Package: recodeflow (via r-universe)

October 21, 2024

Type Package

Title Contains functions to interface with variable details sheets, including recoding variables and converting them to PMML

Version 0.1.0

Maintainer Rostyslav Vyuha <rvyuha@toh.ca>

Description Recode and harmonize data using variable and details sheets.

Depends R (>= 3.1.0)

Imports sjlabelled, stringr, tidyr, haven, dplyr, magrittr, glue, survival

License MIT + file LICENSE

URL https://github.com/Big-Life-Lab/recodeflow

BugReports https://github.com/Big-Life-Lab/recodeflow/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests DT, kableExtra, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://big-life-lab.r-universe.dev

RemoteUrl https://github.com/big-life-lab/recodeflow

RemoteRef HEAD

RemoteSha 404576eebe3f5f5f605c8617c22cc66e47138c5d

Contents

`compare_value_based_on_interval`
Compare Value Based On Interval

Description

Compare values on the scientific notation interval

Usage

```
compare_value_based_on_interval(
  left_boundary,
  right_boundary,
  data,
  compare_columns,
  interval
)
```

Arguments

`left_boundary`	the min value
`right_boundary`	the max value
`data`	the data that contains values being compared
`compare_columns`	The columns inside data being checked
`interval`	The scientific notation interval

Value

a boolean vector containing true for rows where the comparison is true

`create_id_row` *ID role creation*

Description

Creates ID row for rec_with_table

Usage

```
create_id_row(data, id_role_name, database_name, variables)
```

Arguments

`data`	the data that the ID role row is created for
`id_role_name`	name for the role that ID is created from
`database_name`	the name of the database
`variables`	variables sheet containing variable information

Value

data with the ID row attached

`create_label_list_element` *Create label list element*

Description

A data labeling utility function for creating individual variable labels

Usage

```
create_label_list_element(variable_rows)
```

Arguments

`variable_rows`	all variable details rows containing 1 variable information

Value

a list containing labels for the passed variable

`example_der_fun` *example_der_fun caluclates chol*bili*

Description

example_der_fun caluclates chol*bili

Usage

```
example_der_fun(chol, bili)
```

Arguments

`chol`	the row value for chol
`bili`	the row value for bili

`format_recoded_value` *Recode NA formatting*

Description

Recodes the NA depending on the var type

Usage

```
format_recoded_value(cell_value, var_type)
```

Arguments

`cell_value`	The value inside the recTo column
`var_type`	the toType of a variable

Value

an appropriately coded tagged NA

get_data_variable_name *Get Data Variable Name*

Description

Retrieves the name of the column inside data to use for calculations

Usage

```
get_data_variable_name(
  data_name,
  data,
  row_being_checked,
  variable_being_checked
)
```

Arguments

data_name	name of the database being checked
data	database being checked
row_being_checked	the row from variable details that contains information on this variable
variable_being_checked	the name of the recoded variable

Value

the data equivalent of variable_being_checked

get_table_name *Returns the name of the table for a table start variable*

Description

Returns the name of the table for a table start variable

Usage

```
get_table_name(table_feeder_var)
```

Arguments

table_feeder_var	string The table variable start

Value

string

`is_derived_var` *Whether a variable in a variables details sheet is a derived variable*

Description

Whether a variable in a variables details sheet is a derived variable

Usage

```
is_derived_var(variable_details_row)
```

Arguments

`variable_details_row`	A data frame with a single row which will be checked

Value

A boolean

`is_equal` *Checks whether two values are equal including NA*

Description

Compared to the base "==" operator in R, this function returns true if the two values are NA whereas the base "==" operator returns NA

Usage

```
is_equal(v1, v2)
```

Arguments

`v1`	variable 1
`v2`	variable 2

Value

boolean value of whether or not v1 and v2 are equal

Examples

```
is_equal(1,2)
# FALSE

is_equal(1,1)
# TRUE

1==NA
# NA

is_equal(1,NA)
# FALSE

NA==NA
# NA

is_equal(NA,NA)
# TRUE
```

`is_table_feeder_var` *Checks whether a variable start is a table*

Description

Checks whether a variable start is a table

Usage

```
is_table_feeder_var(feeder_var)
```

Arguments

`feeder_var`	string The variable start to check

Value

bool

`label_data` *label_data*

Description

Attaches labels to the data_to_label to preserve metadata

Usage

```
label_data(label_list, data_to_label)
```

Arguments

`label_list`	the label list object that contains extracted labels from variable details
`data_to_label`	The data that is to be labeled

Value

Returns labeled data

`recode_columns` *recode_columns*

Description

Recodes columns from passed row and returns just table with those columns and same rows as the data

Usage

```
recode_columns(
  data,
  variables_details_rows_to_process,
  data_name,
  log,
  print_note,
  else_default,
  tables
)
```

Arguments

data	The source database
variables_details_rows_to_process	rows from variable details that are applicable to this DB
data_name	Name of the database being passed
log	The option of printing log
print_note	the option of printing the note columns
else_default	default else value to use if no else is present
tables	A list of reference tables

Value

Returns recoded and labeled data

---

rec_with_table *Recode with Table*

Description

Creates new variables by recoding variables in a dataset using the rules specified in a variables details sheet

Usage

```
rec_with_table(
  data,
  variables = NULL,
  database_name = NULL,
  variable_details = NULL,
  else_value = NA,
  append_to_data = FALSE,
  log = FALSE,
  notes = TRUE,
  var_labels = NULL,
  custom_function_path = NULL,
  attach_data_name = FALSE,
  id_role_name = NULL,
  name_of_environment_to_load = NULL,
  append_non_db_columns = FALSE,
  tables = list()
)
```

Arguments

`data`	A dataframe containing the variables to be recoded. Can also be a named list of dataframes.
`variables`	Character vector containing the names of the new variables to recode to or a dataframe containing a variables sheet.
`database_name`	A String containing the name of the database containing the original variables which should match up with a database from the databaseStart column in the variables details sheet. Should be a character vector if data is a named list where each vector item matches a name in the data list and also matches with a value in the databaseStart column of a variable details sheet.
`variable_details`	A dataframe containing the specifications for recoding.
`else_value`	Value (string, number, integer, logical or NA) that is used to replace any values that are outside the specified ranges (no rules for recoding).
`append_to_data`	Logical, if `TRUE` (default), the newly created variables will be appended to the original dataset.
`log`	Logical, if `FALSE` (default), a log containing information about the recoding will not be printed.
`notes`	Logical, if `FALSE` (default), will not print the content inside the 'Note" column of the variable being recoded.
`var_labels`	labels vector to attach to variables in variables
`custom_function_path`	string containing the path to the file containing functions to run for derived variables. This file will be sourced and its functions loaded into the R environment.
`attach_data_name`	logical to attach name of database to end table
`id_role_name`	name for the role to be used to generate id column
`name_of_environment_to_load`	Name of package to load variables and variable_details from
`append_non_db_columns`	boolean determening if data not present in this cycle should be appended as NA
`tables`	named list of data.frame A list of reference tables that can be passed as parameters into the function for a derived variable

Details

The variable_details dataframe needs the following columns:

variable Name of the new variable created. The name of the new variable can be the same as the original variable if it does not change the original variable definition

toType type the new variable *cat = categorical, cont = continuous*

databaseStart Names of the databases that the original variable can come from. Each database name should be seperated by a comma. For eg., "cchs2001_p, cchs2003_p,cchs2005_p,cchs2007_p"

variableStart Names of the original variables within each database specified in the databaseStart column. For eg. , "cchs2001_p::RACA_6A,cchs2003_p::RACC_6A,ADL_01". The final variable specified is the name of the variable for all other databases specified in databaseStart but not in this column. For eg., ADL_01 would be the original variable name in the cchs2005_p and cchs2007_p databases.

fromType variable type of start variable. *cat = categorical or factor variable cont = continuous variable (real number or integer)*

recTo Value to recode to

recFrom Value/range being recoded from

Each row in the *variables details* sheet encodes the rule for recoding value(s) of the original variable to a category in the new variable. The categories of the new variable are encoded in the *recTo* column and the value(s) of the original variable that recode to this new value are encoded in the *recFrom* column. These recode columns follow a syntax similar to the *sjmisc::rec()* function. Whereas in the *sjmisc::rec()* function the recoding rules are in one string, in the variables details sheet they are encoded over multiple rows and columns (recFrom an recTo). For eg., a recoding rule in the sjmisc function would like like "1=2;2=3" whereas in the variables details sheet this would be encoded over two rows with recFrom and recTo values of the first row being 1 and 2 and similarly for the second row it would be 2 and 3. The rules for describing recoding pairs are shown below:

recode pairs Each recode pair is a row

multiple values Multiple values from the old variable that should be recoded into a new category of the new variable should be separated with a comma. e.g., *recFrom = "1,2"; recTo = 1*will recode values of 1 and 2 in the original variable to 1 in the new variable

value range A value range is indicated by a colon, e.g. *recFrom= "1:4"; recTo = 1* will recode all values from 1 to 4 into 1

min* and *max minimum and maximum values are indicated by *min* (or *lo*) and *max* (or *hi*), e.g. *recFrom = "min:4"; recTo = 1* will recode all values from the minimum value of the original variable to 4 into 1

"else" All other values, which have not been specified yet, are indicated by *else*, e.g. *recFrom = "else"; recTo = NA* will recode all other values (not specified in other rows) of the original variable to "NA")

"copy" the *else* token can be combined with *copy*, indicating that all remaining, not yet recoded values should stay the same (are copied from the original value), e.g. *recFrom = "else"; recTo = "copy"*

***NA*'s** *NA* values are allowed both for the original and the new variable, e.g. *recFrom "NA"; recTo = 1. or "recFrom = "3:5"; recTo = "NA"* (recodes all NA into 1, and all values from 3 to 5 into NA in the new variable)

Value

a dataframe that is recoded according to rules in variable_details.

Examples

```
var_details <-
  data.frame(
```

```
    "variable" = c("time", rep("status", times = 3), rep("trt", times = 2), "age", rep("sex", times = 2), rep("ascite
    "dummyVariable" = c("NA", "status0", "status1","status2", "trt1","trt2","NA","sexM","sexF", "ascites0", "ascit
    "typeEnd" = c("cont", rep("cat", times = 3), rep("cat", times = 2), "cont", rep("cat", times = 2), rep("cat", tim
     "databaseStart" = rep("tester1, tester2", times = 31),
    "variableStart" = c("[time]", rep("[status]", times = 3), rep("[trt]", times = 2), "[age]", rep("[sex]", times =
    "typeStart" = c("cont", rep("cat", times = 3), rep("cat", times = 2), "cont", rep("cat", times = 2), rep("cat", t
    "recEnd" = c("copy", "0", "1","2", "1","2","copy","m","f", "0", "1","0","1","0","1","0.0","0.5","1.0",rep("cop
    "catLabel" = c("", "status 0", "status 1","status 2", "trt 1","trt 2","","sex m","sex f", "ascites 0", "ascites 1
    "catLabelLong" = c("", "status 0", "status 1","status 2", "trt 1","trt 2","","sex m","sex f", "ascites 0", "ascit
    "recStart" = c("else", "0", "1","2", "1","2","else","m","f", "0", "1","0","1","0","1","0.0","0.5","1.0",rep("e
    "catStartLabel" = c("", "status 0", "status 1","status 2", "trt 1","trt 2","","sex m","sex f", "ascites 0", "asci
    "variableStartShortLabel" = c("time", rep("status", times = 3), rep("trt", times = 2), "age", rep("sex", times =
    "variableStartLabel" = c("time", rep("status", times = 3), rep("trt", times = 2), "age", rep("sex", times = 2), r
     "units" = rep("NA", times = 31),
     "notes" = rep("This is sample survival pbc data", times = 31)
  )
var_sheet <-
  data.frame(
   "variable" = c("time","status","trt", "age","sex","ascites","hepato", "spiders", "edema", "bili", "chol", "alb
   "label" = c("time","status","trt", "age","sex","ascites","hepato", "spiders", "edema", "bili", "chol", "albumi
   "labelLong" = c("time","status","trt", "age","sex","ascites","hepato", "spiders", "edema", "bili", "chol", "al
    "section" = rep("tester", times=19),
    "subject" = rep("tester",times = 19),
   "variableType" = c("cont", "cat", "cat", "cont","cat", "cat", "cat","cat", "cat", rep("cont", times = 9), "cat")
    "databaseStart" = rep("tester1, tester2", times = 19),
    "units" = rep("NA", times = 19),
   "variableStart" = c("[time]","[status]", "[trt]", "[age]", "[sex]", "[ascites]","[hepato]","[spiders]","[edema
  )
library(survival)
tester1 <- survival::pbc[1:209,]
tester2 <- survival::pbc[210:418,]
db_name1 <- "tester1"
db_name2 <- "tester2"

rec_sample1 <- rec_with_table(data = tester1,
variables = var_sheet,
variable_details = var_details,
database_name = db_name1)

rec_sample2 <- rec_with_table(data = tester2,
variables = var_sheet,
variable_details = var_details,
database_name = db_name2)
```

`select_vars_by_role` *Vars selected by role*

Description

Selects variables from variables sheet based on passed roles

Usage

```
select_vars_by_role(roles, variables)
```

Arguments

roles	a vector containing a single or multiple roles to match by
variables	the variables sheet containing variable info

Value

a vector containing the variable names that match the passed roles

---

set_data_labels *Set Data Labels*

Description

sets labels for passed database, Uses the names of final variables in variable_details/variables_sheet as well as the labels contained in the passed dataframes

Usage

```
set_data_labels(data_to_label, variable_details, variables_sheet = NULL)
```

Arguments

data_to_label	newly transformed dataset
variable_details	variable_details.csv
variables_sheet	variables.csv

Value

labeled data_to_label

Index