Package ‘labelmachine’

October 11, 2019

Title Make Labeling of R Data Sets Easy

Version 1.0.0

Description Assign meaningful labels to data frame columns. 'labelmachine' manages your label assignment rules in 'yaml' files and makes it easy to use the same labels in multiple projects.

Depends R (>= 3.5.0)

Imports yaml (>= 2.2.0)

Suggests testthat (>= 2.1.0), roxygen2 (>= 6.1.1), magrittr (>= 1.5), rlang (>= 0.4.0), covr, knitr, rmarkdown

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 6.1.1

License GPL-3


BugReports https://github.com/a-maldet/labelmachine/issues

Collate 'composerr.R' 'imports.R' 'utilities.R' 'lama_dictionary.R' 'lama_merge.R' 'lama_mutate.R' 'lama_read.R' 'lama_select.R' 'lama_rename.R' 'lama_translate.R' 'lama_translate_all.R' 'lama_write.R' 'lappli.R'

NeedsCompilation no

Author Adrian Maldet [aut, cre]

Maintainer Adrian Maldet <maldet@posteo.at>

Repository CRAN

Date/Publication 2019-10-11 07:30:03 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.lama_dictionary</td>
<td>3</td>
</tr>
<tr>
<td>check_and_translate_all</td>
<td>6</td>
</tr>
<tr>
<td>check_and_translate_df</td>
<td>7</td>
</tr>
<tr>
<td>check_and_translate_df_</td>
<td>8</td>
</tr>
<tr>
<td>check_and_translate_vector</td>
<td>9</td>
</tr>
<tr>
<td>check_and_translate_vector_</td>
<td>10</td>
</tr>
<tr>
<td>check_arguments</td>
<td>11</td>
</tr>
<tr>
<td>check_rename</td>
<td>12</td>
</tr>
<tr>
<td>check_select</td>
<td>12</td>
</tr>
<tr>
<td>composerr_</td>
<td>13</td>
</tr>
<tr>
<td>contains_na_escape</td>
<td>14</td>
</tr>
<tr>
<td>dictionary_to_yaml</td>
<td>14</td>
</tr>
<tr>
<td>escape_to_na</td>
<td>15</td>
</tr>
<tr>
<td>is.lama_dictionary</td>
<td>15</td>
</tr>
<tr>
<td>is.syntactic</td>
<td>16</td>
</tr>
<tr>
<td>lama_get</td>
<td>17</td>
</tr>
<tr>
<td>lama_merge</td>
<td>19</td>
</tr>
<tr>
<td>lama_mutate</td>
<td>20</td>
</tr>
<tr>
<td>lama_read</td>
<td>22</td>
</tr>
<tr>
<td>lama_rename</td>
<td>22</td>
</tr>
<tr>
<td>lama_select</td>
<td>24</td>
</tr>
<tr>
<td>lama_translate</td>
<td>25</td>
</tr>
<tr>
<td>lama_translate_all</td>
<td>31</td>
</tr>
<tr>
<td>lama_write</td>
<td>33</td>
</tr>
<tr>
<td>lappI</td>
<td>34</td>
</tr>
<tr>
<td>named_lapply</td>
<td>35</td>
</tr>
<tr>
<td>NA_lama_</td>
<td>35</td>
</tr>
<tr>
<td>na_to_escape</td>
<td>36</td>
</tr>
<tr>
<td>new_lama_dictionary</td>
<td>36</td>
</tr>
<tr>
<td>print.lama_dictionary</td>
<td>39</td>
</tr>
<tr>
<td>rename_translation</td>
<td>40</td>
</tr>
<tr>
<td>stringify</td>
<td>40</td>
</tr>
<tr>
<td>translate_df</td>
<td>41</td>
</tr>
<tr>
<td>translate_vector</td>
<td>42</td>
</tr>
<tr>
<td>validate_lama_dictionary</td>
<td>43</td>
</tr>
<tr>
<td>validate_translation</td>
<td>44</td>
</tr>
<tr>
<td>yaml_to_dictionary</td>
<td>45</td>
</tr>
</tbody>
</table>

**Index** 46
as.lama_dictionary  Coerce to a lama_dictionary class object

Description

This function allows two types of arguments:

- **named list**: A named list object holding the translations.
- **data.frame**: A data.frame with one or more column pairs. Each column pair consists of a column holding the original values, which should be replaced, and a second character column holding the new labels which should be assigned to the original values. Use the arguments `col_old` and `col_new` in order to define which columns are holding original values and which columns hold the new labels. The names of the resulting translations are defined by a character vector given in argument `translation`. Furthermore, each translation can have a different ordering which can be configured by a character vector given in argument `ordering`.

Usage

```r
as.lama_dictionary(.data, ...)  
## S3 method for class 'list'
as.lama_dictionary(.data, ...)

## S3 method for class 'lama_dictionary'
as.lama_dictionary(.data, ...)

## Default S3 method:
as.lama_dictionary(.data = NULL, ...)

## S3 method for class 'data.frame'
as.lama_dictionary(.data, translation, col_old,  
col_new, ordering = rep("row", length(translation)), ...)
```

Arguments

- `.data` An object holding the translations. `.data` can be of the following data types:
  - **named list**: A named list object, where each list entry is a translation (a named character vector)
  - **data.frame**: A data.frame holding one or more column pairs, where each column pair consists of one column holding the original variable values and a second column holding the new labels, which should be assigned to the original values.
- `...` Various arguments, depending on the data type of `.data`.
- `translation` A character vector holding the names of all translations
col_old

This argument is only used, if the argument given in .data is a data.frame. In this case, the argument col_old must be a character vector (same length as translation) holding the names of the columns in the data.frame (in the argument .data) which hold the original variable values. These columns can be of any type: character, logical, numerical or factor.

col_new

This argument is only used, if the argument given in .data is a data.frame. In this case, the argument col_old must be a character vector (same length as translation) holding the names of the columns in the data.frame (in the argument .data) which hold the new labels, which should be assigned to the original values. These columns can be character vectors or factors with character labels.

ordering

This argument is only used, if the argument given in .data is a data.frame. In this case, the argument ordering must be a character vector (same length as translation) holding one of the following configuration strings configuring the ordering of each corresponding translation:

- "row": The corresponding translation will be ordered exactly in the same way as the rows are ordered in the data.frame .data.
- "old": The corresponding translation will be ordered by the given original values which are contained in the corresponding column col_old. If the column contains a factor variable, then the ordering of the factor will be used. If it just contains a plain character variable, then it will be ordered alphanumerically.
- "new": The corresponding translation will be ordered by the given new labels which are contained in the corresponding column col_new. If the column contains a factor variable, then the ordering of the factor will be used. If it just contains a plain character variable, then it will be ordered alphanumerically.

Value

A new lama_dictionary class object holding the passed in translations.

Translations

A translation is a named character vector of non zero length. This named character vector defines which labels (of type character) should be assigned to which values (can be of type character, logical or numeric) (e.g. the translation c("0" = "urban","1" = "rural") assigns the label "urban" to the value 0 and "rural" to the value 1, for example the variable x = c(0,0,1) is translated to x_new = c("urban","urban","rural"). Therefore, a translation (named character vector) contains the following information:

- The names of the character vector entries correspond to the original variable levels. Variables of types numeric or logical are turned automatically into a character vector (e.g. 0 and 1 are treated like "0" and "1").
- The entries (character strings) of the character vector correspond to the new labels, which will be assigned to the original variable levels. It is also allowed to have missing labels (NAs). In this case, the original values are mapped onto missing values.
The function `lama_translate()` is used in order to apply a translation on a variable. The resulting vector with the assigned labels can be of the following types:

- **character**: An unordered vector holding the new character labels.
- **factor** with character levels: An ordered vector holding the new character labels.

The original variable can be of the following types:

- **character** vector: This is the simplest case. The character values will be replaced by the corresponding labels.
- **numeric or logical** vector: Vectors of type numeric or logical will be turned into character vectors automatically before the translation process and then simply processed like in the character case. Therefore, it is sufficient to define the translation mapping for the character case, since it also covers the numeric and logical case.
- **factor** vector with levels of any type: When translating factor variables one can decide whether or not to keep the original ordering. Like in the other cases the levels of the factor variable will always be turned into character strings before the translation process.

### Missing values

It is also possible to handle missing values with `lama_translate()`. Therefore, the used translation must contain a information that tells how to handle a missing value. In order to define such a translation the missing value (NA) can be escaped with the character string "NA_". This can be useful in two situations:

- All missing values should be labeled (e.g. the translation `c("0" = "urban","1" = "rural",NA_ = "missing")` assigns the character string "missing" to all missing values of a variable).
- Map some original values to NA (e.g. the translation `c("0" = "urban","1" = "rural","2" = "NA_","3" = "NA_")` assigns NA (the missing character) to the original values 2 and 3). Actually, in this case the translation definition does not always have to use this escape mechanism, but only when defining the translations inside of a YAML file, since the YAML parser does not recognize missing values.

### lama_dictionary class objects

Each `lama_dictionary` class object can contain multiple translations, each with a unique name under which the translation can be found. The function `lama_translate()` uses a lama_dictionary class object to translate a normal vector or to translate one or more columns in a data.frame.

Sometimes it may be necessary to have different translations for the same variable, in this case it is best to have multiple translations with different names (e.g. `area_short = c("0" = "urb","1" = "rur")` and `area = c("0" = "urban","1" = "rural")`).

### Examples

```r
## Example-1: Initialize a lama-dictionary from a list object holding the translations

# Initialize a dictionary of country and language translations
obj <- list(
  country = c(uk = "United Kingdom", fr = "France", NA_ = "other countries"),
  language = c(en = "English", fr = "French")
)
```
check_and_translate_all

Check and translate function used by lama_translate_all() and lama_to_factor_all()

Description

Check and translate function used by lama_translate_all() and lama_to_factor_all()

Usage

check_and_translate_all(.data, dictionary, prefix, suffix, fn_colname, keep_order, to_factor, is_translated, err_handler)

Arguments

.data Either a data frame, a factor or a vector.
dictionary A lama_dictionary object, holding the translations for various variables.
prefix A character string, which is used as prefix for the new column names.
suffix A character string, which is used as suffix for the new column names.
fn_colname A function, which transforms character string into a new character string. This function will be used to transform the old column names into new column names under which the labeled variables will then be stored.
keep_order A logical of length one, defining if the original order (factor order or alphabetical order) of the data frame variables should be preserved.
check_and_translate_df

Description

Checks arguments and translate a data.frame

Usage

check_and_translate_df(.data, dictionary, args, keep_order, to_factor, is_translated, err_handler)

Arguments

.data Either a data frame, a factor or an atomic vector.
dictionary A lama_dictionary object, holding the translations for various variables.
args The list of arguments given in ... when calling lama_translate() or lama_to_factor()
keep_order A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector is TRUE, then the order of the original factor variable will be preserved.
to_factor A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
check_and_translate_df

A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If is_translated = TRUE, then the original variable is a character vector holding the right labels (character strings). In this case, the labels are left unchanged, but the variables are turned into factors with order given in the selected translations.

err_handler
An error handling function

---

**check_and_translate_df**

*Checks arguments and translate a data.frame (standard eval)*

**Description**

Checks arguments and translate a data.frame (standard eval)

**Usage**

check_and_translate_df(.data, dictionary, translation, col, col_new, keep_order, to_factor, is_translated, err_handler)

**Arguments**

- **.data** Either a data frame, a factor or an atomic vector.
- **dictionary** A lama_dictionary object, holding the translations for various variables.
- **translation** A character vector holding the names of the variable translations which should be used for assigning new labels to the variable. This names must be a subset of the translation names returned by names(dictionary).
- **col** Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names of the data.frame columns that should be relabeled. If omitted, then it will be assumed that the column names are the same as the given translation names in the argument translation.
- **col_new** Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names under which the relabeled variables should be stored in the data.frame. If omitted, then it will be assumed that the new column names are the same as the column names of the original variables.
- **keep_order** A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in ..., then the to each variable translation there is a corresponding boolean configuration. If a translated variable in the data.frame is a factor variable, and the corresponding boolean configuration is set to TRUE, then the the order of the original factor variable will be preserved.
check_and_translate_vector

Checks arguments and translate a vector

Description

Checks arguments and translate a vector

Usage

check_and_translate_vector(.data, dictionary, args, keep_order, to_factor, is_translated, err_handler)

Arguments

.data Either a data frame, a factor or an atomic vector.
dictionary A lama_dictionary object, holding the translations for various variables.
args The list of arguments given in ... when calling lama_translate() or lama_to_factor()
keep_order A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
to_factor A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
is_translated A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If is_translated = TRUE, then the original variable is a character vector holding the right labels (character strings). In this case, the labels are left unchanged, but the variables are turned into factors with order given in the selected translations.
err_handler An error handling function
is_translated  A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If is_translated = TRUE, then the original variable is a character vector holding the right labels (character strings). In this case, the labels are left unchanged, but the variables are turned into factors with order given in the selected translations.

err_handler  An error handling function

check_and_translate_vector_

Checks arguments and translate a character vector (standard eval)

Description

Checks arguments and translate a character vector (standard eval)

Usage

check_and_translate_vector_(.data, dictionary, translation, keep_order, to_factor, is_translated, err_handler)

Arguments

.data  Either a data frame, a factor or an atomic vector.
dictionary  A lama_dictionary object, holding the translations for various variables.
translation  A character vector holding the names of the variable translations which should be used for assigning new labels to the variable. This names must be a subset of the translation names returned by names(dictionary).
keep_order  A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If a translated variable in the data.frame is a factor variable, and the corresponding boolean configuration is set to TRUE, then the the order of the original factor variable will be preserved.
to_factor  A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
is_translated  A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If is_translated = TRUE, then the original variable is a character vector.
check_arguments

Function that applies some general checks to the arguments of lama_translate() and lama_translate()

Description

Function that applies some general checks to the arguments of lama_translate() and lama_translate().

Usage

check_arguments(.data, dictionary, col_new, keep_order, to_factor, err_handler)

Arguments

.data
Either a data frame, a factor or an atomic vector.
dictionary
A lama_dictionary object, holding the translations for various variables.
col_new
Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names under which the relabeled variables should be stored in the data.frame. If omitted, then it will be assumed that the new column names are the same as the column names of the original variables.
keep_order
A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If a translated variable in the data.frame is a factor variable, and the corresponding boolean configuration is set to TRUE, then the the order of the original factor variable will be preserved.
to_factor
A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
ero_handler
An error handling function
check_rename

Function that checks the passed in arguments for `lama_rename()` and `lama_rename_()`

Description

Function that checks the passed in arguments for `lama_rename()` and `lama_rename_()`

Usage

check_rename(.data, old, new, err_handler)

Arguments

- `.data` A `lama_dictionary` object, holding the variable translations
- `old` A character vector holding the names of the variable translations, that should be renamed.
- `new` A character vector holding the new names of the variable translations.
- `err_handler` A error handling function

check_select

Function that checks the passed in arguments for `lama_select()` and `lama_select_()`

Description

Function that checks the passed in arguments for `lama_select()` and `lama_select_()`

Usage

check_select(.data, key, err_handler)

Arguments

- `.data` A `lama_dictionary` object, holding the variable translations
- `key` A character vector holding the names of the variable translations, that should be renamed.
- `err_handler` A error handling function
**composerr_**

*Composerr_*: _Compose error handlers (concatenate error messages)*

**Description**

The functions `composerr()`, `composerr_()` and `composerr_parent()` modify error handlers by appending character strings to the error messages of the error handling functions:

- `composerr()` uses non-standard evaluation.
- `composerr_()` is the standard evaluation alternative of `composerr()`.
- `composerr_parent()` is a wrapper of `composerr()`, defining the parent environment as the lookup environment of the `err_handler`. This function looks up the prior error handling function in the parent environment of the current environment and allows you to store the modified error handling function under the same name as the error handling function from the parent environment without running into recursion issues. This is especially useful when doing error handling in nested environments (e.g. checking nested list objects) and you don’t want to use different names for the error handling functions in the nested levels. If you don’t have a nested environment situation, better use `composerr()` or `composerr_()`.

**Usage**

```r
composerr_(text_1 = NULL, err_prior = NULL, text_2 = NULL, 
sep_1 = " ", sep_2 = " ", env_prior = parent.frame())

composerr(text_1 = NULL, err_prior = NULL, text_2 = NULL, 
sep_1 = " ", sep_2 = " ", env_prior = parent.frame())

composerr_parent(text_1 = NULL, err_prior = NULL, text_2 = NULL, 
sep_1 = " ", sep_2 = " ", env_prior = parent.frame())
```

**Arguments**

- `text_1`: A character string, which will be appended at the beginning of the error message. The argument `sep_1` will be used as text separator.
- `err_prior`: There are three valid types:
  - `err_prior` is omitted: A new error handling message will be returned.
  - `composerr_` is the calling function: `err_prio` must be a character string holding the name of the error handling function to which the message part should be appended.
  - `composerr` is the calling function: `err_prio` must be the error handling function to which the message part should be appended.
- `text_2`: A character string, which will be appended at the end of the error message. The argument `sep_2` will be used as text separator.
- `sep_1`: A character string that is used as separator for the concatenation of `text_1` at the beginning of the error message.
dictionary_to_yaml

seps

A character string that is used as separator for the concatenation of text_2 at the end of the error message.

env_prior

An environment where the error handling function given in err_prior can be found. If no environment is given, then the err_prior will be looked up in the current environment. In the situation of nested scopes, you may change the lookup environment to the parent environment in order to be able to recursively override the name of the error handling function. In order to keep it simple, the function composerr_parent() can be used instead.

Value

A new error handling function that has an extended error message.

contains_na_escape

Check if a character vector contains NA replacement strings

Description

Check if a character vector contains NA replacement strings

Usage

contains_na_escape(x)

Arguments

x A character vector that should be checked.

Value

TRUE if the vector contains NA replacement strings. FALSE else.

dictionary_to_yaml

Transform data structure from lama_dictionary class input format to the yaml format

Description

In the lama_dictionary class object the data has the structure vars (named list) > translations (named character vector) This structure is transformed to the yaml file structure vars (named list) > translations (named list)

Usage

dictionary_to_yaml(data)
**escape_to_na**

**Arguments**

*data*  
A list that has lama-dictionary structure.

**Value**

An object similar to lama-dictionary object, but each translation is not a named character vector, but a named list holding character strings.

---

**Description**

Replace "NA_" by NA

**Usage**

`escape_to_na(x)`

**Arguments**

*x*  
A character vector that should be modified.

**Value**

A character vector, where the NA replacement strings are replaced by NAs.

---

**is.lama_dictionary**  
*Check if an object is a lama_dictionary class object*

**Description**

Check if an object is a lama_dictionary class object

**Usage**

`is.lama_dictionary(obj)`

**Arguments**

*obj*  
The object in question

**Value**

TRUE if the object is a lama_dictionary class object, FALSE otherwise.
is.syntactic

See Also
validate_lama_dictionary(), as.lama_dictionary(), new_lama_dictionary(), lama_translate(), lama_to_factor(), lama_translate_all(), lama_to_factor_all(), lama_read(), lama_write(), lama_translate(), lama_read(), lama_write(), lama_select(), lama_rename(), lama_mutate(), lama_merge()

Examples

# check if an object is a 'lama_dictionary' class object
dict <- new_lama_dictionary(country = c(uk = "United Kingdom", fr = "France"))
is.lama_dictionary(dict)

References
http://r.789695.n4.nabble.com/Syntactically-valid-names-td3636819.html
lama_get

Retrieve a translation from a lama_dictionary class object

Description

The functions lama_get() and lama_get_() take a lama_dictionary and extract a specific translation. The function lama_get() uses non-standard evaluation, whereas lama_get_() is the standard evaluation alternative.

Usage

lama_get(.data, translation)

## S3 method for class 'lama_dictionary'
lama_get(.data, translation)

lama_get_(.data, translation)

## S3 method for class 'lama_dictionary'
lama_get_(.data, translation)

Arguments

.data A lama_dictionary object

translation Depending on which function was used:

• lama_get: An unquoted translation name.
• lama_get_: A character string holding the translation name.

Value

The wanted translation (named character vector).

Translations

A translation is a named character vector of non zero length. This named character vector defines which labels (of type character) should be assigned to which values (can be of type character, logical or numeric) (e.g. the translation c("0" = "urban","1" = "rural") assigns the label "urban" to the value 0 and "rural" to the value 1, for example the variable x = c(0,0,1) is translated to x_new = c("urban","urban","rural"). Therefore, a translation (named character vector) contains the following information:

• The names of the character vector entries correspond to the original variable levels. Variables of types numeric or logical are turned automatically into a character vector (e.g. 0 and 1 are treated like "0" and "1").
• The entries (character strings) of the character vector correspond to the new labels, which will be assigned to the original variable levels. It is also allowed to have missing labels (NAs). In this case, the original values are mapped onto missing values.
The function `lama_translate()` is used in order to apply a translation on a variable. The resulting vector with the assigned labels can be of the following types:

- **character**: An unordered vector holding the new character labels.
- **factor** with character levels: An ordered vector holding the new character labels.

The original variable can be of the following types:

- **character** vector: This is the simplest case. The character values will replaced by the corresponding labels.
- **numeric or logical** vector: Vectors of type numeric or logical will be turned into character vectors automatically before the translation process and then simply processed like in the character case. Therefore, it is sufficient to define the translation mapping for the character case, since it also covers the numeric and logical case.
- **factor** vector with levels of any type: When translating factor variables one can decide whether or not to keep the original ordering. Like in the other cases the levels of the factor variable will always be turned into character strings before the translation process.

### Missing values

It is also possible to handle missing values with `lama_translate()`. Therefore, the used translation must contain a information that tells how to handle a missing value. In order to define such a translation the missing value (NA) can be escaped with the character string "NA_". This can be useful in two situations:

- All missing values should be labeled (e.g. the translation `c("0" = "urban","1" = "rural",NA_ = "missing") assigns the character string "missing" to all missing values of a variable).
- Map some original values to NA (e.g. the translation `c("0" = "urban","1" = "rural","2" = "NA_","3" = "NA_") assigns NA (the missing character) to the original values 2 and 3). Actually, in this case the translation definition does not always have to use this escape mechanism, but only when defining the translations inside of a YAML file, since the YAML parser does not recognize missing values.

### lama_dictionary class objects

Each `lama_dictionary` class object can contain multiple translations, each with a unique name under which the translation can be found. The function `lama_translate()` uses a `lama_dictionary` class object to translate a normal vector or to translate one or more columns in a `data.frame`. Sometimes it may be necessary to have different translations for the same variable, in this case it is best to have multiple translations with different names (e.g. `area_short = c("0" = "urb","1" = "rur")` and `area = c("0" = "urban","1" = "rural")`).
**Description**

This function takes multiple lama_dictionary class objects and merges them together into a single lama_dictionary class object. In case some class objects have entries with the same name, the class objects passed in later overwrite the class objects passed in first (e.g. in lama_merge(x,y,z): The lexicon z overwrites x and y. The lexicon y overwrites x).

**Usage**

```r
lama_merge(..., show_warnings = TRUE)
```

```r
# S3 method for class 'lama_dictionary'
lama_merge(..., show_warnings = TRUE)
```

**Arguments**

- `...` Two or more lama_dictionary class objects, which should be merged together.
- `show_warnings` A logical flag that defines, whether warnings should be shown (TRUE) or not (FALSE).

**Value**

The merged lama_dictionary class object

**See Also**

lama_translate(), lama_to_factor(), lama_translate_all(), lama_to_factor_all(), new_lama_dictionary(), as.lama_dictionary(), lama_rename(), lama_select(), lama_mutate(), lama_read(), lama_write()

**Examples**

```r
# initialize lama_dictinoary
dict_1 <- new_lama_dictionary(
  subject = c(en = "English", ma = "Mathematics"),
  result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)
dict_2 <- new_lama_dictionary(
  result = c("1" = "Super", "2" = "Fantastic", "3" = "Brilliant"),
  grade = c(a = "Primary School", b = "Secondary School")
)
dict_3 <- new_lama_dictionary(
  country = c(en = "England", "at" = "Austria", NA_ = "Some other country")
)
dict <- lama_merge(dict_1, dict_2, dict_3)
# The lama_dictionary now contains the translations
lama_mutate

Change or append a variable translation to an existing lama_dictionary object

Description

The functions lama_mutate() and lama_mutate_() alter a lama_dictionary object. They can be used to alter, delete or append a translations to a lama_dictionary object. The function lama_mutate() uses named arguments to assign the translations to the new names (similar to dplyr::mutate), whereas the function lama_mutate_() is takes a character string key holding the name to which the translation should be assigned and a named character vector translation holding the actual translation mapping.

Usage

lama_mutate(.data, ...)

## S3 method for class 'lama_dictionary'
lama_mutate(.data, ...)

lama_mutate_(.data, key, translation)

## S3 method for class 'lama_dictionary'
lama_mutate_(.data, key, translation)

Arguments

.data

A lama_dictionary object

... One or more unquoted expressions separated by commas. Use named arguments, e.g. new_translation_name = c(a = "A", b = "B"), to set translations (named character vectors) to new translation names. If you want to delete an existing translation assign the value NULL (e.g. old_translation = NULL). It is also possible use complex expressions as long as the resulting object is a valid translation object (named character vector). Furthermore, it is possible to use translation names that are already existing in the dictionary, in order to modify them (e.g. new_translation = c(v = "V", w = "W", old_translation, z = "Z"), where old_translation = c(x = "X", y = "Y").

key The name of the variable translation that should be altered. It can also be variable translation name that does not exist yet.

translation A named character vector holding the new variable translation that should be assigned to the name given in argument key. The names of the character vector translation correspond to the original variable values that should be replaced by the new labels. The values in the character vector translations are the labels that should be assigned to the original values.
Value

An updated lama_dictionary class object.

See Also

lama_translate(), lama_to_factor(), lama_translate_all(), lama_to_factor_all(), new_lama_dictionary(),
as.lama_dictionary(), lama_rename(), lama_select(), lama_merge(), lama_read(), lama_write()

Examples

# initialize lama_dictionary
dict <- new_lama_dictionary(
  subject = c(en = "English", ma = "Mathematics"),
  result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)

## Example-1: mutate and append with 'lama_mutate'
# add a few subjects and a few grades
dict_new <- lama_mutate(
  dict,
  subject = c(bio = "Biology", subject, sp = "Sports"),
  result = c("0" = "Beyond expectations", result, "4" = "Failed", NA_ = "Missed")
)
# the subjects "Biology" and "Sports" were added
# and the results "Beyond expectations", "Failed" and "Missed"
dict_new

## Example-2: delete with 'lama_mutate'
dict_new <- lama_mutate(
  dict,
  subject = NULL
)
dict_new

## Example-3: Alter and append with 'lama_mutate_'
# generate the new translation (character string)
subj <- c(
  bio = "Biology",
  lama_get(dict, subject),
  sp = "Sports"
)
# save the translation under the name "subject"
dict_new <- lama_mutate_(
  dict,
  key = "subject",
  translation = subj
)
# the translation "subject" now also contains
# the subjects "Biology" and "Sports"
dict_new

## Example-4: Delete with 'lama_mutate_'
# save the translation under the name "subject"

```r
dict_new <- lama_mutate_(
  dict,
  key = "subject",
  translation = NULL
)
```

# the translation "subject" was deleted

dict_new

---

### lama_read

**Read in a yaml file holding translations for one or multiple variables**

**Description**

Read in a yaml file holding translations for one or multiple variables

**Usage**

```r
lama_read(yaml_path)
```

**Arguments**

- `yaml_path` 
  Path to yaml file holding the labels and translations for multiple variables

**Value**

A *lama_dictionary* class object holding the variable translations defined in the yaml file

**Examples**

```r
path_to_file <- system.file("extdata", "dictionary_exams.yaml", package = "labelmachine")
dict <- lama_read(path_to_file)
```

---

### lama_rename

**Rename multiple variable translations in a lama_dictionary object**

**Description**

The functions *lama_rename()* and *lama_rename_*() are used to rename one or more variable translations inside of a *lama_dictionary* class object. The function *lama_rename()* uses non-standard evaluation, whereas *lama_rename_*() is the standard evaluation alternative.
Usage

```r
lama_rename(.data, ...)
```

```r
# S3 method for class 'lama_dictionary'
lama_rename(.data, ...)
```

```r
dlama_rename_(.data, old, new)
```

```r
# S3 method for class 'lama_dictionary'
dlama_rename_(.data, old, new)
```

Arguments

- `.data` A `lama_dictionary` object, holding the variable translations
- `...` One or more unquoted expressions separated by commas. Use named arguments, e.g. `new_name = old_name`, to rename selected variables.
- `old` A character vector holding the names of the variable translations, that should be renamed.
- `new` A character vector holding the new names of the variable translations.

Value

The updated `lama_dictionary` class object.

See Also

`lama_translate()`, `lama_to_factor()`, `lama_translate_all()`, `lama_to_factor_all()`, `new_lama_dictionary()`, `as.lama_dictionary()`, `lama_select()`, `lama_mutate()`, `lama_merge()`, `lama_read()`, `lama_write()`

Examples

```r
# initialize lama_dictionary
dict <- new_lama_dictionary(
country = c(uk = "United Kingdom", fr = "France", NA = "other countries"),
language = c(en = "English", fr = "French"),
result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)
```

```r
# Example-1: Usage of 'lama_rename'
# rename translations 'result' and 'language' to 'res' and 'lang'
dict.new <- lama_rename(dict, res = result, lang = language)
dict.new
```

```r
# Example-2: Usage of 'lama_rename_'
# rename translations 'result' and 'language' to 'res' and 'lang'
dict.new <- lama_rename_(dict, c("result", "language"), c("res", "lang"))
dict.new
```
The functions `lama_select()` and `lama_select_()` pick one or more variable translations from a `lama_dictionary` class object and create a new `lama_dictionary` class object. The function `lama_select()` uses non-standard evaluation, whereas `lama_select_()` is the standard evaluation alternative.

### Usage

```r
lama_select(.data, ...)  
```

```
## S3 method for class 'lama_dictionary'
lama_select(.data, ...)  

lama_select_(.data, key)  
```

### Arguments

- `.data` A `lama_dictionary` object, holding the variable translations
- `...` One or more unquoted translation names separated by commas.
- `key` A character vector holding the names of the variable translations that should be picked.

### Value

A new `lama_dictionary` class object, holding the picked variable translations.

### See Also

`lama_translate()`, `lama_to_factor()`, `lama_translate_all()`, `lama_to_factor_all()`, `new_lama_dictionary()`, `as.lama_dictionary()`, `lama_rename()`, `lama_mutate()`, `lama_merge()`, `lama_read()`, `lama_write()`

### Examples

```r
# initialize lama_dictinoary
dict <- new_lama_dictionary(
  country = c(uk = "United Kingdom", fr = "France", NA_ = "other countries"),
  language = c(en = "English", fr = "French"),
  result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)

## Example-1: Usage of 'lama_select'
```
# pick the translations 'result' and 'language'
# and add them to a new lama_dictionary
dict_sub <- lama_select(dict, result, language)
dict_sub

## Example-2: Usage of 'lama_select_'
# pick the translations 'result' and 'language'
# and add them to a new lama_dictionary
dict_sub <- lama_select_(dict, c("result", "language"))
dict_sub

### lama_translate

#### Assign new labels to a variable of a data.frame

---

**Description**

The functions `lama_translate()` and `lama_translate_()` take a factor, a vector or a data.frame and convert one or more of its categorical variables (not necessarily a factor variable) into factor variables with new labels. The function `lama_translate()` uses non-standard evaluation, whereas `lama_to_factor()` is the standard evaluation alternative. The functions `lama_to_factor()` and `lama_to_factor_()` are very similar to the functions `lama_translate()` and `lama_translate_()`, but instead of assigning new label strings to values, it is assumed that the variables are character vectors or factors, but need to be turned into factors with the order given in the translations:

- `lama_translate()` and `lama_translate_()`: Assign new labels to a variable and turn it into a factor variable with the order given in the corresponding translation (keep_order = FALSE) or in the same order as the original variable (keep_order = TRUE).
- `lama_to_factor()` and `lama_to_factor_()`: The variable is a character vector or a factor already holding the right label strings. The variables are turned into a factor variable with the order given in the corresponding translation (keep_order = FALSE) or in the same order as the original variable (keep_order = TRUE).

**Usage**

```r
lama_translate(.data, dictionary, ..., keep_order = FALSE, to_factor = TRUE)
```

```
# S3 method for class 'data.frame'
lama_translate(.data, dictionary, ..., keep_order = FALSE, to_factor = TRUE)
```

```
# Default S3 method:
lama_translate(.data, dictionary, ..., keep_order = FALSE, to_factor = TRUE)
```

```r
lama_translate_(.data, dictionary, translation, col = translation, col_new = col, keep_order = FALSE, to_factor = TRUE, ...)
```
## S3 method for class 'data.frame'
lama_translate_(.data, dictionary, translation,
    col = translation, col_new = col, keep_order = FALSE,
    to_factor = TRUE, ...)

## Default S3 method:
lama_translate_(.data, dictionary, translation, ..., 
    keep_order = FALSE, to_factor = TRUE)

lama_to_factor(.data, dictionary, ..., keep_order = FALSE)

## S3 method for class 'data.frame'
lama_to_factor(.data, dictionary, ..., 
    keep_order = FALSE)

## Default S3 method:
lama_to_factor(.data, dictionary, ..., 
    keep_order = FALSE)

lama_to_factor_(.data, dictionary, translation, col = translation,
    col_new = col, keep_order = FALSE, ...)

## S3 method for class 'data.frame'
lama_to_factor_(.data, dictionary, translation, col = translation,
    col_new = col, keep_order = FALSE, ...)

## Default S3 method:
lama_to_factor_(.data, dictionary, translation, ..., 
    keep_order = FALSE)

### Arguments

.data
Either a data frame, a factor or an atomic vector.

dictionary
A lama_dictionary object, holding the translations for various variables.

...
Only used by lama_translate() and lama_to_factor(). Each argument in ...

is an unquoted expression and defines a translation. Use unquoted

arguments that tell which translation should be applied to which column and which

column name the relabeled variable should be assigned to. E.g. lama_translate(.data,dict,Y1

= TRANS1(X1),Y2 = TRANS2(Y2)) and lama_to_factor(.data,dict,Y1 = TRANS1(X1),Y2

= TRANS2(Y2)) and to apply the translations TRANS1 and TRANS2 to the data.frame

columns X1 and X2 and save the new labeled variables under the column names

Y1 and Y2. There are also two abbreviation mechanisms available: The argument

assignment FOO(X) is the same as X = FOO(X) and FOO is an abbreviation for FOO

= FOO(FOO). In case, .data is not a data frame but a plain factor or an atomic

vector, then the argument ... must be a single unquoted translation name (e.g.

lama_translate(x,dict,TRANS1), where x is a factor or an atomic vector and

TRANS1 is the name of the translation, which should be used to assign the labels
to the values of x.)
**keep_order**
A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If a translated variable in the data.frame is a factor variable, and the corresponding boolean configuration is set to TRUE, then the the order of the original factor variable will be preserved.

**to_factor**
A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.

**translation**
A character vector holding the names of the variable translations which should be used for assigning new labels to the variable. This names must be a subset of the translation names returned by names(dictionary).

**col**
Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names of the data.frame columns that should be relabeled. If omitted, then it will be assumed that the column names are the same as the given translation names in the argument translation.

**col_new**
Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names under which the relabeled variables should be stored in the data.frame. If omitted, then it will be assumed that the new column names are the same as the column names of the original variables.

### Details
The functions lama_translate(), lama_translate_(), lama_to_factor() and lama_to_factor_() require different arguments, depending on the data type passed into argument .data. If .data is of type character, logical, numeric or factor, then the arguments col and col_new are omitted, since those are only necessary in the case of data frames.

### Value
An extended data.frame, that has a factor variable holding the assigned labels.

### See Also
lama_translate_all(), lama_to_factor_all(), new_lama_dictionary(), as.lama_dictionary(), lama_rename(), lama_select(), lama_mutate(), lama_merge(), lama_read(), lama_write()

### Examples
```r
# initialize lama_dictinary
dict <- new_lama_dictionary(
```
subject = c(en = "English", ma = "Mathematics"),
result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)
# the data frame which should be translated
df <- data.frame(
  pupil = c(1, 1, 2, 2, 3),
  subject = c("en", "ma", "ma", "en", "en"),
  res = c(1, 2, 3, 2, 2)
)

## Example-1: Usage of 'lama_translate' for data frames
## Full length assignment
# (apply translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply translation 'result' to column 'res' and save it to column 'res_new')
df_new <- lama_translate(
  df,
  dict,
  sub_new = subject(subject),
  res_new = result(res)
)
str(df_new)

## Example-2: Usage of 'lama_translate' for data frames
## Abbreviation overwriting original columns
# (apply translation 'subject' to column 'subject' and save it to column 'subject')
# (apply translation 'result' to column 'res' and save it to column 'res')
df_new_overwritten <- lama_translate(
  df,
  dict,
  subject(subject),
  result(res)
)
str(df_new_overwritten)

## Example-3: Usage of 'lama_translate' for data frames
## Abbreviation if 'translation_name == column_name'
# (apply translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply translation 'result' to column 'res' and save it to column 'res_new')
df_new_overwritten <- lama_translate(
  df,
  dict,
  subject_new = subject,
  res_new = result(res)
)
str(df_new_overwritten)

## Example-4: Usage of 'lama_translate' for data frames labeling as character vectors
# (apply translation 'subject' to column 'subject' and
# save it as a character vector to column 'subject_new')
df_new_overwritten <- lama_translate(
  df,
  dict,
  subject_new = subject,
to_factor = TRUE
)
str(df_new_overwritten)

## Example-5: Usage of 'lama_translate' for atomic vectors
sub <- c("ma", "en", "ma")
sub_new <- df_new_overwritten <- lama_translate(
    sub,
    dict,
    subject
)
str(sub_new)

## Example-6: Usage of 'lama_translate' for factors
sub <- factor(c("ma", "en", "ma"), levels = c("ma", "en"))
sub_new <- df_new_overwritten <- lama_translate(
    sub,
    dict,
    subject,
    keep_order = TRUE
)
str(sub_new)

## Example-7: Usage of 'lama_translate_' for data frames
# (apply translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply translation 'result' to column 'res' and save it to column 'res_new')
df_new <- lama_translate_(
    df,
    dict,
    translation = c("subject", "result"),
    col = c("subject", "res"),
    col_new = c("subject_new", "res_new")
)
str(df_new)

## Example-8: Usage of 'lama_translate_' for data frames and store as character vector
# (apply translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply translation 'result' to column 'res' and save it to column 'res_new')
df_new <- lama_translate_(
    df,
    dict,
    translation = c("subject", "result"),
    col = c("subject", "res"),
    col_new = c("subject_new", "res_new"),
    to_factor = c(FALSE, FALSE)
)
str(df_new)

## Example-9: Usage of 'lama_translate_' for atomic vectors
res <- c(1, 2, 1, 3, 1, 2)
res_new <- df_new_overwritten <- lama_translate_(
    res,
    dict,
## Example-10: Usage of `lama_translate_` for factors

```r
sub <- factor(c("ma", "en", "ma"), levels = c("ma", "en"))
sub_new <- lama_translate_(
  sub,
  dict,
  "subject",
  keep_order = TRUE
)
str(sub_new)
```

# the data frame which holds the right labels, but no factors
```r
df_translated <- data.frame(
  pupil = c(1, 1, 2, 2, 3),
  subject = c("English", "Mathematics", "Mathematics", "English", "English"),
  res = c("Very good", "Good", "Not so good", "Good", "Good")
)
```

## Example-11: Usage of `lama_to_factor` for data frames

```r
# Full length assignment
# (apply order of translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply order of translation 'result' to column 'res' and save it to column 'res_new')
df_new <- lama_to_factor(
  df_translated,
  dict,
  sub_new = subject(subject),
  res_new = result(res)
)
str(df_new)
```

## Example-12: Usage of `lama_to_factor` for data frames

```r
# Abbreviation overwriting original columns
# (apply order of translation 'subject' to column 'subject' and save it to column 'subject')
# (apply order of translation 'result' to column 'res' and save it to column 'res')
df_new_overwritten <- lama_to_factor(
  df_translated,
  dict,
  subject(subject),
  result(res)
)
str(df_new_overwritten)
```

## Example-13: Usage of `lama_to_factor` for data frames

```r
# Abbreviation if `translation_name == column_name`
# (apply order of translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply order of translation 'result' to column 'res' and save it to column 'res_new')
df_new_overwritten <- lama_to_factor(
  df_translated,
  dict,
  subject_new = subject,
  res_new = result(res)
)
### Example-14: Usage of 'lama_translate' for atomic vectors

```r
var <- c("Mathematics", "English", "Mathematics")
var_new <- lama_to_factor(
  var,
  dict = subject
)
str(var_new)
```

### Example-15: Usage of 'lama_to_factor_' for data frames

```r
# (apply order of translation 'subject' to column 'subject' and save it to column 'subject_new')
# (apply order of translation 'result' to column 'res' and save it to column 'res_new')
df_new <- lama_to_factor_(
  df_translated,
  dict = c("subject", "result"),
  col = c("subject", "res"),
  col_new = c("subject_new", "res_new")
)
str(df_new)
```

### Example-16: Usage of 'lama_to_factor_' for atomic vectors

```r
var <- c("Very good", "Good", "Good")
var_new <- lama_to_factor_(
  var,
  dict = c("result")
)
str(var_new)
```

---

**lama_translate_all**  
Assign new labels to all variables of a data.frame

---

**Description**

The functions `lama_translate_all()` and `lama_to_factor_all()` converts all variables (which have a translation in the given lama-dictionary) of a data frame `.data` into factor variables with new labels. These functions are special versions of the functions `lama_translate()` and `lama_to_factor()`. The difference to `lama_translate()` and `lama_to_factor()` is, that when using `lama_translate_all()` and `lama_to_factor_all()` the used translations in dictionary must have the exact same names as the corresponding columns in the data frame `.data`.

**Usage**

```r
lama_translate_all(.data, dictionary, prefix = "", suffix = "",
  fn_colname = function(x) x, keep_order = FALSE, to_factor = TRUE)
```
## S3 method for class 'data.frame'
lama_translate_all(.data, dictionary, prefix = "", suffix = "", fn_colname = function(x) x, keep_order = FALSE, to_factor = TRUE)

lama_to_factor_all(.data, dictionary, prefix = "", suffix = "", fn_colname = function(x) x, keep_order = FALSE)

## S3 method for class 'data.frame'
lama_to_factor_all(.data, dictionary, prefix = "", suffix = "", fn_colname = function(x) x, keep_order = FALSE)

### Arguments
- `.data` Either a data frame, a factor or a vector.
- `dictionary` A `lama_dictionary` object, holding the translations for various variables.
- `prefix` A character string, which is used as prefix for the new column names.
- `suffix` A character string, which is used as suffix for the new column names.
- `fn_colname` A function, which transforms character string into a new character string. This function will be used to transform the old column names into new column names under which the labeled variables will then be stored.
- `keep_order` A logical of length one, defining if the original order (factor order or alphanumerical order) of the data frame variables should be preserved.
- `to_factor` A logical of length one, defining if the resulting labeled variables should be factor variables (`to_factor = TRUE`) or plain character vectors (`to_factor = FALSE`).

### Details
The difference between `lama_translate_all()` and `lama_to_factor_all()` is the following:

- **`lama_translate_all()`**: Assign new labels to the variables and turn them into factor variables with the order given in the corresponding translations (`keep_order = FALSE`) or in the same order as the original variable (`keep_order = TRUE`).
- **`lama_to_factor_all()`**: The variables are character vectors or factors already holding the right label strings. The variables are turned into a factor variables with the order given in the corresponding translation (`keep_order = FALSE`) or in the same order as the original variable (`keep_order = TRUE`).

### Value
An extended data.frame, that has a factor variable holding the assigned labels.

### See Also
- `lama_translate()`, `lama_to_factor()`, `new_lama_dictionary()`, `as.lama_dictionary()`, `lama_rename()`, `lama_select()`, `lama_mutate()`, `lama_merge()`, `lama_read()`, `lama_write()`
Examples

```r
## initialize lama_dictionary
dict <- new_lama_dictionary(
  subject = c(en = "English", ma = "Mathematics"),
  result = c("1" = "Very good", "2" = "Good", "3" = "Not so good")
)

## data frame which should be translated
df <- data.frame(
  pupil = c(1, 1, 2, 2, 3),
  subject = c("en", "ma", "ma", "en", "en"),
  result = c(1, 2, 3, 2, 2)
)

## Example-1: 'lama_translate_all'
df_new <- lama_translate_all(
  df,
  dict,
  prefix = "pre_",
  fn_colname = toupper,
  suffix = "_suf"
)
str(df_new)

## Example-2: 'lama_translate_all' with 'to_factor = FALSE'
# The resulting variables are plain character vectors
df_new <- lama_translate_all(df, dict, suffix = "_new", to_factor = TRUE)
str(df_new)

## Example-3: 'lama_to_factor_all'
# The variables 'subject' and 'result' are turned into factor variables
# The ordering is taken from the translations 'subject' and 'result'
df_2 <- data.frame(
  pupil = c(1, 1, 2, 2, 3),
  subject = c("English", "Mathematics", "Mathematics", "English", "English"),
  result = c("Very good", "Good", "Good", "Very good", "Good")
)
df_2_new <- lama_to_factor_all(
  df_2, dict,
  prefix = "pre_",
  fn_colname = toupper,
  suffix = "_suf"
)
str(df_new)
```

lama_write

Write a yaml file holding translations for one or multiple variables

Description

Write a yaml file holding translations for one or multiple variables
Usage

```r
lama_write(x, yaml_path)
```

Arguments

- `x`: A `lama_dictionary` class object holding the variable translations
- `yaml_path`: File path, where the yaml file should be saved

Examples

```r
dict <- new_lama_dictionary(results = c(p = "Passed", f = "Failed"))
path_to_file <- file.path(tempdir(), "my_dictionary.yaml")
lama_write(dict, path_to_file)
```

Description

Improve `base::lapply()` and `base::sapply()` functions by allowing an extra index argument `.I` to be passed into the function given in FUN. If the function given in FUN has an argument `.I` then, for each entry of X passed into FUN the corresponding index is passed into argument `.I`. If the function given in FUN has no argument `.I`, then `lapplI` and `sapplI` are exactly the same as `base::lapply()` and `base::sapply()`. Besides this extra feature, there is no difference to `base::lapply()` and `base::sapply()`.

Usage

```r
lapplI(X, FUN, ...)
sapplI(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Arguments

- `X`: a vector (atomic or list) or an `expression` object. Other objects (including classed objects) will be coerced by `base::as.list`.
- `FUN`: Here comes the great difference to `base::lapply()` and `base::sapply()`. When using `lapplI` and `sapplI`, the function passed into FUN may also have an extra argument `.I`. If it does, then for each item of `X` the current item index is passed into argument `.I` of FUN. Besides this extra feature, there is no difference to `base::lapply()` and `base::sapply()`.
- `...`: optional arguments to FUN.
named_lapply

- **simplify**: logical or character string; should the result be simplified to a vector, matrix or higher dimensional array if possible? For sapply it must be named and not abbreviated. The default value, TRUE, returns a vector or matrix if appropriate, whereas if simplify = "array" the result may be an array of “rank” (=length(dim(.))) one higher than the result of FUN(X[[i]])

- **USE.NAMES**: logical; if TRUE and if X is character, use X as names for the result unless it had names already. Since this argument follows ... its name cannot be abbreviated.

---

**Description**

Create a named list with lapply from a character vector

**Usage**

named_lapply(.names, FUN, ...)

**Arguments**

- `.names`: A character vector holding the names of the list
- `FUN`: Here comes the great difference to base::lapply() and base::sapply(). When using lapply and sapply, the function passed into FUN may also have an extra argument .i. If it does, then for each item of X the current item index is passed into argument .i of FUN. Besides this extra feature, there is no difference to base::lapply() and base::sapply().

- `...`: optional arguments to FUN.

**Value**

A named list

---

**NA.lama_**

**NA replace string**

---

**Description**

In order to replace NA values in yaml files and in translations the following character string is used

**Usage**

NA.lama_

**Format**

An object of class character of length 1.
new_lama_dictionary

---

na_to_escape

*Replace NA by "NA_"

---

**Description**

Replace NA by "NA_"

**Usage**

```r
da_to_escape(x)
```

**Arguments**

- `x`: A character vector that should be modified.

---

**Value**

A character vector, where the NAs are replaced.

---

new_lama_dictionary

*Create a new lama_dictionary class object*

---

**Description**

Generates an S3 class object, which holds the variable translations. There are three valid ways to use `new_lama_dictionary` in order to create a `lama_dictionary` class object:

- **No arguments were passed into ...**: In this case `new_lama_dictionary` returns an empty `lama_dictionary` class object (e.g. `dict <- new_lama_dictionary()`).

- **The first argument is a list**: In this case only the first argument of `new_lama_dictionary` is used. It is not necessary to pass in a named argument. The passed in object must be a named list object, which contains all translations that should be added to the new `lama_dictionary` class object. Each item of the named list object must be a named character vector defining a translation (e.g. `new_lama_dictionary(list(area = c("0" = "urban","1" = "rural"),= c(l = "Low",h = "High")))` generates a `lama_dictionary` class object holding the translations "area" and "density").

- **The first argument is a character vector**: In this case, it is allowed to pass in more than one argument. In this case, all given arguments must be named arguments holding named character vectors defining translations (e.g. `new_lama_dictionary(area = c("0" = "urban","1" = "rural"),density = c(l = "Low",h = "High"))` generates a `lama_dictionary` class object holding the translations "area" and "density"). The names of the passed in arguments will be used as the names, under which the given translations will be added to the new `lama_dictionary` class object.
Usage

new_lama_dictionary(...)

## S3 method for class 'list'
new_lama_dictionary(.data = NULL, ...)

## S3 method for class 'character'
new_lama_dictionary(...)

## Default S3 method:
new_lama_dictionary(...)

Arguments

... None, one or more named/unnamed arguments. Depending on the type of the
  type of the first argument passed into new_lama_dictionary, there are different
  valid ways of using new_lama_dictionary:

  • No arguments were passed into ...: In this case new_lama_dictionary re-
    turns an empty lama_dictionary class object (e.g. dict <- new_lama_dictionary()).

  • The first argument is a list: In this case, only the first argument of new_lama_dictionary
    is used and it is allowed to use an unnamed argument call. Furthermore, the
    passed in object must be a named list object, which contains all translations
    that should be added to the new lama_dictionary class object. Each item
    of the named list object must be a named character vector defining a trans-
    lation (e.g. new_lama_dictionary(list(area = c("0" = "urban","1" =
"
rural"), = c(l = "Low", h = "High"))) generates a lama_dictionary class
    object holding the translations "area" and "density").

  • The first argument is a character vector: In this case, it is allowed to
    pass in more than one argument, but all given arguments when calling
    new_lama_dictionary must be named arguments and each argument must be a
    named character vectors defining translations (e.g. new_lama_dictionary(area
    = c("0" = "urban","1" = "rural"), density = c(l = "Low", h = "High"))
    generates a lama_dictionary class object holding the translations "area" and
    "density"). The names of the caller arguments will be used as names
    under which the given translations will be added to the new lama_dictionary
    class object.

.data A named list object, where each list entry corresponds to a translation that should
  be added to the lama_dictionary object (e.g. new_lama_dictionary(list(area
  = c("0" = "urban","1" = "rural"), = c(l = "Low", h = "High"))) generates a
  lama_dictionary class object holding the translations "area" and "density").
  The names of the list entries are the names under which the translation will be
  added to the new lama_dictionary class object (e.g. area and density). Each
  list entry must be a named character vector defining a translation (e.g. c("0"
  = "urban","1" = "rural") is the translation with the name area and c(l =
"Low", h = "High") is the translation with the name density).
Value

A new lama_dictionary class object holding the passed in translations.

Translations

A translation is a named character vector of non zero length. This named character vector defines which labels (of type character) should be assigned to which values (can be of type character, logical or numeric) (e.g. the translation `c("0" = "urban","1" = "rural")` assigns the label "urban" to the value 0 and "rural" to the value 1, for example the variable `x = c(0,0,1)` is translated to `x_new = c("urban","urban","rural").` Therefore, a translation (named character vector) contains the following information:

- The names of the character vector entries correspond to the original variable levels. Variables of types numeric or logical are turned automatically into a character vector (e.g. 0 and 1 are treated like "0" and "1").
- The entries (character strings) of the character vector correspond to the new labels, which will be assigned to the original variable levels. It is also allowed to have missing labels (NAs). In this case, the original values are mapped onto missing values.

The function `lama_translate()` is used in order to apply a translation on a variable. The resulting vector with the assigned labels can be of the following types:

- character: An unordered vector holding the new character labels.
- factor with character levels: An ordered vector holding the new character labels.

The original variable can be of the following types:

- character vector: This is the simplest case. The character values will replaced by the corresponding labels.
- numeric or logical vector: Vectors of type numeric or logical will be turned into character vectors automatically before the translation process and then simply processed like in the character case. Therefore, it is sufficient to define the translation mapping for the character case, since it also covers the numeric and logical case.
- factor vector with levels of any type: When translating factor variables one can decide whether or not to keep the original ordering. Like in the other cases the levels of the factor variable will always be turned into character strings before the translation process.

Missing values

It is also possible to handle missing values with `lama_translate()`. Therefore, the used translation must contain a information that tells how to handle a missing value. In order to define such a translation the missing value (NA) can be escaped with the character string "NA_". This can be useful in two situations:

- All missing values should be labeled (e.g. the translation `c("0" = "urban","1" = "rural",NA_ = "missing")` assigns the character string "missing" to all missing values of a variable).
- Map some original values to NA (e.g. the translation `c("0" = "urban","1" = "rural","2" = "NA_","3" = "NA_")` assigns NA (the missing character) to the original values 2 and 3). Actually, in this case the translation definition does not always have to use this escape mechanism, but only when defining the translations inside of a YAML file, since the YAML parser does not recognize missing values.
lama_dictionary class objects

Each lama_dictionary class object can contain multiple translations, each with a unique name under which the translation can be found. The function lama_translate() uses a lama_dictionary class object to translate a normal vector or to translate one or more columns in a data.frame. Sometimes it may be necessary to have different translations for the same variable, in this case it is best to have multiple translations with different names (e.g. area_short = c("0" = "urb","1" = "rur") and area = c("0" = "urban","1" = "rural")).

See Also

is.lama_dictionary(), as.lama_dictionary(), lama_translate(), lama_translate_all(), lama_to_factor(), lama_to_factor_all(), lama_read(), lama_write(), lama_select(), lama_rename(), lama_mutate(), lama_merge()

Examples

```r
## Example-1: Initialize a lama-dictionary from a list object holding the translations
dict <- new_lama_dictionary(list(
  country = c(uk = "United Kingdom", fr = "France", NA_ = "other countries"),
  language = c(en = "English", fr = "French")
))
dict

## Example-2: Initialize the lama-dictionary directly by assigning each translation to a name
dict <- new_lama_dictionary(
  country = c(uk = "United Kingdom", fr = "France", NA_ = "other countries"),
  language = c(en = "English", fr = "French")
)
dict
```

print.lama_dictionary

Print a lama_dictionary class object

Description

Print a lama_dictionary class object

Usage

```r
## S3 method for class 'lama_dictionary'
print(x, ...)
```

Arguments

- `x` The lama_dictionary class object that should be printed.
- `...` Unused arguments
See Also

new_lama_dictionary(), as.lama_dictionary(), lama_translate(), lama_to_factor(), lama_translate_all(), lama_to_factor_all(), lama_read(), lama_write(), lama_rename(), lama_select(), lama_mutate(), lama_merge(), lama_read(), lama_write()

rename_translation

Function that actually performs the renaming of the translations

Description

Function that actually performs the renaming of the translations

Usage

rename_translation(.data, old, new)

Arguments

.data A lama_dictionary object, holding the variable translations
.old A character vector holding the names of the variable translations, that should be renamed.
.new A character vector holding the new names of the variable translations.

Value

The updated lama_dictionary class object.

stringify

Coerce a vector into a character string ('x1', 'x2', ...)

Description

Coerce a vector into a character string ('x1', 'x2', ...)

Usage

stringify(x)

Arguments

x A vector that should be coerced.

Value

A character string holding the collapsed vector.
translate_df

This function relabels several variables in a data.frame

Description

This function relabels several variables in a data.frame

Usage

translate_df(.data, dictionary, translation, col, col_new, keep_order, to_factor, is_translated, err_handler)

Arguments

.data Either a data frame, a factor or an atomic vector.
dictionary A lama_dictionary object, holding the translations for various variables.
translation A character vector holding the names of the variable translations which should be used for assigning new labels to the variable. This names must be a subset of the translation names returned by names(dictionary).
col Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names of the data.frame columns that should be relabeled. If omitted, then it will be assumed that the column names are the same as the given translation names in the argument translation.
col_new Only used if .data is a data frame. The argument col must be a character vector of the same length as translation holding the names under which the relabeled variables should be stored in the data.frame. If omitted, then it will be assumed that the new column names are the same as the column names of the original variables.
keep_order A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If a translated variable in the data.frame is a factor variable, and the corresponding boolean configuration is set to TRUE, then the the order of the original factor variable will be preserved.
to_factor A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If the vector has the same length as the number of arguments in . . . , then the to each variable translation there is a corresponding boolean configuration. If to_factor is TRUE, then the resulting labeled variable will be a factor. If to_factor is set to FALSE, then the resulting labeled variable will be a plain character vector.
translate_vector

is_translated  A boolean vector of length one or the same length as the number of translations. If the vector has length one, then the same configuration is applied to all variable translations. If is_translated = TRUE, then the original variable is a character vector holding the right labels (character strings). In this case, the labels are left unchanged, but the variables are turned into factors with order given in the selected translations.

err_handler  An error handling function

Value

An factor vector holding the assigned labels.

translate_vector  *This function relabels a vector*

Description

This function relabels a vector

Usage

translate_vector(val, translation, keep_order, to_factor, is_translated, err_handler)

Arguments

val  The vector that should be relabeled. Allowed are all vector types (also factor).
translation  Named character vector holding the label assignments.
keep_order  A logical flag. If the vector in val is a factor variable and keep_order is set to TRUE, then the order of the original factor variable is preserved.
to_factor  A logical flag. If set to TRUE, the the resulting labeled variable will be a factor and a plain character vector otherwise.
is_translated  A logical flag. If is_translated = TRUE, then val must be a character vector holding the right labels (character strings) and will be turned into a factor with ordering given in the translation (except for the case when keep_order = TRUE).
err_handler  An error handling function

Value

A factor vector holding the assigned labels
validate_lama_dictionary

Check if an object has a valid lama_dictionary structure

Description

This function checks if the object structure is right. It does not check class type.

Usage

```r
validate_lama_dictionary(obj,
    err_handler = composerr("The object has not a valid lama_dictionary structure"))
```

Arguments

- `obj`: An object that should be tested
- `err_handler`: An error handling function

Translations

A translation is a named character vector of non zero length. This named character vector defines which labels (of type character) should be assigned to which values (can be of type character, logical or numeric) (e.g. the translation `c("0" = "urban","1" = "rural")` assigns the label "urban" to the value 0 and "rural" to the value 1, for example the variable `x = c(0,0,1)` is translated to `x_new = c("urban","urban","rural"). Therefore, a translation (named character vector) contains the following information:

- The names of the character vector entries correspond to the original variable levels. Variables of types numeric or logical are turned automatically into a character vector (e.g. 0 and 1 are treated like "0" and "1").
- The entries (character strings) of the character vector correspond to the new labels, which will be assigned to the original variable levels. It is also allowed to have missing labels (NAs). In this case, the original values are mapped onto missing values.

The function `lama_translate()` is used in order to apply a translation on a variable. The resulting vector with the assigned labels can be of the following types:

- `character`: An unordered vector holding the new character labels.
- `factor` with character levels: An ordered vector holding the new character labels.

The original variable can be of the following types:

- `character` vector: This is the simplest case. The character values will replaced by the corresponding labels.
- `numeric` or `logical` vector: Vectors of type `numeric` or `logical` will be turned into `character` vectors automatically before the translation process and then simply processed like in the `character` case. Therefore, it is sufficient to define the translation mapping for the `character` case, since it also covers the `numeric` and `logical` case.
- **factor** vector with levels of any type: When translating factor variables one can decide whether or not to keep the original ordering. Like in the other cases the levels of the factor variable will always be turned into character strings before the translation process.

**Missing values**

It is also possible to handle missing values with `lama_translate()`. Therefore, the used translation must contain a information that tells how to handle a missing value. In order to define such a translation the missing value (NA) can be escaped with the character string "NA_". This can be useful in two situations:

- All missing values should be labeled (e.g. the translation c("0" = "urban","1" = "rural",NA_ = "missing") assigns the character string "missing" to all missing values of a variable).
- Map some original values to NA (e.g. the translation c("0" = "urban","1" = "rural","2" = "NA_","3" = "NA_") assigns NA (the missing character) to the original values 2 and 3). Actually, in this case the translation definition does not always have to use this escape mechanism, but only when defining the translations inside of a YAML file, since the YAML parser does not recognize missing values.

**lama_dictionary class objects**

Each `lama_dictionary` class object can contain multiple translations, each with a unique name under which the translation can be found. The function `lama_translate()` uses a `lama_dictionary` class object to translate a normal vector or to translate one or more columns in a `data.frame`. Sometimes it may be necessary to have different translations for the same variable, in this case it is best to have multiple translations with different names (e.g. `area_short = c("0" = "urb","1" = "rur")` and `area = c("0" = "urban","1" = "rural")`).

See Also

`is.lama_dictionary()`, `as.lama_dictionary()`, `new_lama_dictionary()`, `lama_translate()`, `lama_to_factor()`, `lama_translate_all()`, `lama_to_factor_all()`, `lama_read()`, `lama_write()`, `lama_select()`, `lama_rename()`, `lama_mutate()`, `lama_merge()`

---

**validate_translation**

Check if an object has a valid translation structure

**Description**

This function checks if the object structure is that of a translation (named character vector).

**Usage**

```r
validate_translation(obj, err_handler = composerr("The object has not a valid translation structure"))
```
**yaml_to_dictionary**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>An object that should be tested</td>
</tr>
<tr>
<td>err_handler</td>
<td>An error handling function</td>
</tr>
</tbody>
</table>

**Description**

When a yaml file is read in, the data has the structure vars (named list) > translations (named list) This structure is transformed to the `lama_dictionary` class input structure vars (named list) > translations (named character vector)

**Usage**

`yaml_to_dictionary(data)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An object similar to a lama-dictionary object, but each translation is not a named character vector, but a named list holding character strings.</td>
</tr>
</tbody>
</table>

**Value**

A list that has lama-dictionary structure.
Index

*Topic datasets
  NA_lama_, 35
array, 35
as.lama_dictionary, 3
as.lama_dictionary(), 16, 19, 21, 23, 24, 27, 32, 39, 40, 44
as.list, 34
base::lapply, 34, 35
base::sapply, 34, 35
check_and_translate_all, 6
check_and_translate_df, 7
check_and_translate_df_, 8
check_and_translate_vector, 9
check_and_translate_vector_, 10
check_arguments, 11
check_rename, 12
check_select, 12
composerr (composerr_), 13
composerr_, 13
composerr(), 13
composerr_parent (composerr_), 13
composerr_parent(), 13, 14
contains_na_escape, 14
dictionary_to_yaml, 14
escape_to_na, 15
expression, 34
is.lama_dictionary, 15
is.lama_dictionary(), 39, 44
is.syntactic, 16
lama_dictionary, 3, 6–12, 14, 15, 17, 19–24, 26, 32, 34, 39–41, 43, 45
lama_get, 17
lama_get(), 17
lama_(lama_get), 17
lama_get(), 17
lama_merge, 19
lama_merge(), 16, 21, 23, 24, 27, 32, 39, 40, 44
lama_mutate, 20
lama_mutate(), 16, 19, 20, 23, 24, 27, 32, 39, 40, 44
lama_mutate_(lama_mutate), 20
lama_mutate(), 20
lama_read, 22
lama_read(), 16, 19, 21, 23, 24, 27, 32, 39, 40, 44
lama_rename, 22
lama_rename(), 12, 16, 19, 21, 22, 24, 27, 32, 39, 40, 44
lama_rename_(lama_rename), 22
lama_rename(), 12, 22
lama_select, 24
lama_select(), 12, 16, 19, 21, 23, 24, 27, 32, 39, 40, 44
lama_select_(lama_select), 24
lama_select(), 12, 24
lama_to_factor (lama_translate), 25
lama_to_factor(), 16, 19, 21, 23–27, 31, 32, 39, 40, 44
lama_to_factor_(lama_translate), 25
lama_to_factor(), 25, 27
lama_to_factor_all (lama_translate_all), 31
lama_to_factor_all(), 16, 19, 21, 23, 24, 27, 31, 32, 39, 40, 44
lama_translate, 25
lama_translate(), 5, 11, 16, 18, 19, 21, 23–27, 31, 32, 38–40, 43, 44
lama_translate_(lama_translate), 25
lama_translate(), 11, 25, 27
lama_translate_all, 31
lama_translate_all(), 16, 19, 21, 23, 24,
INDEX

27, 31, 32, 39, 40, 44
lama_write, 33
lama_write(), 16, 19, 21, 23, 24, 27, 32, 39, 40, 44
lapplI, 34

NA_lama_, 35
na_to_escape, 36
named_lapply, 35
names, 35
new_lama_dictionary, 36
new_lama_dictionary(), 16, 19, 21, 23, 24, 27, 32, 40, 44

print.lama_dictionary, 39
rename_translation, 40

sapplI(lapplI), 34
stringify, 40

translate_df, 41
translate_vector, 42

validate_lama_dictionary, 43
validate_lama_dictionary(), 16
validate_translation, 44

yaml_to_dictionary, 45