Package 'labelmachine'

September 5, 2019

Title Easy management of variable labels in R data sets

Version 1.0.1

Description
labelmachine is an R package that helps you assigning new labels to data.frame variables. Furthermore, you can manage your label translations in yaml files. This makes it very easy using the same label translations in multiple projects that share similar data structure.

Encoding UTF-8

Depends R (>= 3.3.0)

Remotes a-maldet/composerr

Imports composerr, yaml (>= 2.2.0)

License GPL-3 + file LICENSE

Roxygen list(markdown = TRUE)

RoxygenNote 6.1.1

Collate 'lappli.R' 'utilities.R' 'dictionary.R' 'imports.R'
'lama_check.R' 'lama_merge.R' 'lama_mutate.R' 'lama_read.R'
'lama_select.R' 'lama_rename.R' 'lama_translate.R'
'lama_write.R'

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

NeedsCompilation no

Author Adrian Maldet [aut, cre]

Maintainer Adrian Maldet <adrian.maldet@statistik.gv.at>

R topics documented:

`as.dictionary` *Coerce to a LabelDictionary 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 ore more column pairs. Each column pair consists of a column holding the original values, which should be relapced, and a second charachter 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

```
as.dictionary(.data, ...)

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

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

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

## S3 method for class 'data.frame'
as.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 should be orde: • `"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 LabelDictionary 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 (`NA`s). In this case, the original values are mappend to 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 sitations:

- All missing values should be labelled (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 recognice missing values.

LabelDictionary class objects

Each *LabelDictionary* class object can contain multiple *translations*, each with a unique name under which the translation can be found. The function `lama_translate()` uses a LabelDictionary 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")`).

`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 LabelDictionary 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 LabelDictionary 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

`check_translate_general` *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_translate_general(.data, dictionary, col_new, keep_order,
  err_handler)
```

Arguments

`.data`	The data.frame object which contains the variable that should be relabelled
`dictionary`	A LabelDictionary object, holding the translations for various variables.
`col_new`	A character vector of the same length as `translation` holding the names under which the relabelled 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 arguments in .... 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.
err_handler	An error handling function

`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 LabelDictionary class input format to the yaml format*

Description

In the LabelDictionary 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)
```

Arguments

data	A pre-dictionary object.

Value

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

`escape_to_na` *Replace* `"NA_"` *by* `NA`

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 `NA`s.

`is.dictionary` *Check if an object is a LabelDictionary class object*

Description

Check if an object is a LabelDictionary class object

Usage

```
is.dictionary(obj)
```

Arguments

`obj`	The object in question

Value

`TRUE` if the object is a LabelDictionary class object, `FALSE` otherwise.

See Also

`validate_dictionary()`, `as.dictionary()`, `new_dictionary()`, `lama_translate()`, `lama_read()`, `lama_write()`, `lama_select()`, `lama_rename()`, `lama_mutate()`, `lama_merge()`

is.syntactic *Check if a variable name is syntactically valid*

Description

This function was suggested by Hadley Wickham in a forum

Usage

```
is.syntactic(x)
```

Arguments

x	A character string that should be checked, if it conatains a valid object name.

Value

TRUE if valid, FALSE else.

References

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

lama_check_character *Check if an object is a character*

Description

Check if an object is a character

Usage

```
lama_check_character(obj, len_1 = TRUE, allow_null = FALSE,
  allow_empty = FALSE, allow_na = FALSE,
  err_handler = composerr("Error in 'lama_check_character"))
```

Arguments

obj	An object that should be checked if it has the right length
len_1	A flag that tells if obj should be of length 1
allow_null	A flag that tells if NULL is allowed.
allow_empty	A flag that tells if "" is allowed.
allow_na	A flag that tells if NA is allowed.
err_handler	An error handling function, which should be used

lama_check_length *Check if an object has the right length*

Description

Check if an object has the right length

Usage

```
lama_check_length(obj, comp_obj = NULL, comp_len = NULL,
  err_handler = composerr("Error in 'lama_check_length"))
```

Arguments

obj	An object that should be checked if it has the right length
comp_obj	A object that should be used for comparison
comp_len	A numeric vector holding valid length values
err_handler	An error handling function, which should be used

lama_merge *Merge multiple label lexicas into one*

Description

This function takes multiple LabelDictionary class objects and merges them together into a single LabelDictionary 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

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

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

Arguments

...	Two or more LabelDictionary 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 LabelDictionary class object

See Also

lama_translate(), new_dictionary(), lama_rename(), lama_select(), lama_mutate(), lama_read(), lama_write()

---

lama_mutate *Change or append a variable translation to an existing LabelDictionary object*

Description

The functions lama_mutate() and lama_mutate_() alter a LabelDictionary object. They either alter or append a translation to a LabelDictionary 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 'LabelDictionary'
lama_mutate(.data, ...)

lama_mutate_(.data, key, translation)

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

Arguments

.data	A LabelDictionary object
...	One or more unquoted expressions separated by commas. Use named arguments, e.g. new_transation_name = c(a = "A",b = "B"), to set translations (named character vectors) to new translation names. 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 LabelDictionary class object.

See Also

lama_translate(), new_dictionary(), lama_rename(), lama_select(), lama_merge(), lama_read(), lama_write()

---

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

```
lama_read(yaml_path)
```

Arguments

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

Value

A LabelDictionary class object holding the variable translations defined in the yaml file

---

lama_rename *Rename multiple variable translations in a LabelDictionary object*

Description

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

Usage

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

lama_rename_(.data, old, new)

## S3 method for class 'LabelDictionary'
lama_rename_(.data, old, new)
```

Arguments

.data	A LabelDictionary 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 LabelDictionary class object.

See Also

lama_translate(), new_dictionary(), lama_select(), lama_mutate(), lama_merge(), lama_read(), lama_write()

---

lama_select *Select multiple variable translations and create a new LabelDictionary object*

Description

The functions lama_select() and lama_select_() pick one or more variable translations from a LabelDictionary class object and create a new LabelDictionary class object. The function lama_select() uses non-standard evaluation, whereas lama_select_() is the standard evaluation alternative.

Usage

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

lama_select_(.data, key)

## S3 method for class 'LabelDictionary'
lama_select_(.data, key)
```

Arguments

.data	A LabelDictionary 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 LabelDictionary class object, holding the picked variable translations.

See Also

lama_translate(), new_dictionary(), lama_rename(), lama_mutate(), lama_merge(), lama_read(), lama_write()

---

lama_translate *Assign new labels to a variable of a data.frame*

Description

The functions lama_translate() and lama_translate_() take a data.frame and convert one or more of its categorical variables (not necessarely a factor variable) into factor variables with new labels. The function lama_translate() uses non-standard evaluation, whereas lama_translate_() is the standard evaluation alternative.

Usage

```
lama_translate(.data, dictionary, ..., keep_order = FALSE)

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

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

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

Arguments

.data	The data.frame object which contains the variable that should be relabelled
dictionary	A LabelDictionary object, holding the translations for various variables.
...	One or more unquoted expressions separated by commas. Use unquoted arguments that tell which translation should be applied to which column and which column name the relabelled variable should be assigned to. E.g. lama_trans(.data,dict,Y1 = TRANS1(X1),Y2 = TRANS2(Y2)) to apply the translations TRANS1 and TRANS2 to the data.frame variables X1 and X2 and save the new labelled variables under the names Y1 and Y2. There are also two abbreviation mechanisms available: The argument assignement FOO(X) is the same as X = FOO(X) and FOO is an abbreviation for FOO = FOO(FOO).
keep_order	A boolean vector of length one or the same length as the number of arguments in .... 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.

`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` A character vector of the same length as `translation` holding the names of the data.frame columns that should be relabelled. 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` A character vector of the same length as `translation` holding the names under which the relabelled 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.

Value

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

`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

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

Arguments

`x` A LabelDictionary class object holding the variable translations

`yaml_path` File path, where the yaml file should be saved

lapplI *Improve* lapply and *sapply' with index*

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

```
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.
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.

Examples

```
# 'lapply' with index
lapplI(
  list("x1", "x2"),
  function(x, y, .I) list(x = x, y = y, i = .I),
  y = "extra argument"
)

# 'lapply' without index
lapplI(
```

```
  list("x1", "x2"),
  function(x, y) list(x = x, y = y),
  y = "extra argument"
)
# 'sapply' with index
sapplI(
  c("x1", "x2"),
  function(x, y, .I) paste(x, y, .I),
  y = "extra argument",
  USE.NAMES = FALSE
)

# 'sapply' without index
sapplI(
  c("x1", "x2"),
  function(x, y) paste(x, y),
  y = "extra argument",
  USE.NAMES = FALSE
)
```

---

named_lapply *Create a named list with lapply from a character vector*

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 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.

Value

A named list

named_list *Create a named list*

Description

Create a named list

Usage

```
named_list(.names, obj)
```

Arguments

.names	A character vector holding the names of the list
obj	A vector or list object of the same length

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.

na_to_escape *Replace* NA *by* "NA_"

Description

Replace NA by "NA_"

Usage

```
na_to_escape(x)
```

Arguments

x	A character vector that should be modified.

Value

A character vector, where the NAs are replaced.

new_dictionary *Create a new LabelDictionary class object*

Description

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

- *No arguments* were passed into ...: In this case new_dictionary returns an empty LabelDictionary class object (e.g. dict <-new_dictionary()).
- *The first argument is a list*: In this case only the first argument of new_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 LabelDictionary class object. Each item of the named list object must be a *named character vector* defining a translation (e.g. new_dictionary(list(area = c("0" = "urban","1" = "rural"),= c(l = "Low",h = "High"))) generates a LabelDictionary 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_dictionary(area = c("0" = "urban","1" = "rural"),density = c(l = "Low",h = "High")) generates a LabelDictionary 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 LabelDictionary class object.

Usage

```
new_dictionary(...)

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

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

## Default S3 method:
new_dictionary(...)
```

Arguments

`...`	None, one or more named/unnamed arguments. Depending on the type of the type of the first argument passed into `new_dictionary`, there are different valid ways of using `new_dictionary`: • *No arguments* were passed into `...`: In this case `new_dictionary` returns an empty LabelDictionary class object (e.g. `dict <-new_dictionary()`). • *The first argument is a list*: In this case, only the first argument of `new_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 LabelDictionary class object. Each item of the named list object must be a named character vector defining a translation (e.g. `new_dictionary(list(area = c("0" = "urban","1" = "rural"),= c(l = "Low",h = "High")))` generates a LabelDictionary 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_directory` must be *named arguments* and each argument must be a named character vectors defining translations (e.g. `new_dictionary(area = c("0" = "urban","1" = "rural"),density = c(l = "Low",h = "High"))` generates a LabelDictionary 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 LabelDictionary class object.
`.data`	A named list object, where each list entry corresponds to a translation that should be added to the LabelDictionary object (e.g. `new_dictionary(list(area = c("0" = "urban","1" = "rural"),= c(l = "Low",h = "High")))` generates a LabelDictionary 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 LabelDictionary 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 LabelDictionary 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 (`NA`s). In this case, the original values are mappend to 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 sitations:

- All missing values should be labelled (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 recognice missing values.

LabelDictionary class objects

Each *LabelDictionary* class object can contain multiple *translations*, each with a unique name under which the translation can be found. The function `lama_translate()` uses a LabelDictionary 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.dictionary()`, `lama_translate()`, `lama_read()`, `lama_write()`, `lama_select()`, `lama_rename()`, `lama_mutate()`, `lama_merge()`

---

print.LabelDictionary *Print a LabelDictionary class object*

Description

Print a LabelDictionary class object

Usage

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

Arguments

`x`	The LabelDictionary class object that should be printed.
`...`	Unused arguments

See Also

`lama_translate()`, `new_dictionary()`, `lama_select()`, `lama_rename()`, `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 LabelDictionary 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 LabelDictionary 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,
  err_handler)
```

Arguments

.data	The data.frame object which contains the variable that should be relabelled
dictionary	A LabelDictionary 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	A character vector of the same length as `translation` holding the names of the data.frame columns that should be relabelled. 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	A character vector of the same length as `translation` holding the names under which the relabelled 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 arguments in `....` 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.
err_handler	An error handling function

Value

An factor vector holding the assigned labels.

`translate_variable` *This function relabels a vector*

Description

This function relabels a vector

Usage

```
translate_variable(val, translation, keep_order, err_handler)
```

Arguments

`val`	The vector that should be relabelled. Allowed are all vector types (also factor).
`translation`	Named character vector holding the label assignements.
`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.
`err_handler`	An error handling function

Value

A factor vector holding the assigned labels

`validate_dictionary` *Check if an object has a valid LabelDictionary structure*

Description

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

Usage

```
validate_dictionary(obj,
  err_handler = composerr("The object has not a valid LabelDictionary 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 (`NA`s). In this case, the original values are mappend to 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 sitations:

- All missing values should be labelled (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 recognice missing values.

LabelDictionary class objects

Each *LabelDictionary* class object can contain multiple *translations*, each with a unique name under which the translation can be found. The function `lama_translate()` uses a LabelDictionary 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.dictionary()`, `as.dictionary()`, `new_dictionary()`, `lama_translate()`, `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

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

Arguments

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

---

yaml_to_dictionary *Transform data structure from yaml format to the LabelDictionary class input format*

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 LabelDictionary class input structure vars (named list) > translations (named character vector)

Usage

```
yaml_to_dictionary(data)
```

Arguments

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

Value

A pre-dictionary object.

Index