Package ‘recodeflow’

June 9, 2021

Type Package
Title Interface Functions for PMML Creation, and Data Recoding
Version 0.1.0
Maintainer Rostyslav Vyuha <rvyuha@toh.ca>
Description Contains functions to interface with variables and variable details sheets, including recoding variables and converting them to PMML.
Depends R (>= 3.1.0)
Imports XML (>= 3.98-1.11), sjlabelled, stringr, tidyr, haven, dplyr, magrittr
License MIT + file LICENSE
URL https://github.com/Big-Life-Lab/recodeflow
BugReports https://github.com/Big-Life-Lab/recodeflow/issues
Encoding UTF-8
RoxygenNote 7.1.1
Suggests testthat (>= 2.1.0), survival
NeedsCompilation no
Author Yulric Sequeira [aut], Luke Bailey [aut], Rostyslav Vyuha [aut, cre], The Ottawa Hospital [cph], Doug Manuel [cph] (<https://orcid.org/0000-0003-0912-0845>)
Repository CRAN
Date/Publication 2021-06-09 07:00:02 UTC

R topics documented:

  add_data_field_children_for_start_var .................................. 2
  attach_apply_nodes ........................................................ 3
  attach_cat_value_nodes_for_start_var .................................... 4
  attach_cont_value_nodes_for_start_var .................................. 4
add_data_field_children_for_start_var

Add DataField child nodes for start variable.

Description

Add DataField child nodes for start variable.
attach_apply_nodes

Usage

add_data_field_children_for_start_var(data_field, var_details_rows)

Arguments

data_field: DataField node to attach child nodes.
var_details_rows: Variable details rows associated with current variable.

Value

Updated DataField node.

attach_apply_nodes

Attach Apply nodes to a parent node.

Description

Attach Apply nodes to a parent node.

Usage

attach_apply_nodes(var_details_rows, parent_node, db_name)

Arguments

var_details_rows: Variable details rows associated with a variable.
parent_node: An XML node.
db_name: Database name.

Value

Updated parent node.
attach_cat_value_nodes_for_start_var

Attach categorical value nodes to DataField node for start variable.

Description
Attach categorical value nodes to DataField node for start variable.

Usage
attach_cat_value_nodes_for_start_var(var_details_row, data_field)

Arguments
- var_details_row: Variable details sheet row.
- data_field: DataField node to attach Value nodes.

Value
Updated DataField node.

attach_cont_value_nodes_for_start_var

Attach continuous Value nodes for start variable.

Description
Attach continuous Value nodes for start variable.

Usage
attach_cont_value_nodes_for_start_var(var_details_row, data_field)

Arguments
- var_details_row: Variable details sheet row.
- data_field: DataField node to attach Value nodes.

Value
Updated DataField node.
attach_derived_field_child_nodes

Attach child nodes to DerivedField node.

Description

Attach child nodes to DerivedField node.

Usage

attach_derived_field_child_nodes(
    derived_field_node,
    var_details_sheet,
    var_name,
    db_name
)

Arguments

- derived_field_node: DerivedField node to attach child nodes.
- var_details_sheet: Variable details sheet data frame.
- var_name: Variable name.
- db_name: Database name.

Value

Updated DerivedField node.

attach_range_value_nodes

Attach Value nodes to DataField node. Used when 'recFrom' has a value range.

Description

Attach Value nodes to DataField node. Used when 'recFrom' has a value range.

Usage

attach_range_value_nodes(var_details_row, data_field)
Arguments

var_details_row    Variable details sheet row.
data_field    DataField node to attach Value nodes.

Value

Updated DataField node.

---

**build_data_field_for_start_var**

*Build DataField node for start variable.*

---

Description

Build DataField node for start variable.

Usage

`build_data_field_for_start_var(var_name, var_details_rows)`

Arguments

var_name    Variable name.
var_details_rows    All variable details rows for the 'var_name' variable.

Value

DataField node with optype and dataType according to ‘fromType’.

---

**build_data_field_for_var**

*Build DataField node for variable.*

---

Description

Build DataField node for variable.

Usage

`build_data_field_for_var(var_name, vars_sheet)`
**Arguments**

- **var_name**  
  Variable name.

- **vars_sheet**  
  Variable sheet data frame.

**Value**

DataField node for variable.

---

**build_derived_field_node**

*Build DerivedField node.*

---

**Description**

Build DerivedField node.

**Usage**

```python
build_derived_field_node(vars_sheet, var_details_sheet, var_name, db_name)
```

**Arguments**

- **vars_sheet**  
  Variables sheet data frame.

- **var_details_sheet**  
  Variable details sheet data frame.

- **var_name**  
  Variable name.

- **db_name**  
  Database name.

**Value**

DerivedField node.

---

**build_derived_field_value_node**

*Build Value node for DerivedField node.*

---

**Description**

Build Value node for DerivedField node.

**Usage**

```python
build_derived_field_value_node(var_details_row)
```
**build_numeric_derived_field_apply_node**

*Arguments*

var_details_row

Variable details sheet row.

*Value*

Value node.

---

**build_missing_const_node**

*Build Constant node for a missing value for a variable.*

*Description*

Build Constant node for a missing value for a variable.

*Usage*

build_missing_const_node(var_details_row)

*Arguments*

var_details_row

Variable details sheet row.

*Value*

Constant node.

---

**build_numeric_derived_field_apply_node**

*Build Apply node with singleton numeric for DerivedField node.*

*Description*

Build Apply node with singleton numeric for DerivedField node.

*Usage*

build_numeric_derived_field_apply_node(var_details_row, db_name)

*Arguments*

var_details_row

Variable details sheet row.

db_name

Database name.
**build_ranged_derived_field_apply_node**

Value

Apply node for DerivedField node.

---

**build_ranged_derived_field_apply_node**

*Build Apply node with interval nodes for DerivedField node.*

---

**Description**

Build Apply node with interval nodes for DerivedField node.

**Usage**

`build_ranged_derived_field_apply_node(var_details_row, db_name)`

**Arguments**

- `var_details_row` - Variable details sheet row.
- `db_name` - Database name.

**Value**

Apply node with intervals for DerivedField node.

---

**build_trans_dict**

*Build a TransformationDictionary node.*

---

**Description**

Build a TransformationDictionary node.

**Usage**

`build_trans_dict(vars_sheet, var_details_sheet, var_names, db_name)`

**Arguments**

- `vars_sheet` - Variable sheet data frame.
- `var_details_sheet` - Variable details sheet data frame.
- `var_names` - Vector of variable names.
- `db_name` - Database name.

**Value**

TransformationDictionary node.
**build_variable_field_ref_node**

*Build FieldRef node for variable.*

**Description**

Build FieldRef node for variable.

**Usage**

```python
build_variable_field_ref_node(var_details_row, db_name)
```

**Arguments**

- `var_details_row`: Variable details sheet row.
- `db_name`: Database name.

**Value**

FieldRef node.

---

**compare_value_based_on_interval**

*Compare Value Based On Interval*

**Description**

Compare values on the scientific notation interval.

**Usage**

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

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**

```python
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` calulates chol*bili

**Description**

`example_der_fun` calulates chol*bili

**Usage**

```python
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
get_margin_closure

Value
  the data equivalent of variable_being_checked

get_margins

Description
  Extract margins from character vector.

Usage
  get_margins(chars)

Arguments
  chars  Character vector.

Value
  Margins as character vector.

get_margin_closure

Description
  Get closure type for a margin.

Usage
  get_margin_closure(chars)

Arguments
  chars  Character vector.

Value
  Closure type.
get_start_var_name

Get variable name from variableStart using database name.

Description
Get variable name from variableStart using database name.

Usage
get_start_var_name(var_details_row, db_name)

Arguments
- var_details_row: A variable details row.
- db_name: Name of database to extract from.

Value
character: The name of the start variable.

get_variable_type_data_type

Get data type for variable type.

Description
Get data type for variable type.

Usage
get_variable_type_data_type(var_details_rows, var_type, is_start_var)

Arguments
- var_details_rows: All variable details rows for the variable.
- var_type: Variable type
- is_start_var: boolean if the passed variable is variable start

Value
'var_type' data type.
get_var_details_rows  Get all variable details rows for a variable and database combination.

Description
Get all variable details rows for a variable and database combination.

Usage
get_var_details_rows(var_details_sheet, var_name, db_name)

Arguments
- var_details_sheet: A data frame representing a variable details sheet.
- var_name: Variable name.
- db_name: Database name.

Value
All variable details rows for the variable and database combination.

get_var_details_row_indices  Get all variable details row indices for a variable.

Description
Get all variable details row indices for a variable.

Usage
get_var_details_row_indices(var_details_sheet, var_name)

Arguments
- var_details_sheet: A data frame representing a variable details sheet.
- var_name: Variable name.

Value
All variable details row indices for a variable.
**get_var_sheet_row**

Get variable row from variable sheet.

**Description**

Get variable row from variable sheet.

**Usage**

`get_var_sheet_row(var_name, vars_sheet)`

**Arguments**

- `var_name`: Variable name.
- `vars_sheet`: Variable sheet data frame.

**Value**

Variable row.

---

**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

```r
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_left_open**

*Extract margins from character vector.*

**Description**

Extract margins from character vector.

**Usage**

```r
is_left_open(chars)
```

**Arguments**

- `chars` Character vector.

**Value**

Whether the left endpoint of an interval is open.
is_numeric

Check if a character object can be converted to a number.

Description
Check if a character object can be converted to a number.

Usage

is_numeric(chars)

Arguments
chars Character object.

Value
Whether 'chars' can be converted to a numeric value.

is_rec_from_range

Check if recFrom is a range for a variable details row.

Description
Check if recFrom is a range for a variable details row.

Usage

is_rec_from_range(var_details_row)

Arguments
var_details_row Variable details sheet row.

Value
Whether recFrom is a range.
**is_right_open**  
*Extract margins from character vector.*

**Description**

Extract margins from character vector.

**Usage**

```r
is_right_open(chars)
```

**Arguments**

- `chars`: Character vector.

**Value**

Whether the right endpoint of an interval is open.

---

**label_data**  
*label_data*

**Description**

Attaches labels to the `data_to_label` to preserve metadata.

**Usage**

```r
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

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
)

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

Value
Returns recoded and labeled data

recode_to_pmml

Description
Creates a PMML document from variable and variable details sheets for specified database.

Usage
recode_to_pmml(var_details_sheet, vars_sheet, db_name, vars_to_convert = NULL)
Arguments

var_details_sheet
A data frame representing a variable details sheet.

vars_sheet
A data frame representing a variables sheet.

db_name
A string containing the name of the database that holds the start variables. Should match up with one of the databases in the databaseStart column.

vars_to_convert
A vector of strings containing the names of variables from the variable column in the variable details sheet that should be converted to PMML. Passing in an empty vector will convert all the variables.

Value

A PMML document.

Examples

var_details_sheet <-
data.frame(
  "variable" = rep(c("A", "B", "C"), each = 3),
  "dummyVariable" = c("AY", "AN", "ANA", "BY", "BN", "BNA", "CY", "CN", "CNA"),
  "toType" = rep("cat", times = 9),
  "databaseStart" = rep("tester", times = 9),
  "variableStart" = rep(c("tester::startA", "tester::startB", "tester::startC"), each = 3),
  "fromType" = rep("cat", times = 9),
  "recTo" = rep(c("1", "2", "NA::a"), times = 3),
  "numValidCat" = rep("2", times = 9),
  "catLabel" = rep(c("Yes", "No", "Not answered"), times = 3),
  "catLabelLong" = rep(c("Yes", "No", "Not answered"), times = 3),
  "recFrom" = rep(c("1", "2", "9"), times = 3),
  "catStartLabel" = rep(c("Yes", "No", "Not answered"), times = 3),
  "variableStartShortLabel" = rep(c("Group A", "Group B", "Group C"), each = 3),
  "variableStartLabel" = rep(c("Group A", "Group B", "Group C"), each = 3),
  "units" = rep("NA", times = 9),
  "notes" = rep("This is not real data", times = 9)
)

vars_sheet <-
data.frame(
  "variable" = c("A", "B", "C"),
  "label" = c("Group A", "Group B", "Group C"),
  "labelLong" = c("Group A", "Group B", "Group C"),
  "section" = rep("tester", times=3),
  "subject" = rep("tester", times = 3),
  "variableType" = rep("Categorical", times=3),
  )
rec_with_table

```
"databaseStart" = rep("tester", times = 3),
"units" = rep("NA", times = 3),
"variableStart" = c("tester::startA", "tester::startB", "tester::startC")
)
db_name <- "tester"
vars <- c("A", "B", "C")

actual_pmml <- recode_to_pmml(
  var_details_sheet,
  vars_sheet,
  db_name,
  vars
)
```

---

**Description**

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

**Usage**

```r
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
)
```

**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 determining if data not present in this cycle should be appended as NA

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 separated 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 and 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 1to4 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("ascites", times = 2),
  rep("hepato", times = 2), rep("spiders", times = 2),
  rep("edema", times = 3),
  "bili", "chol", "albumin", "copper", "alk.phos", "ast",
  "trig", "platelet", "protime", rep("stage", times = 4)),
  "dummyVariable" = c("NA", "status0", "status1", "status2", "trt1", "trt2",
  "NA", "sexM", "sexF", "ascites0", "ascites1", "hepato0", "hepato1",
  "spiders0", "spiders1", "edema0.0", "edema0.5", "edema1.0",
  rep("NA", times = 9), "stage1", "stage2", "stage3", "stage4"),
  "toType" = c("cont", rep("cat", times = 3), rep("cat", times = 2),
  "cont", rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2),
  rep("cat", times = 2), rep("cat", times = 2)
)

rec_with_table

```r
var_sheet <-
  data.frame(
    units = rep("NA", times = 31),
    notes = rep("This is sample survival pbc data", times = 31)
  )
```
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

add_data_field_children_for_start_var, 2
attach_apply_nodes, 3
attach_cont_value_nodes_for_start_var, 4
attach_cat_value_nodes_for_start_var, 4
attach_derived_field_child_nodes, 5
attach_range_value_nodes, 5
build_data_field_for_start_var, 6
build_data_field_for_var, 6
build_derived_field_node, 7
build_derived_field_value_node, 7
build_missing_const_node, 8
build_numeric Derived_field_apply_node, 8
build_ranged_derived_field_apply_node, 9
build_trans_dict, 9
build_variable_field_ref_node, 10
compare_value_based_on_interval, 10
create_id_row, 11
create_label_list_element, 12
example_der_fun, 12
format_recoded_value, 13
get_data_variable_name, 13
get_margin_closure, 14
get_margins, 14
get_start_var_name, 15
get_var_details_row_indices, 16
get_var_details_rows, 16
get_var_sheet_row, 17
get_variable_type_data_type, 15
is_equal, 17
is_left_open, 18
is_numeric, 19
is_rec_from_range, 19
is_right_open, 20
label_data, 20
rec_with_table, 23
recode_columns, 21
recode_to_pmml, 21
select_vars_by_role, 27
set_data_labels, 28