Package ‘metatools’

March 13, 2023

Type Package
Title Enable the Use of ‘metacore’ to Help Create and Check Dataset
Version 0.1.5
Description Uses the metadata information stored in ‘metacore’ objects to check and build metadata associated columns.
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.3
Imports dplyr, metacore (>= 0.0.4), purrr, rlang, stringr, tidyr, tibble, magrittr
Suggests testthat (>= 3.0.0), haven, covr, safetyData, admiral.test
Config/testthat/edition 3
URL https://pharmaverse.github.io/metatools/
BugReports https://github.com/pharmaverse/metatools/issues
NeedsCompilation no
Author Christina Fillmore [aut, cre] (<https://orcid.org/0000-0003-0595-2302>), Mike Stackhouse [aut] (<https://orcid.org/0000-0001-6030-723X>), GlaxoSmithKline LLC [cph, fnl], F. Hoffmann-La Roche AG [cph, fnl], Atorus Research LLC [cph, fnl]
Maintainer Christina Fillmore <christina.e.fillmore@gsk.com>
Repository CRAN
Date/Publication 2023-03-13 10:00:05 UTC

R topics documented:

  add_labels ..................................................... 2
  add_variables .................................................. 3
  build_from Derived .......................................... 3
  build_qnam .................................................... 4
add_labels

Apply labels to multiple variables on a data frame

Description

This function allows a user to apply several labels to a dataframe at once.

Usage

add_labels(data, ...)  

Arguments

- data: A data.frame or tibble
- ...: Named parameters in the form of variable = 'label'

Value

data with variable labels applied

Examples

```r
add_labels(  
  mtcars,  
  mpg = "Miles Per Gallon",  
  cyl = "Cylinders"  
)
```
add_variables

Add Missing Variables

Description

This function adds in missing columns according to the type set in the metacore object. All values in the new columns will be missing, but typed correctly. If unable to recognize the type in the metacore object will return a logical type.

Usage

add_variables(dataset, metacore, dataset_name = NULL)

Arguments

- **dataset**: Dataset to add columns to. If all variables are present no columns will be added.
- **metacore**: metacore object that only contains the specifications for the dataset of interest.
- **dataset_name**: Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetted.

Value

The given dataset with any additional columns added

Examples

library(metacore)
library(haven)
library(dplyr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt")) %>%
  select(-TRTSDT, -TRT01P, -TRT01PN)
add_variables(data, spec)

build_from_derived

Build a dataset from derived

Description

This function builds a dataset out of the columns that just need to be pulled through. So any variable that has a derivation in the format of 'dataset.variable' will be pulled through to create the new dataset. When there are multiple datasets present, they will be joined by the shared 'key_seq' variables. These columns are often called 'Predecessors' in ADaM, but this is not universal so that is optional to specify.
Usage

\[
\text{build_from_derived(}
\begin{align*}
\text{metacore,} & \\
\text{ds_list,} & \\
\text{dataset_name = NULL,} & \\
\text{predecessor_only = TRUE,} & \\
\text{keep = FALSE} \\
\end{align*}
\]

Arguments

- **metacore**: metacore object that contains the specifications for the dataset of interest.
- **ds_list**: Named list of datasets that are needed to build the from.
- **dataset_name**: Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn’t already been subsetted.
- **predecessor_only**: By default ‘FALSE’, but if ‘TRUE’ will only use derivations with the origin of ‘Predecessor’.
- **keep**: Boolean to determine if the original columns should be kept. By default ‘FALSE’, so only the ADaM columns are kept. If ‘TRUE’ the resulting dataset will have all the ADaM columns as well as any SDTM column that were renamed in the ADaM (i.e ‘ARM’ and ‘TRT01P’ will be in the resulting dataset).

Value

- **dataset**

Examples

```r
library(metacore)
library(haven)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
ds_list <- list(DM = read_xpt(metatools_example("dm.xpt")))
build_from_derived(spec, ds_list, predecessor_only = FALSE)
```

---

**build_qnam**

*Build the observations for a single QNAM*

### Description

Build the observations for a single QNAM

### Usage

\[
\text{build_qnam(}\begin{align*}
dataset, & \\
\text{qnam,} & \\
\text{qlabel,} & \\
\text{idvar,} & \\
\text{qeval,} & \\
\text{qorig} & \end{align*}\]

```r
```
**check_ct_col**

**Arguments**

- **dataset**: Input dataset
- **qnam**: QNAM value
- **qlabel**: QLABEL value
- **idvar**: IDVAR variable name (provided as a string)
- **qeval**: QEVAL value to be populated for this QNAM
- **qorig**: QORIG value to be populated for this QNAM

**Value**

Observations structured in SUPP format

---

**check_ct_col**

*Check Control Terminology for a Single Column*

**Description**

This function checks the column in the dataset only contains the control terminology as defined by the metacore specification

**Usage**

`check_ct_col(data, metacore, var, na_acceptable = NULL)`

**Arguments**

- **data**: Data to check
- **metacore**: A metacore object to get the codelist from. If the variable has different codelists for different datasets the metacore object will need to be subsetted using `select_dataset` from the metacore package.
- **var**: Name of variable to check
- **na_acceptable**: Logical value, set to `NULL` by default, so the acceptability of missing values is based on if the core for the variable is "Required" in the `metacore` object. If set to `TRUE` then will pass check if values are in the control terminology or are missing. If set to `FALSE` then NA will not be acceptable.

**Value**

Given data if column only contains control terms. If not, will error given the values which should not be in the column
Examples

```r
library(metacore)
library(haven)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt"))
check_ct_col(data, spec, "TRT01PN")
check_ct_col(data, spec, "TRT01PN")
```

---

**check_ct_data**  
*Check Control Terminology for a Dataset*

**Description**

This function checks that all columns in the dataset only contains the control terminology as defined by the metacore specification.

**Usage**

```r
check_ct_data(data, metacore, na_acceptable = NULL)
```

**Arguments**

- `data`: Dataset to check
- `metacore`: metacore object that contains the specifications for the dataset of interest. If any variable has different codelists for different datasets the metacore object will need to be subsetted using 'select_dataset' from the metacore package.
- `na_acceptable`: Logical value, set to ‘NULL’ by default, so the acceptability of missing values is based on if the core for the variable is "Required" in the 'metacore' object. If set to 'TRUE' then will pass check if values are in the control terminology or are missing. If set to ‘FALSE’ then NA will not be acceptable.

**Value**

Given data if all columns pass. It will error otherwise.

**Examples**

```r
library(haven)
library(metacore)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt"))
check_ct_data(data, spec)
```
check_variables  

Check Variable Names

Description

This function checks the variables in the dataset against the variables defined in the metacore specifications. If everything matches the function will return ‘TRUE’ and a message starting everything is as expected. If there are additional or missing variables and error will explain the discrepancies

Usage

check_variables(data, metacore, dataset_name = NULL)

Arguments

data  
Dataset to check  

metacore  
metacore object that only contains the specifications for the dataset of interest.

dataset_name  
Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetted.

Value

message if the dataset matches the specification and the dataset, and error otherwise

Examples

library(haven)
library(metacore)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt"))
check_variables(data, spec)

combine_supp  

Combine the Domain and Supplemental Qualifier

Description

Combine the Domain and Supplemental Qualifier

Usage

combine_supp(dataset, supp)
convert_var_to_fct

Arguments

- **dataset**: Domain dataset
- **supp**: Supplemental Qualifier dataset

Value

- A dataset with the supp variables added to it

Examples

```r
library(safetyData)
library(tibble)
combine_supp(sdtm_ae, sdtm_suppae) %>% as_tibble()
```

---

### convert_var_to_fct

**Convert Variable to Factor with Levels Set by Control Terms**

Description

This function takes a dataset, a metacore object and a variable name. Then looks at the metacore object for the control terms for the given variable and uses that to convert the variable to a factor with those levels. If the control terminology is a code list, the code column will be used. The function fails if the control terminology is an external library.

Usage

```r
convert_var_to_fct(data, metacore, var)
```

Arguments

- **data**: A dataset containing the variable to be modified
- **metacore**: A metacore object to get the codelist from. If the variable has different codelists for different datasets the metacore object will need to be subselected using `select_dataset` from the metacore package
- **var**: Name of variable to change

Value

- Dataset with variable changed to a factor
create_cat_var

Examples

```r
library(metacore)
library(haven)
library(dplyr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
dm <- read_xpt(metatools_example("dm.xpt")) %>%
    select(USUBJID, SEX, ARM)
# Variable with codelist control terms
convert_var_to_fct(dm, spec, SEX)
# Variable with permitted value control terms
convert_var_to_fct(dm, spec, ARM)
```

---

create_cat_var  
*Create Categorical Variable from Codelist*

Description

Using the grouping from either the ‘decode_var’ or ‘code_var’ and a reference variable (‘ref_var’) it will create a categorical variable and the numeric version of that categorical variable.

Usage

```r
create_cat_var(data, metacore, ref_var, grp_var, num_grp_var = NULL)
```

Arguments

- `data`  
  Dataset with reference variable in it
- `metacore`  
  A metacore object to get the codelist from. If the variable has different codelists for different datasets the metacore object will need to be subsetted using `select_dataset` from the metacore package.
- `ref_var`  
  Name of variable to be used as the reference i.e AGE when creating AGEGR1
- `grp_var`  
  Name of the new grouped variable
- `num_grp_var`  
  Name of the new numeric decode for the grouped variable. This is optional if no value given no variable will be created

Value

- `dataset` with new column added

Examples

```r
library(metacore)
library(haven)
library(dplyr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
```
create_subgrps

Create Subgroups

Description

Create Subgroups

Usage

create_subgrps(ref_vec, grp_defs)

Arguments

ref_vec Vector of numeric values

grp_defs Vector of strings with groupings defined. Format must be either: <00, >=00, 00-00, or 00<00

Value

Character vector of the values in the subgroups

Examples

create_subgrps(c(1:10), c("<2", "2-5", ">5"))
create_subgrps(c(1:10), c("<=2", ">2-5", ">5"))
create_subgrps(c(1:10), c("<2", "2<5", ">=5"))

create_var_from_codelist

Create Variable from Codelist

Description

This function uses code/decode pairs from a metacore object to create new variables in the data
**create_var_from_codelist**

**Usage**

```r
create_var_from_codelist(
  data,
  metacore,
  input_var,
  out_var,
  decode_to_code = TRUE
)
```

**Arguments**

- **data**: Dataset that contains the input variable
- **metacore**: A metacore object to get the codelist from. If the `out_var` has different codelists for different datasets the metacore object will need to be subsetted using `select_dataset` from the metacore package.
- **input_var**: Name of the variable that will be translated for the new column
- **out_var**: Name of the output variable. Note: the grouping will always be from the code of the codelist associates with `out_var`
- **decode_to_code**: Direction of the translation. By default assumes the `input_var` is the decode column of the codelist. Set to `FALSE` if the `input_var` is the code column of the codelist

**Value**

Dataset with a new column added

**Examples**

```r
library(metacore)
library(tibble)
data <- tribble(
  ~USUBJID, ~VAR1, ~VAR2,
  1, "M", "Male",
  2, "F", "Female",
  3, "F", "Female",
  4, "U", "Unknown",
  5, "M", "Male",
)
spec <- spec_to_metacore(metacore_example("p21_mock.xlsx"), quiet = TRUE)
create_var_from_codelist(data, spec, VAR2, SEX)
create_var_from_codelist(data, spec, "VAR2", "SEX")
create_var_from_codelist(data, spec, VAR1, SEX, decode_to_code = FALSE)
```
### drop_unspec_vars  
**Drop Unspecified Variables**

**Description**
This function drops all unspecified variables. It will throw an error if the dataset does not contain all expected variables.

**Usage**

```r
drop_unspec_vars(dataset, metacore, dataset_name = NULL)
```

**Arguments**

- `dataset`: Dataset to change
- `metacore`: metacore object that only contains the specifications for the dataset of interest.
- `dataset_name`: Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetted.

**Value**
Dataset with only specified columns

**Examples**

```r
library(metacore)
library(haven)
library(dplyr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt")) %>%
  select(USUBJID, SITEID) %>%
  mutate(foo = "Hello")
drop_unspec_vars(data, spec)
```

### get_bad_ct  
**Gets vector of control terminology which should be there**

**Description**
This function checks the column in the dataset only contains the control terminology as defined by the metacore specification. It will return all values not found in the control terminology.

**Usage**

```r
get_bad_ct(data, metacore, var, na_acceptable = NULL)
```
### make_supp_qual

**Arguments**

- **data**
  Data to check

- **metacore**
  A metacore object to get the codelist from. If the variable has different codelists for different datasets the metacore object will need to be subsetted using `select_dataset` from the metacore package.

- **var**
  Name of variable to check

- **na_acceptable**
  Logical value, set to ‘NULL’ by default, so the acceptability of missing values is based on if the core for the variable is “Required” in the `metacore` object. If set to ‘TRUE’ then will pass check if values are in the control terminology or are missing. If set to ‘FALSE’ then NA will not be acceptable.

**Value**

- vector

---

### Description

Make Supplemental Qualifier

### Usage

```r
make_supp_qual(dataset, metacore, dataset_name = NULL)
```

### Arguments

- **dataset**
  dataset the supp will be pulled from

- **metacore**
  A subsetted metacore object to get the supp information from. If not already subsetted then a `dataset_name` will need to be provided

- **dataset_name**
  optional name of dataset

**Value**

- a CDISC formatted SUPP dataset

### Examples

```r
library(metacore)
library(safetyData)
library(tibble)
load(metacore_example("pilot_SDTM.rda"))
spec <- metacore %>% select_dataset("AE")
ae <- combine_supp(sdtm_ae, sdtm_suppae)
make_supp_qual(ae, spec) %>% as_tibble()
```
metatools_example

*Get path to pkg example*

**Description**

pkg comes bundled with a number of sample files in its ‘inst/extdata’ directory. This function makes them easy to access.

**Usage**

```r
metatools_example(file = NULL)
```

**Arguments**

- `file`: Name of file. If ‘NULL’, the example files will be listed.

**Examples**

```r
metatools_example()
metatools_example("dm.xpt")
```

---

**order_cols**

*Sort Columns by Order*

**Description**

This function sorts the dataset according to the order found in the metacore object.

**Usage**

```r
order_cols(data, metacore, dataset_name = NULL)
```

**Arguments**

- `data`: Dataset to sort
- `metacore`: metacore object that contains the specifications for the dataset of interest.
- `dataset_name`: Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetting.

**Value**

dataset with ordered columns
Examples

library(metacore)
library(haven)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt"))
order_cols(data, spec)

remove_labels(data)

Description

This function allows a user to remove all labels to a dataframe at once.

Usage

remove_labels(data)

Arguments

data

A data.frame or tibble

Value

data with variable labels applied

Examples

library(haven)
data <- read_xpt(metatools_example("adsl.xpt"))
remove_labels(data)

set_variable_labels(data, metacore, dataset_name = NULL)

Description

This function leverages metadata available in a metacore object to apply labels to a data frame.

Usage

set_variable_labels(data, metacore, dataset_name = NULL)
Arguments

data A dataframe or tibble upon which labels will be applied
metacore metacore object that contains the specifications for the dataset of interest.
dataset_name Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetted.

Value

Dataframe with labels applied

Examples

mc <- metacore::spec_to_metacore(
    metacore::metacore_example("p21_mock.xlsx"),
    quiet=TRUE
)
dm <- haven::read_xpt(metatools_example("dm.xpt"))
set_variable_labels(dm, mc, dataset_name = "DM")

sort_by_key Sort Rows by Key Sequence

Description

This function sorts the dataset according to the key sequence found in the metacore object.

Usage

sort_by_key(data, metacore, dataset_name = NULL)

Arguments

data Dataset to sort
metacore metacore object that contains the specifications for the dataset of interest.
dataset_name Optional string to specify the dataset. This is only needed if the metacore object provided hasn’t already been subsetted.

Value

dataset with ordered columns
Examples

library(metacore)
library(haven)
library(magrittr)
load(metacore_example("pilot_ADaM.rda"))
spec <- metacore %>% select_dataset("ADSL")
data <- read_xpt(metatools_example("adsl.xpt"))
sort_by_key(data, spec)
Index

add_labels, 2
add_variables, 3
build_from_derived, 3
build_qnam, 4
check_ct_col, 5
check_ct_data, 6
check_variables, 7
combine_supp, 7
convert_var_to_fct, 8
create_cat_var, 9
create_subgrps, 10
create_var_from_codelist, 10
drop_unspec_vars, 12
get_bad_ct, 12
make_supp_qual, 13
metatools_example, 14
order_cols, 14
remove_labels, 15
set_variable_labels, 15
sort_by_key, 16