Package 'dunlin'

December 13, 2023

Title Preprocessing Tools for Clinical Trial Data

Version 0.1.7

Date 2023-12-04

Description A collection of functions to preprocess data and organize
them in a format amenable to use by chevron.

License Apache License 2.0

URL https://insightsengineering.github.io/dunlin/,
https://github.com/insightsengineering/dunlin/

BugReports https://github.com/insightsengineering/dunlin/issues

Depends R (>= 4.0.0)

Imports checkmate (>= 2.1.0), dplyr (>= 1.1.0), forcats (>= 1.0.0),
glue (>= 1.0.0), magrittr (>= 1.5), methods, rlang (>= 1.0.0),
stringr (>= 1.4.1), tibble (>= 1.2), yaml (>= 2.1.15)

Suggests knitr (>= 1.42), rmarkdown (>= 2.19), testthat (>= 3.0.4),
withr (>= 2.1.0)

VignetteBuilder knitr

Config/Needs/verdepcheck mllg/checkmate, tidyverse/dplyr,
tidyverse/forcats, tidyverse/glue, tidyverse/magrittr,
r-lib/rlang, tidyverse/stringr, tidyverse/tibble,
yaml=vubiostat/r-yaml, yihui/knitr, rstudio/rmarkdown,
r-lib/testthat, r-lib/withr

Config/Needs/website insightsengineering/nesttemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Collate 'assertions.R' 'co_relevels.R' 'cut_by_group.R'
'dunlin-package.R' 'explicit_na.R' 'filter.R'
'join_adsub_adsl.R' 'pivot.R' 'propagate.R' 'reformat.R'
'render_safe.R' 'rules.R' 'unite.R' 'utils.R' 'zzz.R'

NeedsCompilation no

Author Liming Li [aut, cre],
Benoit Falquet [aut],
Xiaoli Duan [ctb],
Pawel Rucki [ctb],
F. Hoffmann-La Roche AG [cph, fnd]

Maintainer Liming Li <liming.li@roche.com>

Repository CRAN

Date/Publication 2023-12-13 12:30:02 UTC

R topics documented:

`dunlin-package` `Dunlin` *Package*

Description

A collection of functions to preprocess data and organize them in a format amenable to use by chevron.

Author(s)

Maintainer: Liming Li <`liming.li@roche.com`>

Authors:

- Benoit Falquet <`benoit.falquet@roche.com`>

Other contributors:

- Xiaoli Duan <`xiaoli.duan@roche.com`> [contributor]
- Pawel Rucki <`pawel.rucki@roche.com`> [contributor]
- F. Hoffmann-La Roche AG [copyright holder, funder]

See Also

Useful links:

- `https://insightsengineering.github.io/dunlin/`
- `https://github.com/insightsengineering/dunlin/`
- Report bugs at `https://github.com/insightsengineering/dunlin/issues`

`add_whisker` *Add whisker values*

Description

Add whisker values

Usage

```
add_whisker(x)
```

Arguments

`x`	Named (`character`) input.

Details

The names of the character gives the string to be replaced and the value gives the new string.

Value

invisible `NULL`. Assign the key-value pair provided as argument in the whisker environment.

Examples

```
my_whiskers <- c(Placeholder = "Replacement", Placeholder2 = "Replacement2")
add_whisker(my_whiskers)
```

`as.list.rule` *Convert Rule to List*

Description

Convert Rule to List

Usage

```
## S3 method for class 'rule'
as.list(x, ...)
```

Arguments

`x`	(`rule`) to convert.
`...`	not used.

Value

an object of class `list`.

Examples

```
x <- rule("a" = c("a", "b"), "X" = "x", .to_NA = c("v", "w"))
as.list(x)
```

`assert_all_tablenames` *Assert that all names are among names of a* `list` *of* `data.frame`*.*

Description

Assert that all names are among names of a `list` of `data.frame`.

Usage

```
assert_all_tablenames(db, tab, null_ok = TRUE, qualifier = NULL)
```

Arguments

`db`	(`list` of `data.frame`) input to check for the presence of tables.
`tab`	(`character`) the names of the tables to be checked.
`null_ok`	(`flag`) can x be NULL.
`qualifier`	(`string`) to be returned if the check fails.

Value

invisible `TRUE` or an error message if the criteria are not fulfilled.

Examples

```
lsd <- list(
  mtcars = mtcars,
  iris = iris
)
assert_all_tablenames(lsd, c("mtcars", "iris"), qualifier = "first test:")
```

`assert_valid_format` *Assert Nested List can be used as Format Argument in Reformat.*

Description

Assert Nested List can be used as Format Argument in Reformat.

Usage

```
assert_valid_format(object)
```

Arguments

`object`	(`list`) to assert.

Value

invisible TRUE or an error message if the criteria are not fulfilled.

Examples

```
format <- list(
  df1 = list(
    var1 = rule("X" = "x", "N" = c(NA, ""))
  ),
  df2 = list(
    var1 = rule(),
    var2 = rule("f11" = "F11", "NN" = NA)
  ),
  df3 = list()
)

assert_valid_format(format)
```

---

assert_valid_list_format *Assert List can be Converted into a Nested List Compatible with the Format Argument of Reformat.*

Description

Assert List can be Converted into a Nested List Compatible with the Format Argument of Reformat.

Usage

```
assert_valid_list_format(object)
```

Arguments

object	(list) to assert.

Value

invisible TRUE or an error message if the criteria are not fulfilled.

Examples

```
format <- list(
  df1 = list(
    var1 = list("X" = "x", "N" = c(NA, ""))
  ),
  df2 = list(
    var1 = list(),
    var2 = list("f11" = "F11", "NN" = NA)
  ),
  df3 = list()
)

assert_valid_list_format(format)
```

---

attr_label *Setting the Label Attribute*

Description

Setting the Label Attribute

Usage

```
attr_label(var, label)
```

Arguments

`var`	(`object`) whose label attribute can be set.
`label`	(`character`) the label to add.

Value

`object` with label attribute.

Examples

```
x <- c(1:10)
attr(x, "label")

y <- attr_label(x, "my_label")
attr(y, "label")
```

---

attr_label_df *Setting the Label Attribute to Data Frame Columns*

Description

Setting the Label Attribute to Data Frame Columns

Usage

```
attr_label_df(df, label)
```

Arguments

df	(data.frame).
label	(character) the labels to add.

Value

data.frame with label attributes.

Examples

```
res <- attr_label_df(mtcars, letters[1:11])
res
lapply(res, attr, "label")
```

co_relevels *Reorder Two Columns Levels Simultaneously*

Description

Reorder Two Columns Levels Simultaneously

Usage

```
co_relevels(df, primary, secondary, levels_primary)
```

Arguments

df	(data.frame) with two column whose factors should be reordered.
primary	(string) the name of the column on which the levels reordering should be based.
secondary	(string) the name of the column whose levels should be reordered following the levels of the primary column.
levels_primary	(character) the levels in the desired order. Existing levels that are not included will be placed afterward in their current order.

Details

The function expect a 1:1 matching between the elements of the two selected column.

Value

a data.frame with the secondary column converted to factor with reordered levels.

Examples

```
df <- data.frame(
  SUBJID = 1:3,
  PARAMCD = factor(c("A", "B", "C")),
  PARAM = factor(paste("letter", LETTERS[1:3]))
)
co_relevels(df, "PARAMCD", "PARAM", levels_primary = c("C", "A", "B"))
```

`cut_by_group` *Cutting data by group*

Description

Cutting data by group

Usage

```
cut_by_group(df, col_data, col_group, group, cat_col)
```

Arguments

`df`	(`dataframe`) with a column of data to be cut and a column specifying the group of each observation.
`col_data`	(`character`) the column containing the data to be cut.
`col_group`	(`character`) the column containing the names of the groups according to which the data should be split.
`group`	(`nested list`) providing for each parameter value that should be analyzed in a categorical way: the name of the parameter (`character`), a series of breakpoints (`numeric`) where the first breakpoints is typically `-Inf` and the last `Inf`, and a series of name which will describe each category (`character`).
`cat_col`	(`character`) the name of the new column in which the cut label should he stored.

Details

Function used to categorize numeric data stored in long format depending on their group. Intervals are closed on the right (and open on the left).

Value

`data.frame` with a column containing categorical values.

Examples

```
group <- list(
  list(
    "Height",
    c(-Inf, 150, 170, Inf),
    c("=<150", "150-170", ">170")
  ),
  list(
    "Weight",
    c(-Inf, 65, Inf),
    c("=<65", ">65")
  ),
  list(
    "Age",
    c(-Inf, 31, Inf),
    c("=<31", ">31")
  ),
  list(
    "PreCondition",
    c(-Inf, 1, Inf),
    c("=<1", "<1")
  )
)
data <- data.frame(
  SUBJECT = rep(letters[1:10], 4),
  PARAM = rep(c("Height", "Weight", "Age", "other"), each = 10),
  AVAL = c(rnorm(10, 165, 15), rnorm(10, 65, 5), runif(10, 18, 65), rnorm(10, 0, 1)),
  index = 1:40
)

cut_by_group(data, "AVAL", "PARAM", group, "my_new_categories")
```

`get_log` *Get Log*

Description

Get Log

Usage

```
get_log(data, incl, incl.adsl)

## S3 method for class 'data.frame'
get_log(data, incl = TRUE, incl.adsl = TRUE)

## S3 method for class 'list'
get_log(data, incl = TRUE, incl.adsl = TRUE)
```

Arguments

data	(list of data.frame or data.frame) filtered with log_filter.
incl	(flag) should information about unfiltered data.frame be printed.
incl.adsl	(flag) should indication of filtering performed through adsl be printed.

Value

character or list of character describing the filtering applied to data.

Examples

```
data <- log_filter(iris, Sepal.Length >= 7, "xx")
data <- log_filter(data, Sepal.Length < 2)
data <- log_filter(data, Sepal.Length >= 2, "yy")
get_log(data)

data <- log_filter(
  list(iris1 = iris, iris2 = iris),
  Sepal.Length >= 7,
  "iris1",
  character(0),
  "Sep"
)
get_log(data)
```

---

join_adsub_adsl *Join* adsub *to* adsl

Description

Join adsub to adsl

Usage

```
join_adsub_adsl(
  adam_db,
  keys,
  continuous_var,
  categorical_var,
  continuous_suffix,
  categorical_suffix,
  drop_na = TRUE,
  drop_lvl = TRUE
)

## S3 method for class 'list'
```

```
join_adsub_adsl(
  adam_db,
  keys = c("USUBJID", "STUDYID"),
  continuous_var = "all",
  categorical_var = "all",
  continuous_suffix = "",
  categorical_suffix = "_CAT",
  drop_na = TRUE,
  drop_lvl = FALSE
)
```

Arguments

`adam_db`	(`list` of `data.frame`) object input with an `adsl` and `adsub` table.
`keys`	(`character`) the name of the columns in `adsl` uniquely identifying a row.
`continuous_var`	(`character`) the value of a parameter in the `PARAMCD` column of the `adsub` table from which columns containing continuous values should be created. If `"all"`, all parameter values are selected, if `NULL`, none are selected.
`categorical_var`	(`character`) the value of a parameter in the `PARAMCD` column of the `adsub` table from which columns containing categorical values should be created. If `"all"`, all parameter values are selected, if `NULL`, none are selected.
`continuous_suffix`	(`string`) the suffixes to add to the newly generated columns containing continuous values.
`categorical_suffix`	(`string`) the suffixes to add to the newly generated columns containing categorical values.
`drop_na`	(`logical`) whether resulting columns containing only `NA`s should be dropped.
`drop_lvl`	(`logical`) should missing levels be dropped in the resulting columns.

Value

a `list` of `data.frame` with new columns in the `adsl` table.

Examples

```
adsl <- data.frame(
  USUBJID = c("S1", "S2", "S3", "S4"),
  STUDYID = "My_study",
  AGE = c(60, 44, 23, 31)
)

adsub <- data.frame(
  USUBJID = c("S1", "S2", "S3", "S4", "S1", "S2", "S3"),
  STUDYID = "My_study",
  PARAM = c("weight", "weight", "weight", "weight", "height", "height", "height"),
  PARAMCD = c("w", "w", "w", "w", "h", "h", "h"),
  AVAL = c(98, 75, 70, 71, 182, 155, 152),
  AVALC = c(">80", "<=80", "<=80", "<=80", ">180", "<=180", "<=180")
)

db <- list(adsl = adsl, adsub = adsub)

x <- join_adsub_adsl(adam_db = db)
x <- join_adsub_adsl(adam_db = db, continuous_var = c("w", "h"), categorical_var = "h")
```

`list2rules` *Convert nested list into list of* `rule`

Description

Convert nested list into list of `rule`

Usage

```
list2rules(obj)
```

Arguments

`obj`	(`nested list`) to convert into list of rules.

Value

a `list` of `rule` objects.

Examples

```
obj <- list(
  rule1 = list("X" = c("a", "b"), "Z" = "c", .to_NA = "xxxx"),
  rule2 = list(Missing = c(NA, "")),
  rule3 = list(Missing = c(NA, ""), .drop = TRUE),
  rule4 = list(Absent = c(NA, ""), .drop = TRUE, .to_NA = "yyyy")
)
list2rules(obj)
```

`log_filter` *Filter Data with Log*

Description

Filter Data with Log

Usage

```
log_filter(data, condition, ...)

## S3 method for class 'data.frame'
log_filter(data, condition, suffix = NULL, ...)

## S3 method for class 'list'
log_filter(
  data,
  condition,
  table,
  by = c("USUBJID", "STUDYID"),
  suffix = NULL,
  ...
)
```

Arguments

data	(`data.frame`) input data to subset, or named (`list` of `data.frame`).
condition	(`call`) of subset condition. Must evaluate as logical.
...	further arguments to be passed to or from other methods.
suffix	(`string`) optional argument describing the filter.
table	(`string`) table name.
by	(`character`) variable names shared by `adsl` and other datasets for filtering.

Details

`log_filter` will filter the data/named list of data according to the `condition`. All the variables in `condition` must exist in the data (as variables) or in the parent frame(e.g., in global environment). For named list of data, if `ADSL` is available, `log_filter` will also try to subset all other datasets with `USUBJID`.

Value

a `data.frame` or `list` of `data.frame` filtered for the provided conditions.

Examples

```
data <- iris
attr(data$Sepal.Length, "label") <- "cm"
log_filter(data, Sepal.Length >= 7)

log_filter(list(iris = iris), Sepal.Length >= 7, "iris", character(0))
```

---

ls_explicit_na   *Encode Categorical Missing Values in a* `list` *of* `data.frame`

Description

Encode Categorical Missing Values in a `list` of `data.frame`

Usage

```
ls_explicit_na(
  data,
  omit_tables = NULL,
  omit_columns = NULL,
  char_as_factor = TRUE,
  na_level = "<Missing>"
)
```

Arguments

`data`	(`list` of `data.frame`) to be transformed.
`omit_tables`	(`character`) the names of the tables to omit from processing.
`omit_columns`	(`character`) the names of the columns to omit from processing.
`char_as_factor`	(`logical`) should character columns be converted into factor.
`na_level`	(`string`) the label to encode missing levels.

Details

This is a helper function to encode missing values (i.e `NA` and `empty string`) of every `character` and `factor` variable found in a `list` of `data.frame`. The `label` attribute of the columns is preserved.

Value

`list` of `data.frame` object with explicit missing levels.

Examples

```
df1 <- data.frame(
  "char" = c("a", "b", NA, "a", "k", "x"),
  "char2" = c("A", "B", NA, "A", "K", "X"),
  "fact" = factor(c("f1", "f2", NA, NA, "f1", "f1")),
  "logi" = c(NA, FALSE, TRUE, NA, FALSE, NA)
)
df2 <- data.frame(
  "char" = c("a", "b", NA, "a", "k", "x"),
  "fact" = factor(c("f1", "f2", NA, NA, "f1", "f1")),
  "num" = c(1:5, NA)
)
df3 <- data.frame(
  "char" = c(NA, NA, "A")
)

db <- list(df1 = df1, df2 = df2, df3 = df3)

ls_explicit_na(db)
ls_explicit_na(db, omit_tables = "df3", omit_columns = "char2")
```

`ls_unite` *Unite Columns of a Table in a* `list` *of* `data.frame`.

Description

Unite Columns of a Table in a `list` of `data.frame`.

Usage

```
ls_unite(adam_db, tab, cols, sep = ".", new = NULL)
```

Arguments

`adam_db`	(`list` of `data.frames`) to be transformed.
`tab`	(`string`) the name of a table in the `adam_db` object.
`cols`	(`character`) the name of the columns to unite.
`sep`	(`string`) the separator for the new column name.
`new`	(`string`) the name of the new column. If `NULL` the concatenation of `cols` separated by `sep` is used.

Value

`list` of `data.frames` object with a united column.

Examples

```
db <- list(mtcars = mtcars, iris = iris)

x <- ls_unite(db, "mtcars", c("mpg", "hp"), new = "FUSION")
x$mtcars
```

`multi_id_pivot_wider` *Transforming data.frame with Multiple Identifying columns into Wide Format*

Description

Transforming data.frame with Multiple Identifying columns into Wide Format

Usage

```
multi_id_pivot_wider(
  data,
  id,
  param_from,
  value_from,
  drop_na = FALSE,
  drop_lvl = FALSE
)
```

Arguments

`data`	(`data.frame`) to be pivoted.
`id`	(`character`) the name of the columns whose combination uniquely identify the observations.
`param_from`	(`character`) the name of the column containing the names of the parameters to be pivoted. The unique values in this column will become column names in the output.
`value_from`	(`character`) the name of the column containing the values that will populate the output.
`drop_na`	(`logical`) should column containing only `NA`s be dropped.
`drop_lvl`	(`logical`) should missing levels be dropped in the columns coming from (`value_from`).

Details

This function allows to identify observations on the basis of several columns. Warning: Instead of nesting duplicated values, the function will throw an error if the same parameter is provided twice for the same observation.

Value

`data.frame` in a wide format.

Examples

```
test_data <- data.frame(
  the_obs = c("A", "A", "A", "B", "B", "B", "C", "D"),
  the_obs2 = c("Ax", "Ax", "Ax", "Bx", "Bx", "Bx", "Cx", "Dx"),
  the_param = c("weight", "height", "gender", "weight", "gender", "height", "height", "other"),
  the_val = c(65, 165, "M", 66, "F", 166, 155, TRUE)
)

multi_id_pivot_wider(test_data, c("the_obs", "the_obs2"), "the_param", "the_val")
multi_id_pivot_wider(test_data, "the_obs2", "the_param", "the_val")
```

poly_pivot_wider *Transforming data.frame with multiple Data Column into Wide Format*

Description

Transforming data.frame with multiple Data Column into Wide Format

Usage

```
poly_pivot_wider(
  data,
  id,
  param_from,
  value_from,
  labels_from = NULL,
  drop_na = TRUE,
  drop_lvl = FALSE
)
```

Arguments

data	(`data.frame`) to be pivoted.
id	(`character`) the name of the columns whose combination uniquely identify the observations.
param_from	(`character`) the name of the columns containing the names of the parameters to be pivoted. The unique values in this column will become column names in the output.
value_from	(`character`) the name of the column containing the values that will populate the output.
labels_from	(`character`) the name of the column congaing the labels of the new columns. from. If not provided, the labels will be equal to the column names. When several labels are available for the same column, the first one will be selected.
drop_na	(`logical`) should column containing only `NA`s be dropped.
drop_lvl	(`logical`) should missing levels be dropped in the columns coming from `value_from`.

Details

This function is adapted to cases where the data are distributed in several columns while the name of the parameter is in one. Typical example is adsub where numeric data are stored in AVAL while categorical data are in AVALC.

Value

list of data.frame in a wide format with label attribute attached to each columns.

Examples

```
test_data <- data.frame(
  the_obs = c("A", "A", "A", "B", "B", "B", "C", "D"),
  the_obs2 = c("Ax", "Ax", "Ax", "Bx", "Bx", "Bx", "Cx", "Dx"),
 the_param = c("weight", "height", "gender", "weight", "gender", "height", "height", "other"),
  the_label = c(
    "Weight (Kg)", "Height (cm)", "Gender", "Weight (Kg)",
    "Gender", "Height (cm)", "Height (cm)", "Pre-condition"
  ),
  the_val = c(65, 165, NA, 66, NA, 166, 155, NA),
  the_val2 = c(65, 165, "M", 66, "F", 166, 155, TRUE)
)

x <- poly_pivot_wider(
  test_data,
  c("the_obs", "the_obs2"),
  "the_param",
  c("the_val", "the_val2"),
  "the_label"
)
x
Reduce(function(u, v) merge(u, v, all = TRUE), x)
```

---

print_log *Print Log*

Description

Print Log

Usage

```
print_log(data, incl, incl.adsl)

## S3 method for class 'data.frame'
print_log(data, incl = TRUE, incl.adsl = TRUE)

## S3 method for class 'list'
print_log(data, incl = TRUE, incl.adsl = TRUE)
```

Arguments

`data`	(`list` of `data.frame` or `data.frame`) filtered with `log_filter`.
`incl`	(`flag`) should information about unfiltered `data.frame` be printed.
`incl.adsl`	(`flag`) should indication of filtering performed through `adsl` be printed.

Value

`NULL`. Print a description of the filtering applied to `data`.

Examples

```
data <- log_filter(iris, Sepal.Length >= 7, "Sep")
print_log(data)
data <- log_filter(
  list(
    adsl = iris,
    iris2 = iris,
    mtcars = mtcars,
    iris3 = iris
  ),
  Sepal.Length >= 7,
  "adsl",
  character(0),
  "adsl filter"
)
data <- log_filter(data, Sepal.Length >= 7, "iris2", character(0), "iris2 filter")
print_log(data)
print_log(data, incl = FALSE)
print_log(data, incl.adsl = FALSE, incl = FALSE)
```

`propagate` *Propagate Column*

Description

`propagate`copy columns from a given table of a `list` of `data.frame` to all tables based on other common columns. If several rows are associated with the same key, the rows will be duplicated in the receiving tables. In safe mode, the key must be unique in the original table.

Usage

```
propagate(db, from, add, by, safe = TRUE)

## S3 method for class 'list'
propagate(db, from, add, by, safe = TRUE)
```

Arguments

db	(list of data.frame) object for which some variable need to be propagated.
from	(string) the name of the table where the variables to propagate are stored.
add	(character) the names of the variables to propagate.
by	(character) the key binding the from table to the other tables.
safe	(flag) should the key be checked for uniqueness in the from table.

Value

updated list of data.frame.

Examples

```
df1 <- data.frame(
  id1 = c("a", "a", "c", "d", "e", "f"),
  id2 = c("A", "B", "A", "A", "A", "A"),
  int = c(1, 2, 3, 4, 5, 6),
  bool = c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE)
)

df2 <- data.frame(
  id1 = c("a", "a", "d", "e", "f", "g"),
  id2 = c("A", "B", "A", "A", "A", "A")
)

df3 <- data.frame(
  id1 = c("a", "c", "d", "e", "f", "x"),
  id2 = c("A", "A", "A", "A", "B", "A"),
  int = c(11, 22, 33, 44, 55, 66)
)

db <- list(df1 = df1, fd2 = df2, df3 = df3)
propagate(db, from = "df1", add = c("int", "bool"), by = c("id1", "id2"))
```

---

reformat *Reformat Values*

Description

Reformat Values

Usage

```
reformat(obj, ...)

## Default S3 method:
reformat(obj, format, ...)

## S3 method for class 'character'
reformat(obj, format, ...)

## S3 method for class 'factor'
reformat(obj, format, ...)

## S3 method for class 'list'
reformat(obj, format, ...)
```

Arguments

`obj`	(`character`, `factor` or `list of data.frame`) to reformat.
`...`	for compatibility between methods and pass additional special mapping to transform rules. • `.string_as_fct` (`flag`) whether the reformatted character object should be converted to factor. • `.to_NA` (`character`) values that should be converted to `NA`. For `factor`, the corresponding levels are dropped. If `NULL`, the argument will be taken from the `to_NA`attribute of the rule. • `.drop` (`flag`) whether to drop empty levels. If `NULL`, the argument will be taken from the `drop`attribute of the rule. • `.na_last` (`flag`) whether the level replacing `NA` should be last.
`format`	(`rule`) or (`list`) of `rule` depending on the class of obj.

Value

(`character`, `factor` or `list of data.frame`) with remapped values.

Note

When the rule is empty rule or when values subject to reformatting are absent from the object, no error is raised. The conversion to factor if `.string_as_fct = TRUE`) is still carried out. The conversion of the levels declared in `.to_NA` to `NA` values occurs after the remapping. `NA` values created this way are not affected by a rule declaring a remapping of `NA` values. For factors, level dropping is the last step, hence, levels converted to `NA` by the `.to_NA` argument, will be removed if `.drop` is `TRUE`. Arguments passed via `reformat` override the ones defined during rule creation.

the variables listed under the `all_dataset` keyword will be reformatted with the corresponding rule in every data set except where another rule is specified for the same variable under a specific data set name.

Examples

```
# Reformatting of character.
obj <- c("a", "b", "x", NA, "")
attr(obj, "label") <- "my label"
format <- rule("A" = "a", "NN" = NA)

reformat(obj, format)
reformat(obj, format, .string_as_fct = FALSE, .to_NA = NULL)


# Reformatting of factor.
obj <- factor(c("first", "a", "aa", "b", "x", NA), levels = c("first", "x", "b", "aa", "a", "z"))
attr(obj, "label") <- "my label"
format <- rule("A" = c("a", "aa"), "NN" = c(NA, "x"), "Not_present" = "z", "Not_a_level" = "P")

reformat(obj, format)
reformat(obj, format, .na_last = FALSE, .to_NA = "b", .drop = FALSE)


# Reformatting of list of data.frame.
df1 <- data.frame(
  var1 = c("a", "b", NA),
  var2 = factor(c("F1", "F2", NA))
)

df2 <- data.frame(
  var1 = c("x", NA, "y"),
  var2 = factor(c("F11", NA, "F22"))
)

db <- list(df1 = df1, df2 = df2)

format <- list(
  df1 = list(
    var1 = rule("X" = "x", "N" = NA, .to_NA = "b")
  ),
  df2 = list(
    var2 = rule("f11" = "F11", "NN" = NA)
  ),
  all_datasets = list(
    var1 = rule("xx" = "x", "aa" = "a")
  )
)

reformat(db, format)
```

`remove_whisker` *Remove whisker values*

Description

Remove whisker values

Usage

```
remove_whisker(x)
```

Arguments

`x`	Named (`character`) input.

Value

invisible `NULL`. Removes `x` from the whisker environment.

`render_safe` *Render whiskers safely*

Description

Render whiskers safely

Usage

```
render_safe(x)
```

Arguments

`x`	(`character`) input to be rendered safely.

Value

`character` with substituted placeholders.

Note

The strings enclosed in `{}` are substituted using the key-values pairs set with `add_whiskers`.

Examples

```
render_safe("Name of {Patient_label}")
```

`rule` *Create rule based on mappings*

Description

Create rule based on mappings

Usage

```
rule(
  ...,
  .lst = list(...),
  .string_as_fct = TRUE,
  .na_last = TRUE,
  .drop = FALSE,
  .to_NA = ""
)
```

Arguments

`...`	Mapping pairs, the argument name is the transformed while its values are original values.
`.lst`	(`list`) of mapping.
`.string_as_fct`	(`flag`) whether to convert characters to factors.
`.na_last`	(`flag`) whether the level replacing `NA` should be last.
`.drop`	(`flag`) whether to drop empty levels.
`.to_NA`	(`character`) values that should be converted to `NA`. Set to `NULL` if nothing should be converted to `NA`.

Value

a `rule` object.

Note

Conversion to `NA` is the last step of the remapping process.

Examples

```
rule("X" = "x", "Y" = c("y", "z"))
rule("X" = "x", "Y" = c("y", "z"), .drop = TRUE, .to_NA = c("a", "b"), .na_last = FALSE)
```

`show_whisker` *Show Whisker Values*

Description

Show Whisker Values

Usage

```
show_whisker()
```

Value

invisible `NULL`. Prints the values stored in the whisker environment.

Examples

```
show_whisker()
```

Index