

# Package: metatools (via r-universe)

August 23, 2024

**Type** Package

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

**Version** 0.1.6

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

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Imports** dplyr, metacore (>= 0.0.4), purrr, rlang, stringr, tidyr, tibble, magrittr

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

**Config/testthat/edition** 3

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

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

**Language** en-US

**NeedsCompilation** no

**Author** Christina Fillmore [aut, cre]  
(<<https://orcid.org/0000-0003-0595-2302>>), Mike Stackhouse [aut] (<<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** Christina Fillmore <[christina.e.fillmore@gsk.com](mailto:christina.e.fillmore@gsk.com)>

**Repository** CRAN

**Date/Publication** 2024-07-23 16:20:01 UTC

## Contents

add_labels . . . . .	2
add_variables . . . . .	3
build_from_derived . . . . .	4
build_qnam . . . . .	5
check_ct_col . . . . .	5
check_ct_data . . . . .	6
check_unique_keys . . . . .	7
check_variables . . . . .	8
combine_supp . . . . .	9
convert_var_to_fct . . . . .	9
create_cat_var . . . . .	10
create_subgrps . . . . .	11
create_var_from_codelist . . . . .	12
drop_unspec_vars . . . . .	13
get_bad_ct . . . . .	14
make_supp_qual . . . . .	15
metatools_example . . . . .	15
order_cols . . . . .	16
remove_labels . . . . .	17
set_variable_labels . . . . .	17
sort_by_key . . . . .	18
<b>Index</b>	<b>19</b>

---

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	<i>Add Missing Variables</i>
---------------	------------------------------

---

## 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

```
build_from_derived(
  metacore,
  ds_list,
  dataset_name = NULL,
  predecessor_only = TRUE,
  keep = FALSE
)
```

### 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	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 'TRUE', so only variables with the origin of 'Predecessor' will be used. If 'FALSE' any derivation matching the dataset.variable will be used.
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

```
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)
```

### Arguments

dataset	Input dataset
qnam	QNAM value
qlabel	QLABEL value
idvar	IDVAR variable name (provided as a string)
qeval	QEQVAL 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**

```
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)
```

**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 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, 'omit_vars' is evaluated before 'na_acceptable'.

### Value

Given data if all columns pass. It will 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_ct_data(data, spec)
## 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 = 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 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	<i>Check Variable Names</i>
-----------------	-----------------------------

---

**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 = 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)
```

**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

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


---

**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 ‘select_dataset’ 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)
```

**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**

```
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	<i>Create Subgroups</i>
----------------	-------------------------

---

**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 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,  
  decode_to_code = TRUE  
)
```

## Arguments

<code>data</code>	Dataset that contains the input variable
<code>metacore</code>	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.
<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: the grouping will always be from the code of the codelist associates with 'out_var'
<code>decode_to_code</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

```
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

```
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

```
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

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

### Arguments

<code>data</code>	Data to check
<code>metacore</code>	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.
<code>var</code>	Name of variable to check
<code>na_acceptable</code>	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 <code>'metacore'</code> 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, "DISCONFL")
get_bad_ct(data, spec, "DISCONFL", na_acceptable = FALSE)
```

---

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

---

**Description**

Make Supplemental Qualifier

**Usage**

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

**Arguments**

dataset	dataset the supp will be pulled from
metacore	A subsetting metacore object to get the supp information from. If not already subsetting then a 'dataset_name' will need to be provided
dataset_name	optional name of 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_suppaes)
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 = 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"))  
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 = 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, 4  
build\_qnam, 5  
  
check\_ct\_col, 5  
check\_ct\_data, 6  
check\_unique\_keys, 7  
check\_variables, 8  
combine\_supp, 9  
convert\_var\_to\_fct, 9  
create\_cat\_var, 10  
create\_subgrps, 11  
create\_var\_from\_codelist, 12  
  
drop\_unspec\_vars, 13  
  
get\_bad\_ct, 14  
  
make\_supp\_qual, 15  
metatools\_example, 15  
  
order\_cols, 16  
  
remove\_labels, 17  
  
set\_variable\_labels, 17  
sort\_by\_key, 18