_remove = "of")
remove_label_pieces(labs, pieces_to_remove = c("of", "->"))
remove_label_pieces(labs, pieces_to_remove = c("in", "into"))
remove_label_pieces(labs, pieces_to_remove = c("of", "in"))
```

`row-col-notation` *Row and column notation*

Description

It is often convenient to represent matrix row and column names with notation that includes a prefix and a suffix, with corresponding separators or start-end string sequences. There are several functions to generate specialized versions or otherwise manipulate row and column names on their own or as row or column names.

- `flip_pref_suff()` Switches the location of prefix and suffix, such that the prefix becomes the suffix, and the suffix becomes the prefix. E.g., "a -> b" becomes "b -> a" or "a [b]" becomes "b [a]".
- `get_pref_suff()` Selects only prefix or suffix, discarding notational elements and the rejected part. Internally, this function calls `split_pref_suff()` and selects only the desired portion.
- `notation_vec()` Builds a vector of notation symbols in a standard format. By default, it builds a list of notation symbols that provides an arrow separator (" -> ") between prefix and suffix.

- paste_pref_suff() paste0's prefixes and suffixes, the inverse of split_pref_suff(). Always returns a character vector.
- preposition_notation() Builds a list of notation symbols that provides (by default) square brackets around the suffix with a preposition ("prefix [preposition suffix]").
- split_pref_suff() Splits prefixes from suffixes, returning each in a list with names pref and suff. If no prefix or suffix delimiters are found, x is returned in the pref item, unmodified, and the suff item is returned as "" (an empty string). If there is no prefix, and empty string is returned for the pref item. If there is no suffix, and empty string is returned for the suff item.
- switch_notation() Switches from one type of notation to another based on the from and to arguments. Optionally, prefix and suffix can be flipped.

Parts of a notation vector are "pref_start", "pref_end", "suff_start", and "suff_end". None of the strings in a notation vector are considered part of the prefix or suffix. E.g., "a -> b" in arrow notation means that "a" is the prefix and "b" is the suffix. If sep only is specified for notation_vec() (default is " -> "), pref_start, pref_end, suff_start, and suff_end are set appropriately.

For functions where the notation argument is used to identify portions of the row or column label (such as split_pref_suff(), get_pref_suff(), and the from argument to switch_notation()), (Note: flip_pref_suff() cannot infer notation, because it switches prefix and suffix in a known, single notation.) if notation is a list, it is treated as a store from which the most appropriate notation is inferred by infer_notation(choose_most_specific = TRUE). Because default is RCLabels::notations_list, notation is inferred by default. The argument choose_most_specific tells what to do when two notations match a label: if TRUE (the default), the notation with most characters is selected. If FALSE, the first matching notation in notation will be selected. See details at infer_notation().

If specifying more than one notation, be sure the notations are in a list. notation = c(RCLabels::bracket_notation, RCLabels::arrow_notation) is unlikely to produce the desired result, because the notations are concatenated together to form a long string vector. Rather say notation = list(RCLabels::bracket_notation, RCLabels::arrow_notation).

For functions that construct labels (such as paste_pref_suff()), notation can be a list of notations over which the paste tasks is mapped. If notation is a list, it must have as many items as there are prefix/suffix pairs to be pasted.

If either pref or suff are a zero-length character vector (essentially an empty character vector such as obtained from character()) input to paste_pref_suff(), an error is thrown. Instead, use an empty character string (such as obtained from "").

Usage

```
notation_vec(
  sep = " -> ",
  pref_start = "",
  pref_end = "",
  suff_start = "",
  suff_end = ""
)

preposition_notation(preposition, suff_start = " [", suff_end = "]")
```

```
split_pref_suff(
  x,
  transpose = FALSE,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = TRUE
)

paste_pref_suff(
  ps = list(pref = pref, suff = suff),
  pref = NULL,
  suff = NULL,
  notation = RCLabels::arrow_notation,
  squish = TRUE
)

flip_pref_suff(
  x,
  notation = RCLabels::notations_list,
  inf_notation = TRUE,
  choose_most_specific = TRUE
)

get_pref_suff(
  x,
  which = c("pref", "suff"),
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = TRUE
)

switch_notation(
  x,
  from = RCLabels::notations_list,
  to,
  flip = FALSE,
  inf_notation = TRUE
)
```

Arguments

`sep`	A string separator between prefix and suffix. Default is " -> ".
`pref_start`	A string indicating the start of a prefix. Default is `NULL`.
`pref_end`	A string indicating the end of a prefix. Default is the value of `sep`.
`suff_start`	A string indicating the start of a suffix. Default is the value of `sep`.
`suff_end`	A string indicating the end of a suffix. Default is `NULL`.

`preposition`	A string used to indicate position for energy flows, typically "from" or "to" in different notations.
`x`	A string or vector of strings to be operated upon.
`transpose`	A boolean that tells whether to `purrr::transpose()` the result. Set `transpose = TRUE` when using `split_pref_suff()` in a `dplyr::mutate()` call in the context of a data frame. Default is `FALSE`.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	A notation vector generated by one of the `*_notation()` functions, such as `notation_vec()`, `arrow_notation`, or `bracket_notation`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from the `notation` argument when the `notation` argument is a list.
`ps`	A list of prefixes and suffixes in which each item of the list is itself a list with two items named `pref` and `suff`.
`pref`	A string or list of strings that are prefixes. Default is `NULL`.
`suff`	A string of list of strings that are suffixes. Default is `NULL`.
`squish`	A boolean that tells whether to remove extra spaces in the output of `paste_*()` functions. Default is `TRUE`.
`which`	Tells which to keep, the prefix ("pref") or the suffix ("suff").
`from`	The `notation` to switch *away from*.
`to`	The `notation` to switch *to*.
`flip`	A boolean that tells whether to also flip the notation. Default is `FALSE`.

Value

For `notation_vec()`, `arrow_notation`, and `bracket_notation`, a string vector with named items `pref_start`, `pref_end`, `suff_start`, and `suff_end`; For `split_pref_suff()`, a string list with named items `pref` and `suff`. For `paste_pref_suff()`, `split_pref_suff()`, and `switch_notation()`, a string list in notation format specified by various `notation` arguments, including `from`, and `to`. For `keep_pref_suff`, one of the prefix or suffix or a list of prefixes or suffixes.

Examples

```
notation_vec()
arrow_notation
bracket_notation
split_pref_suff("a -> b", notation = arrow_notation)
# Or infer the notation (by default from notations_list)
split_pref_suff("a -> b")
split_pref_suff(c("a -> b", "c -> d", "e -> f"))
split_pref_suff(c("a -> b", "c -> d", "e -> f"), transpose = TRUE)
flip_pref_suff("a [b]", notation = bracket_notation)
# Infer notation
flip_pref_suff("a [b]")
get_pref_suff("a -> b", which = "suff")
```

```
switch_notation("a -> b", from = arrow_notation, to = bracket_notation)
# Infer notation and flip prefix and suffix
switch_notation("a -> b", to = bracket_notation, flip = TRUE)
# Also works for vectors
switch_notation(c("a -> b", "c -> d"),
                from = arrow_notation,
                to = bracket_notation)
# Functions can infer the correct notation and return multiple matches
infer_notation("a [to b]",
               allow_multiple = TRUE,
               choose_most_specific = FALSE)
# Or choose the most specific notation
infer_notation("a [to b]",
               allow_multiple = TRUE,
               choose_most_specific = TRUE)
# When setting the from notation, only that type of notation will be switched
switch_notation(c("a -> b", "c [to d]"),
                from = arrow_notation,
                to = bracket_notation)
# But if notations are inferred, all notations can be switched
switch_notation(c("a -> b", "c [to d]"), to = bracket_notation)
# A double-switch can be accomplished.
# In this first example, `RCLabels::first_dot_notation` is inferred.
switch_notation("a.b.c", to = arrow_notation)
# In this second example,
# it is easier to specify the `from` and `to` notations.
switch_notation("a.b.c", to = arrow_notation) %>%
  switch_notation(from = first_dot_notation, to = arrow_notation)
# "" can be used as an input
paste_pref_suff(pref = "a", suff = "", notation = RCLabels::from_notation)
```

`split_noun_pp` *Split row and column labels into nouns and prepositional phrases*

Description

This function is similar to `split_pref_suff()` in that it returns a list. However, this function's list is more detailed than `split_pref_suff()`. The return value from this function is a list with the first named item being the prefix (with the name noun) followed by objects of prepositional phrases (with names being prepositions that precede the objects).

Usage

```
split_noun_pp(
  labels,
  inf_notation = TRUE,
  notation = RCLabels::notations_list,
  choose_most_specific = FALSE,
  prepositions = RCLabels::prepositions_list
)
```

Arguments

`labels`	The row and column labels from which prepositional phrases are to be extracted.
`inf_notation`	A boolean that tells whether to infer notation for `x`. Default is `TRUE`. See `infer_notation()` for details.
`notation`	The notation type to be used when extracting prepositions. Default is `RCLabels::notations_list`, meaning that the notation is inferred using `infer_notation()`.
`choose_most_specific`	A boolean that tells whether to choose the most specific notation from `notation` when inferring notation. Default is `FALSE` so that a less specific notation can be inferred. In combination with `RCLabels::notations_list`, the default value of `FALSE` means that `RCLabels::bracket_notation` will be selected instead of anything more specific, such as `RCLabels::from_notation`.
`prepositions`	A vector of strings to be treated as prepositions. Note that a space is appended to each word internally, so, e.g., "to" becomes "to ". Default is `RCLabels::prepositions_list`.

Details

Unlike `split_pref_suff()`, it does not make sense to have a `transpose` argument on `split_noun_pp()`. Labels may not have the same structure, e.g., they may have different prepositions.

Value

A list of lists with items named `noun` and `pp`.

Examples

```
# Specify the notation
split_noun_pp(c("a [of b in c]", "d [of e into f]"),
              notation = bracket_notation)
# Infer the notation via default arguments
split_noun_pp(c("a [of b in c]", "d [of e into f]"))
```

`strip_label_part` *A convenience function to help splitting prefixes and suffixes*

Description

This function should only ever see a single label (`x`) and a single `notation`.

Usage

```
strip_label_part(x, notation, part, pattern_pref = "", pattern_suff = "")
```

Arguments

`x`	The label(s) to be split.
`notation`	The notations to be used for each `x`.
`part`	The part of the label to work on, such as "pref_start", "pref_end", "suff_start", or "suff_end".
`pattern_pref`	The prefix to a regex pattern to be used in `gsub()`.
`pattern_suff`	The suffix to a regex pattern to be used in `gsub()`.

Details

If `notation` is `NULL`, `x` is returned, unmodified.

Value

A label shorn of the part to be stripped.

`to_notation` *To notation*

Description

A description of to notation.

Usage

```
to_notation
```

Format

A vector of notational symbols that provides to ("a [to b]") notation.

Examples

```
to_notation
```

Index