

Package ‘metatools’

May 8, 2026

Type Package

Title Enable the Use of 'metacore' to Help Create and Check Dataset

Version 0.3.0

Description Uses the metadata information stored in 'metacore' objects to check and build metadata associated columns.

License MIT + file LICENSE

URL <https://github.com/pharmaverse/metatools>,
<https://pharmaverse.github.io/metatools/>

BugReports <https://github.com/pharmaverse/metatools/issues>

Depends R (>= 4.1.0)

Imports cli, dplyr, lifecycle, magrittr, metacore (>= 0.2.0), purrr,
rlang, stringr, tibble, tidyr

Suggests covr, haven, pharmaversesdtm, safetyData, spelling, testthat
(>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Liam Hobby [aut, cre],
Christina Fillmore [aut] (ORCID:
<<https://orcid.org/0000-0003-0595-2302>>),
Bill Denney [aut],
Mike Stackhouse [aut] (ORCID: <<https://orcid.org/0000-0001-6030-723X>>),
Jana Stoilova [aut],
Tamara Senior [aut],
GlaxoSmithKline LLC [cph, fnd],
F. Hoffmann-La Roche AG [cph, fnd],
Atorus Research LLC [cph, fnd]

Maintainer Liam Hobby <liam.f.hobby@gsk.com>

Repository CRAN

Date/Publication 2026-04-28 11:30:02 UTC

Contents

add_labels	2
add_variables	3
build_from_derived	4
build_qnam	5
check_ct_col	6
check_ct_data	7
check_unique_keys	9
check_variables	10
combine_supp	11
convert_var_to_fct	11
create_cat_var	12
create_subgrps	13
create_var_from_codelist	14
drop_unspec_vars	16
get_bad_ct	17
make_supp_qual	18
metatools_example	18
order_cols	19
remove_labels	20
set_variable_labels	20
sort_by_key	21
Index	23

add_labels	<i>Apply labels to multiple variables on a data frame</i>
------------	---

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

```
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 = deprecated())
```

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. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.

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

```
build_from_derived(
  metacore,
  ds_list,
  dataset_name = deprecated(),
  predecessor_only = TRUE,
  keep = FALSE,
  verbose = c("message", "warn", "silent")
)
```

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. If the list is unnamed, then it will use the names of the objects.
dataset_name	[Deprecated] Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.
predecessor_only	By default TRUE, so only variables with the origin of 'Predecessor' will be used. If FALSE any derivation matching the dataset.variable will be used.
keep	String to determine which columns from the original datasets should be kept <ul style="list-style-type: none"> • "FALSE" (default): only columns that are also present in the ADaM specification are kept in the output. • "ALL": all original columns are carried through to the ADaM, including those that have been renamed. e.g. if DM.ARM is a predecessor to DM.TRTO1P, both ARM and TRTO1P will be present as columns in the ADaM output. • "PREREQUISITE": columns are retained if they are required for future derivations in the specification. Additional prerequisite columns are identified as columns that appear in the 'derivation' column of the metacore object in the format "DATASET.VARIABLE", but not as direct predecessors.

Predecessors are defined as columns where the derivation is a 1:1 copy of a column in a source dataset. e.g. derivation = "VS.VSTESTCD" is a predecessor, while derivation = "Value of VS.VSSTRESN where VS.VSTESTCD == 'Heart Rate'" contains both VS.VSTESTCD and VS.VSSTRESN as prerequisites, and these columns will be kept through to the ADaM.

verbose Character string controlling message verbosity. One of:
 "message" Show both warnings and messages (default)
 "warn" Show warnings but suppress messages
 "silent" Suppress all warnings and messages

Value

dataset

Examples

```
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

```
build_qnam(
  dataset,
  qnam,
  qlabel,
  idvar,
  qeval,
  qorig,
  verbose = c("message", "warn", "silent")
)
```

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
verbose	Character string controlling message verbosity. One of: "message" Show both warnings and messages (default) "warn" Show warnings but suppress messages "silent" Suppress all warnings and messages

Value

Observations structured in SUPP format

check_ct_col	<i>Check Control Terminology for a Single Column</i>
--------------	--

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,
  verbose = "message",
  .internal = FALSE
)
```

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 <code>select_dataset</code> 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.
verbose	character string controlling the verbosity of the output. Possible values are "message" (for general information and success messages) and "warn" (for warnings). Partial matching is allowed. Important: "silent" is explicitly not a valid option for verbose in this function. The primary purpose of check_ct_data is to identify and warn the user about non-compliant or problematic control terminology. Allowing the suppression of these warnings would bypass the function's intent and could lead to unnoticed data quality issues. If verbose = "silent" is provided, it will be coerced to "message" with a warning.
.internal	Logical value indicating whether the function is being called internally by another package function. If TRUE, the function suppresses user-facing messages and instead returns a logical indicator of whether any controlled terminology violations were detected. This argument is intended for internal use only and should not be set by end users.

Value

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

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"))
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

```
check_ct_data(
  data,
  metacore,
```

```

na_acceptable = NULL,
omit_vars = NULL,
verbose = "message"
)

```

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 <code>select_dataset</code> from the metacore package.
na_acceptable	logical value or character vector, set to NULL by default. NULL sets the acceptability of missing values 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. If set to a character vector then only the specified variables may contain NA values.
omit_vars	character vector indicating which variables should be skipped when doing the controlled terminology checks. Internally, <code>omit_vars</code> is evaluated before <code>na_acceptable</code> .
verbose	character string controlling the verbosity of the output. Possible values are "message" (for general information and success messages) and "warn" (for warnings). Partial matching is allowed. Important: "silent" is explicitly not a valid option for verbose in this function. The primary purpose of <code>check_ct_data</code> is to identify and warn the user about non-compliant or problematic control terminology. Allowing the suppression of these warnings would bypass the function's intent and could lead to unnoticed data quality issues. If <code>verbose = "silent"</code> is provided, it will be coerced to "message" with a warning.

Value

Given data if all columns pass. It will issue a warning otherwise.

Examples

```

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

check_ct_data(data, spec, omit_vars = c("AGEGR2", "AGEGR2N"))
## Not run:
# These examples produce errors:
check_ct_data(data, spec, na_acceptable = FALSE)
check_ct_data(data, spec, na_acceptable = FALSE, omit_vars = "DISCONFL")
check_ct_data(data, spec, na_acceptable = c("DSRAEFL", "DCSREAS"), omit_vars = "DISCONFL")

## End(Not run)

```

check_unique_keys	<i>Check Uniqueness of Records by Key</i>
-------------------	---

Description

This function checks the uniqueness of records in the dataset by key using `get_keys` from the `metacore` package. If the key uniquely identifies each record the function will print a message stating everything is as expected. If records are not uniquely identified an error will explain the duplicates.

Usage

```
check_unique_keys(data, metacore, dataset_name = deprecated())
```

Arguments

<code>data</code>	Dataset to check
<code>metacore</code>	metacore object that only contains the specifications for the dataset of interest.
<code>dataset_name</code>	[Deprecated] Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The <code>dataset_name</code> argument will be removed in a future release. Please use <code>metacore::select_dataset</code> to subset the metacore object to obtain metadata for a single dataset.

Value

message if the key uniquely identifies each dataset record, 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_unique_keys(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 print a message stating everything is as expected. If there are additional or missing variables an error will explain the discrepancies

Usage

```
check_variables(data, metacore, dataset_name = deprecated(), strict = FALSE)
```

Arguments

data	Dataset to check
metacore	metacore object that only contains the specifications for the dataset of interest.
dataset_name	[Deprecated] Optional string to specify the dataset. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.
strict	A logical value indicating whether to perform strict validation on the input dataset. If TRUE (default), errors will be raised if validation fails. If FALSE, warnings will be issued instead, allowing the function execution to continue even with invalid data.

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)
data["DUMMY_COL"] <- NA
check_variables(data, spec, strict = FALSE)
```

combine_supp	<i>Combine the Domain and Supplemental Qualifier</i>
--------------	--

Description

Combine the Domain and Supplemental Qualifier

Usage

```
combine_supp(dataset, supp)
```

Arguments

dataset	Domain dataset
supp	Supplemental Qualifier dataset

Value

a dataset with the supp variables added to it

Examples

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

convert_var_to_fct	<i>Convert Variable to Factor with Levels Set by Control Terms</i>
--------------------	--

Description

This functions 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

```
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 subsetted using <code>select_dataset</code> from the metacore package
var	Name of variable to change

Value

Dataset with variable changed to a factor

Examples

```
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

```
create_cat_var(
  data,
  metacore,
  ref_var,
  grp_var,
  num_grp_var = NULL,
  create_from_decode = FALSE,
  strict = TRUE
)
```

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

`create_from_decode` Sets the decode column of the codelist as the column from which the variable will be created. By default the column is code.

`strict` A logical value indicating whether to perform strict checking against the codelist. If TRUE will issue a warning if values in the `ref_var` column do not fit into the group definitions for the codelist in `grp_var`. If FALSE no warning is issued and values not defined by the codelist will likely result in NA results.

Value

dataset with new column added

Examples

```
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, AGE)
# Grouping Column Only
create_cat_var(dm, spec, AGE, AGEGR1)
# Grouping Column and Numeric Decode
create_cat_var(dm, spec, AGE, AGEGR1, AGEGR1N)
```

create_subgrps *Create Subgroups*

Description

Create Subgroups

Usage

```
create_subgrps(ref_vec, grp_defs, grp_labs = NULL)
```

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`

`grp_labs` Vector of strings with labels defined. The labels correspond to the associated `grp_defs`. i.e., "12-17" may translate to "12-17 years". If no `grp_labs` specified then `grp_defs` will be used.

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

```

```
create_var_from_codelist
```

Create Variable from Codelist

Description

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

Usage

```

create_var_from_codelist(
  data,
  metacore,
  input_var,
  out_var,
  codelist = NULL,
  decode_to_code = TRUE,
  strict = TRUE
)

```

Arguments

<code>data</code>	Dataset that contains the input variable
<code>metacore</code>	A metacore object to get the codelist from. This should be a subsetted metacore object (of subclass <code>DatasetMeta</code>) created using <code>metacore::select_dataset</code> .
<code>input_var</code>	Name of the variable that will be translated for the new column
<code>out_var</code>	Name of the output variable. Note: Unless a codelist is provided the grouping will always be from the code of the codelist associates with <code>out_var</code> .
<code>codelist</code>	Optional argument to supply a codelist. Must be a <code>data.frame</code> with <code>code</code> and <code>decode</code> columns such as those created by the function <code>metacore::get_control_term</code> . If no codelist is provided the codelist associated with the column supplied to <code>out_var</code> will be used. By default <code>codelist</code> is <code>NULL</code> .
<code>decode_to_code</code>	Direction of the translation. Default value is <code>TRUE</code> , i.e., assumes the <code>input_var</code> is the <code>decode</code> column of the codelist. Set to <code>FALSE</code> if the <code>input_var</code> is the <code>code</code> column of the codelist.
<code>strict</code>	A logical value indicating whether to perform strict checking against the codelist. If <code>TRUE</code> will issue a warning if values in the <code>input_var</code> column are not present in the codelist. If <code>FALSE</code> no warning is issued and values not present in the codelist will likely result in <code>NA</code> results.

Value

Dataset with a new column added

Examples

```
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)
dm_spec <- select_dataset(spec, "DM", quiet = TRUE)
create_var_from_codelist(data, dm_spec, VAR2, SEX)
create_var_from_codelist(data, dm_spec, "VAR2", "SEX")
create_var_from_codelist(data, dm_spec, VAR1, SEX, decode_to_code = FALSE)

# Example providing a custom codelist
# This example also reverses the direction of translation
load(metacore_example("pilot_ADaM.rda"))
adlb_spec <- select_dataset(metacore, "ADLBC", quiet = TRUE)
adlb <- tibble(PARAMCD = c("ALB", "ALP", "ALT", "AST", "BILI", "BUN"))
create_var_from_codelist(
  adlb,
  adlb_spec,
  PARAMCD,
  PARAM,
  codelist = get_control_term(adlb_spec, PARAMCD),
  decode_to_code = FALSE,
  strict = FALSE
)

## Not run:
# Example expecting warning where `strict` == `TRUE`
adlb <- tibble(PARAMCD = c("ALB", "ALP", "ALT", "AST", "BILI", "BUN", "DUMMY1", "DUMMY2"))
create_var_from_codelist(
  adlb,
  adlb_spec,
  PARAMCD,
  PARAM,
  codelist = get_control_term(adlb_spec, PARAMCD),
  decode_to_code = FALSE,
  strict = TRUE
)

## End(Not run)
```

drop_unspec_vars *Drop Unspecified Variables*

Description

This function drops all unspecified variables.

Usage

```
drop_unspec_vars(
  dataset,
  metacore,
  dataset_name = deprecated(),
  verbose = c("message", "warn", "silent")
)
```

Arguments

dataset	Dataset to change
metacore	metacore object that only contains the specifications for the dataset of interest.
dataset_name	[Deprecated] Optional string to specify the dataset. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.
verbose	Character string controlling message verbosity. One of: "message" Show both warnings and messages (default) "warn" Show warnings but suppress messages "silent" Suppress all warnings and messages

Value

Dataset with only specified columns

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(USUBJID, SITEID) %>%
  mutate(foo = "Hello")
drop_unspec_vars(data, spec)
```

get_bad_ct	<i>Gets vector of control terminology which should be there</i>
------------	---

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

```
get_bad_ct(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 <code>select_dataset</code> from the metacore package.
var	Name of variable to check
na_acceptable	Logical value, set to <code>NULL</code> 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 <code>TRUE</code> then will pass check if values are in the control terminology or are missing. If set to <code>FALSE</code> then NA will not be acceptable.

Value

vector

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"))
get_bad_ct(data, spec, "DCSREAS")
get_bad_ct(data, spec, "DCSREAS", na_acceptable = FALSE)
```

make_supp_qual	<i>Make Supplemental Qualifier</i>
----------------	------------------------------------

Description

Make Supplemental Qualifier

Usage

```
make_supp_qual(dataset, metacore, dataset_name = deprecated())
```

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	[Deprecated] Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.

Value

a CDISC formatted SUPP dataset

Examples

```
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	<i>Get path to pkg example</i>
-------------------	--------------------------------

Description

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

Usage

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

Arguments

file Name of file. If NULL, the example files will be listed.

Examples

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

order_cols	<i>Sort Columns by Order</i>
------------	------------------------------

Description

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

Usage

```
order_cols(data, metacore, dataset_name = deprecated())
```

Arguments

data Dataset to sort

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

dataset_name **[Deprecated]** Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use metacore::select_dataset to subset the metacore object to obtain metadata for a single dataset.

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	<i>Remove labels to multiple variables on a data frame</i>
---------------	--

Description

This function allows a user to removes 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	<i>Apply labels to a data frame using a metacore object</i>
---------------------	---

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 = deprecated(),
  verbose = c("message", "warn", "silent")
)
```

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	[Deprecated] Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use <code>metacore::select_dataset</code> to subset the metacore object to obtain metadata for a single dataset.
verbose	Character string controlling message verbosity. One of: "message" Show both warnings and messages (default) "warn" Show warnings but suppress messages "silent" Suppress all warnings and messages

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 = deprecated())
```

Arguments

data	Dataset to sort
metacore	metacore object that contains the specifications for the dataset of interest.
dataset_name	[Deprecated] Optional string to specify the dataset that is being built. This is only needed if the metacore object provided hasn't already been subsetted. Note: Deprecated in version 0.2.0. The dataset_name argument will be removed in a future release. Please use <code>metacore::select_dataset</code> to subset the metacore object to obtain metadata for a single dataset.

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, 4
build_qnam, 5

check_ct_col, 6
check_ct_data, 7
check_unique_keys, 9
check_variables, 10
combine_supp, 11
convert_var_to_fct, 11
create_cat_var, 12
create_subgrps, 13
create_var_from_codelist, 14

drop_unspec_vars, 16

get_bad_ct, 17

make_supp_qual, 18
metatools_example, 18

order_cols, 19

remove_labels, 20

set_variable_labels, 20
sort_by_key, 21