

Package ‘admiral’

May 31, 2022

Type Package

Title ADaM in R Asset Library

Version 0.7.0

Description A toolbox for programming Clinical Data Standards Interchange Consortium (CDISC) compliant Analysis Data Model (ADaM) datasets in R. ADaM datasets are a mandatory part of any New Drug or Biologics License Application submitted to the United States Food and Drug Administration (FDA). Analysis derivations are implemented in accordance with the “Analysis Data Model Implementation Guide” (CDISC Analysis Data Model Team, 2021, <<https://www.cdisc.org/standards/foundational/adam/adamig-v1-3-release-package>>).

Language en-US

License Apache License (>= 2)

URL <https://pharmaverse.github.io/admiral/index.html>,
<https://github.com/pharmaverse/admiral/>

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Depends R (>= 3.5)

Imports assertthat, dplyr, hms, lifecycle, lubridate, magrittr, purrr,
rlang, stringr, tidyr, tidyselect

Suggests admiral.test, devtools, diffdf, knitr, lintr, pkgdown,
testthat, methods, miniUI, rmarkdown, roxygen2, spelling,
styler, tibble, usethis, covr, DT

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Neitmann [aut, cre],
Stefan Bundfuss [aut],
Ben Straub [aut],
Samia Kabi [aut],
Gordon Miller [aut],
Teckla Akinyi [aut],

Andrew Smith [aut],
 Konstantina Koukourikou [aut],
 Ross Farrugia [aut],
 Eric Simms [aut],
 Annie Yang [aut],
 Robin Koeger [aut],
 Sophie Shapcott [aut],
 Ojesh Upadhyay [aut],
 Jack McGavigan [aut],
 Kamila Duniec [aut],
 Gayatri G [aut],
 Alana Harris [aut],
 Mahdi About [aut],
 Pooja Kumari [aut],
 Claudia Carlucci [aut],
 Daniil Stefonishin [aut],
 Michael Thorpe [ctb],
 Pavan Kumar [ctb],
 Hamza Rahal [ctb],
 Alice Ehmann [ctb],
 Tom Ratford [ctb],
 Vignesh Thanikachalam [ctb],
 Ondrej Slama [ctb],
 Shimeng Huang [ctb],
 James Kim [ctb],
 F. Hoffmann-La Roche AG [cph, fnd],
 GlaxoSmithKline LLC [cph, fnd]

Maintainer Thomas Neitmann <thomas.neitmann@roche.com>

Repository CRAN

Date/Publication 2022-05-31 16:40:02 UTC

R topics documented:

admiral_adae	6
admiral_adcm	6
admiral_adeg	7
admiral_adex	7
admiral_adpp	8
admiral_adsl	8
admiral_adv	9
assert_character_scalar	9
assert_character_vector	10
assert_data_frame	11
assert_db_requirements	12
assert_filter_cond	13
assert_function	14
assert_has_variables	15

assert_integer_scalar	16
assert_list_element	17
assert_list_of	18
assert_logical_scalar	19
assert_numeric_vector	20
assert_one_to_one	21
assert_order_vars	21
assert_param_does_not_exist	22
assert_s3_class	23
assert_symbol	24
assert_terms	25
assert_unit	26
assert_valid_queries	27
assert_vars	28
assert_varval_list	29
call_derivation	30
call_user_fun	31
sensor_source	32
compute_bmi	33
compute_bsa	34
compute_dtf	35
compute_duration	36
compute_map	38
compute_qtc	39
compute_rr	40
compute_tmf	41
convert_blanks_to_na	42
convert_date_to_dtm	43
convert_dtc_to_dt	45
convert_dtc_to_dtm	47
create_query_data	49
create_single_dose_dataset	53
dataset_vignette	55
date_source	56
death_event	58
default_qtc_paramcd	59
derivation_slice	60
derive_derived_param	60
derive_extreme_records	63
derive_last_dose	66
derive_param_bmi	69
derive_param_bsa	71
derive_param_doseint	73
derive_param_exist_flag	75
derive_param_exposure	79
derive_param_first_event	81
derive_param_map	84
derive_param_qtc	87

derive_param_rr	89
derive_param_tte	91
derive_param_wbc_abs	95
derive_summary_records	98
derive_vars_aage	100
derive_vars_atc	102
derive_vars_disposition_reason	103
derive_vars_dt	106
derive_vars_dtm	110
derive_vars_dtm_to_dt	114
derive_vars_dtm_to_tm	115
derive_vars_duration	116
derive_vars_dy	118
derive_vars_last_dose	120
derive_vars_merged	122
derive_vars_merged_dt	125
derive_vars_merged_dtm	129
derive_vars_query	133
derive_vars_suppqual	134
derive_vars_transposed	135
derive_var_ady	137
derive_var_aendy	138
derive_var_agegr_fda	139
derive_var_age_years	140
derive_var_analysis_ratio	141
derive_var_anrind	143
derive_var_astdy	144
derive_var_atirel	145
derive_var_base	146
derive_var_basetype	148
derive_var_chg	150
derive_var_disposition_dt	151
derive_var_disposition_status	152
derive_var_dthcaus	154
derive_var_extreme_dt	157
derive_var_extreme_dtm	160
derive_var_extreme_flag	163
derive_var_last_dose_amt	167
derive_var_last_dose_date	170
derive_var_last_dose_grp	172
derive_var_lstalvdt	174
derive_var_merged_cat	175
derive_var_merged_character	177
derive_var_merged_exist_flag	180
derive_var_obs_number	182
derive_var_ontrfl	183
derive_var_pchg	187
derive_var_shift	188

derive_var_trtdurd	189
derive_var_trtedtm	191
derive_var_trtsdtm	192
derive_var_worst_flag	193
dose_freq_lookup	195
dquote	196
dthcaus_source	197
event_source	198
expect_dfs_equal	199
extend_source_datasets	200
extract_duplicate_records	201
extract_unit	202
ex_single	203
filter_date_sources	203
filter_extreme	205
filter_if	207
filter_relative	208
format.sdg_select	210
format.smq_select	211
format_eoxxstt_default	212
format_reason_default	213
get_duplicates_dataset	214
get_many_to_one_dataset	215
get_one_to_many_dataset	216
get_terms_from_db	217
impute_dtc	218
is_auto	221
list_all_templates	222
lstalvdt_source	223
negate_vars	224
params	224
print.derivation_slice	226
print.tte_source	227
queries	227
query	228
restrict_derivation	230
sdg_select	232
signal_duplicate_records	232
slice_derivation	233
smq_select	235
suppress_warning	236
tte_source	237
use_ad_template	238
validate_query	239
validate_sdg_select	239
validate_smq_select	240
vars2chr	241
warn_if_inconsistent_list	241

warn_if_invalid_dtc	242
warn_if_vars_exist	243
yn_to_numeric	244

Index	245
--------------	------------

admiral_adae	<i>Adverse Event Analysis Dataset</i>
--------------	---------------------------------------

Description

An example adverse event analysis dataset

Usage

```
admiral_adae
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1191 rows and 101 columns.

Source

Derived from the `ads1` and `ae` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adae.R)

admiral_adcm	<i>Concomitant Medication Analysis Dataset</i>
--------------	--

Description

An example concomitant medication analysis dataset

Usage

```
admiral_adcm
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7510 rows and 89 columns.

Source

Derived from the `ads1` and `cm` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adcm.R)

admiral_adeq	<i>ECG Analysis Dataset</i>
--------------	-----------------------------

Description

An example ECG analysis dataset

Usage

```
admiral_adeq
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 78614 rows and 104 columns.

Source

Derived from the `adsl` and `eg` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adeq.R)

admiral_adex	<i>Exposure Analysis Dataset</i>
--------------	----------------------------------

Description

An example exposure analysis dataset

Usage

```
admiral_adex
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 6315 rows and 87 columns.

Source

Derived from the `adsl` and `ex` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adex.R)

`admiral_adpp`*Pharmacokinetics Parameters Analysis Dataset*

Description

An example pharmacokinetics parameters analysis dataset

Usage

```
admiral_adpp
```

Format

An object of class `data.frame` with 1344 rows and 72 columns.

Source

Derived from the `adsl` and `pp` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adpp.R)

`admiral_adsl`*Subject Level Analysis Dataset*

Description

An example subject level analysis dataset

Usage

```
admiral_adsl
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 306 rows and 48 columns.

Source

Derived from the `dm` and `ds` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adsl.R)

admiral_adv	<i>Vital Signs Analysis Dataset</i>
-------------	-------------------------------------

Description

An example vital signs analysis dataset

Usage

```
admiral_adv
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 57763 rows and 102 columns.

Source

Derived from the `adsl` and `vs` datasets using `{admiral}` (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adv.R)

assert_character_scalar	
-------------------------	--

Is an Argument a Character Scalar (String)?

Description

Checks if an argument is a character scalar and (optionally) whether it matches one of the provided values.

Usage

```
assert_character_scalar(
  arg,
  values = NULL,
  case_sensitive = TRUE,
  optional = FALSE
)
```

Arguments

<code>arg</code>	A function argument to be checked
<code>values</code>	A character vector of valid values for <code>arg</code>
<code>case_sensitive</code>	Should the argument be handled case-sensitive? If set to <code>FALSE</code> , the argument is converted to lower case for checking the permitted values and returning the argument.
<code>optional</code>	Is the checked parameter optional? If set to <code>FALSE</code> and <code>arg</code> is <code>NULL</code> then an error is thrown

Value

The function throws an error if `arg` is not a character vector or if `arg` is a character vector but of length > 1 or if its value is not one of the values specified. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
example_fun <- function(msg_type) {
  assert_character_scalar(msg_type, values = c("warning", "error"))
}

example_fun("warning")

try(example_fun("message"))

try(example_fun(TRUE))

# handling parameters case-insensitive
example_fun2 <- function(msg_type) {
  msg_type <- assert_character_scalar(
    msg_type,
    values = c("warning", "error"),
    case_sensitive = FALSE
  )
  if (msg_type == "warning") {
    print("A warning was requested.")
  }
}

example_fun2("Warning")
```

assert_character_vector

Is an Argument a Character Vector?

Description

Checks if an argument is a character vector

Usage

```
assert_character_vector(arg, values = NULL, optional = FALSE)
```

Arguments

arg	A function argument to be checked
values	A character vector of valid values for arg
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not a character vector or if any element is not included in the list of valid values. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
example_fun <- function(chr) {
  assert_character_vector(chr)
}

example_fun(letters)

try(example_fun(1:10))
```

assert_data_frame *Is an Argument a Data Frame?*

Description

Checks if an argument is a data frame and (optionally) whether it contains a set of required variables

Usage

```
assert_data_frame(
  arg,
  required_vars = NULL,
  check_is_grouped = TRUE,
  optional = FALSE
)
```

Arguments

arg	A function argument to be checked
required_vars	A list of variables created using vars()
check_is_grouped	Throw an error if dataset is grouped? Defaults to TRUE.
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if `arg` is not a data frame or if `arg` is a data frame but misses any variable specified in `required_vars`. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
library(admiral.test)
data(admiral_dm)

example_fun <- function(dataset) {
  assert_data_frame(dataset, required_vars = vars(STUDYID, USUBJID))
}

example_fun(admiral_dm)

try(example_fun(dplyr::select(admiral_dm, -STUDYID)))

try(example_fun("Not a dataset"))
```

assert_db_requirements

Check required parameters for SMQ/SDG

Description

If SMQs or SDGs are requested, the version and a function to access the database must be provided. The function checks these requirements.

Usage

```
assert_db_requirements(
  version,
  version_arg_name,
  fun,
  fun_arg_name,
  queries,
  i,
  type
)
```

Arguments

version	Version provided by user
version_arg_name	Name of the argument providing the version
fun	Function provided by user
fun_arg_name	Name of the argument providing the function
queries	Queries provide by user
i	Index of query being checked
type	Type of query Should be "SMQ" or "SDG".

Value

An error is issued if version or fun is null.

Author(s)

Stefan Bundfuss

assert_filter_cond *Is an Argument a Filter Condition?*

Description

Is an Argument a Filter Condition?

Usage

```
assert_filter_cond(arg, optional = FALSE)
```

Arguments

arg	Quosure - filtering condition.
optional	Logical - is the argument optional? Defaults to FALSE.

Details

Check if arg is a suitable filtering condition to be used in functions like subset or dplyr::filter.

Value

Performs necessary checks and returns arg if all pass. Otherwise throws an informative error.

Author(s)

Ondrej Slama

Examples

```
library(admiral.test)
data(admiral_dm)

# typical usage in a function as a parameter check
example_fun <- function(dat, x) {
  x <- assert_filter_cond(rlang::enquo(x))
  dplyr::filter(dat, !!x)
}

example_fun(admiral_dm, AGE == 64)

try(example_fun(admiral_dm, USUBJID))
```

assert_function	<i>Is Argument a Function?</i>
-----------------	--------------------------------

Description

Checks if the argument is a function and if all expected parameters are provided by the function.

Usage

```
assert_function(arg, params = NULL, optional = FALSE)
```

Arguments

arg	A function argument to be checked
params	A character vector of expected parameter names
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown.

Value

The function throws an error

- if the argument is not a function or
- if the function does not provide all parameters as specified for the params parameter.

Author(s)

Stefan Bundfuss

Examples

```
example_fun <- function(fun) {  
  assert_function(fun, params = c("x"))  
}  
  
example_fun(mean)  
  
try(example_fun(1))  
  
try(example_fun(sum))
```

assert_has_variables *Does a Dataset Contain All Required Variables?*

Description

Checks if a dataset contains all required variables

Usage

```
assert_has_variables(dataset, required_vars)
```

Arguments

dataset A data.frame
required_vars A character vector of variable names

Value

The function throws an error if any of the required variables are missing in the input dataset. Otherwise, the dataset is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
library(admiral.test)  
data(admiral_dm)  
  
assert_has_variables(admiral_dm, "STUDYID")  
  
try(assert_has_variables(admiral_dm, "AVAL"))
```

assert_integer_scalar *Is an Argument an Integer Scalar?*

Description

Checks if an argument is an integer scalar

Usage

```
assert_integer_scalar(arg, subset = "none", optional = FALSE)
```

Arguments

arg	A function argument to be checked
subset	A subset of integers that arg should be part of. Should be one of "none" (the default), "positive", "non-negative" or "negative".
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not an integer belonging to the specified subset. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
example_fun <- function(num1, num2) {  
  assert_integer_scalar(num1, subset = "positive")  
  assert_integer_scalar(num2, subset = "negative")  
}
```

```
example_fun(1, -9)
```

```
try(example_fun(1.5, -9))
```

```
try(example_fun(2, 0))
```

```
try(example_fun("2", 0))
```

assert_list_element *Is an Element of a List of Lists/Classes Fulfilling a Condition?*

Description

Checks if the elements of a list of named lists/classes fulfill a certain condition. If not, an error is issued and all elements of the list not fulfilling the condition are listed.

Usage

```
assert_list_element(list, element, condition, message_text, ...)
```

Arguments

list	A list to be checked A list of named lists or classes is expected.
element	The name of an element of the lists/classes A character scalar is expected.
condition	Condition to be fulfilled The condition is evaluated for each element of the list. The element of the lists/classes can be referred to by its name, e.g., <code>sensor == 0</code> to check the sensor field of a class.
message_text	Text to be displayed in the message The text should describe the condition to be fulfilled, e.g., "For events the sensor values must be zero."
...	Objects required to evaluate the condition If the condition contains objects apart from the element, they have to be passed to the function. See the second example below.

Value

An error if the condition is not meet. The input otherwise.

Author(s)

Stefan Bundfuss

Examples

```
death <- event_source(  
  dataset_name = "adsl",  
  filter = DTHFL == "Y",  
  date = DTHDT,  
  set_values_to = vars(  
    EVNTDESC = "DEATH",  
    SRCDOM = "ADSL",
```

```

        SRCVAR = "DTHDT"
    )
)

lstalv <- censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = vars(
    EVNTDESC = "LAST KNOWN ALIVE DATE",
    SRCDOM = "ADSL",
    SRCVAR = "LSTALVDT"
  )
)
events <- list(death, lstalv)
try(assert_list_element(
  list = events,
  element = "censor",
  condition = censor == 0,
  message_text = "For events the censor values must be zero."
))

valid_datasets <- c("adrs", "adae")
try(assert_list_element(
  list = events,
  element = "dataset_name",
  condition = dataset_name %in% valid_datasets,
  valid_datasets = valid_datasets,
  message_text = paste0(
    "The dataset name must be one of the following:\n",
    paste(valid_datasets, collapse = ", ")
  )
)
))

```

 assert_list_of

Is an Argument a List of Objects of a Specific S3 Class?

Description

Checks if an argument is a list of objects inheriting from the S3 class specified.

Usage

```
assert_list_of(arg, class, optional = TRUE)
```

Arguments

arg	A function argument to be checked
class	The S3 class to check for
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if `arg` is not a list or if `arg` is a list but its elements are not objects inheriting from `class`. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
example_fun <- function(list) {  
  assert_list_of(list, "data.frame")  
}  
  
example_fun(list(mtcars, iris))  
  
try(example_fun(list(letters, 1:10)))  
  
try(example_fun(c(TRUE, FALSE)))
```

assert_logical_scalar *Is an Argument a Logical Scalar (Boolean)?*

Description

Checks if an argument is a logical scalar

Usage

```
assert_logical_scalar(arg, optional = FALSE)
```

Arguments

<code>arg</code>	A function argument to be checked
<code>optional</code>	Is the checked parameter optional? If set to <code>FALSE</code> and <code>arg</code> is <code>NULL</code> then an error is thrown. Otherwise, <code>NULL</code> is considered as valid value.

Value

The function throws an error if `arg` is neither `TRUE` or `FALSE`. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann, Stefan Bundfuss

Examples

```
example_fun <- function(flag) {  
  assert_logical_scalar(flag)  
}  
  
example_fun(FALSE)  
  
try(example_fun(NA))  
  
try(example_fun(c(TRUE, FALSE, FALSE)))  
  
try(example_fun(1:10))
```

assert_numeric_vector *Is an Argument a Numeric Vector?*

Description

Checks if an argument is a numeric vector

Usage

```
assert_numeric_vector(arg, optional = FALSE)
```

Arguments

arg	A function argument to be checked
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not a numeric vector. Otherwise, the input is returned invisibly.

Author(s)

Stefan Bundfuss

Examples

```
example_fun <- function(num) {  
  assert_numeric_vector(num)  
}  
  
example_fun(1:10)  
  
try(example_fun(letters))
```

assert_one_to_one *Is There a One to One Mapping between Variables?*

Description

Checks if there is a one to one mapping between two lists of variables.

Usage

```
assert_one_to_one(dataset, vars1, vars2)
```

Arguments

dataset	Dataset to be checked The variables specified for vars1 and vars2 are expected.
vars1	First list of variables
vars2	Second list of variables

Value

An error if the condition is not meet. The input otherwise.

Author(s)

Stefan Bundfuss

Examples

```
data(admiral_adsl)
try(
  assert_one_to_one(admiral_adsl, vars(SEX), vars(RACE))
)
```

assert_order_vars *Is an Argument a List of Order Variables?*

Description

Checks if an argument is a valid list of order variables created using vars()

Usage

```
assert_order_vars(arg, optional = FALSE)
```

Arguments

arg	A function argument to be checked
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not a list of variables or desc() calls created using vars() and returns the input invisibly otherwise.

Author(s)

Stefan Bundfuss

Examples

```
example_fun <- function(by_vars) {  
  assert_order_vars(by_vars)  
}  
  
example_fun(vars(USUBJID, PARAMCD, desc(AVISITN)))  
  
try(example_fun(rlang::exprs(USUBJID, PARAMCD)))  
  
try(example_fun(c("USUBJID", "PARAMCD", "VISITN")))  
  
try(example_fun(vars(USUBJID, toupper(PARAMCD), -AVAL)))
```

assert_param_does_not_exist

Asserts That a Parameter Does Not Exist in the Dataset

Description

Checks if a parameter (PARAMCD) does not exist in a dataset.

Usage

```
assert_param_does_not_exist(dataset, param)
```

Arguments

dataset	A data.frame
param	Parameter code to check

Value

The function throws an error if the parameter exists in the input dataset. Otherwise, the dataset is returned invisibly.

Author(s)

Stefan Bundfuss

Examples

```
data(admiral_advs)
assert_param_does_not_exist(admiral_advs, param = "HR")
try(assert_param_does_not_exist(admiral_advs, param = "WEIGHT"))
```

assert_s3_class *Is an Argument an Object of a Specific S3 Class?*

Description

Checks if an argument is an object inheriting from the S3 class specified.

Usage

```
assert_s3_class(arg, class, optional = TRUE)
```

Arguments

arg	A function argument to be checked
class	The S3 class to check for
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is an object which does *not* inherit from class. Otherwise, the input is returned invisibly.

Author(s)

Thomas Neitmann

Examples

```
example_fun <- function(obj) {
  assert_s3_class(obj, "factor")
}

example_fun(as.factor(letters))

try(example_fun(letters))

try(example_fun(1:10))
```

assert_symbol	<i>Is an Argument a Symbol?</i>
---------------	---------------------------------

Description

Checks if an argument is a symbol

Usage

```
assert_symbol(arg, optional = FALSE)
```

Arguments

arg	A function argument to be checked. Must be a quosure. See examples.
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not a symbol and returns the input invisibly otherwise.

Author(s)

Thomas Neitmann

Examples

```
library(admiral.test)
data(admiral_dm)

example_fun <- function(dat, var) {
  var <- assert_symbol(rlang::enquo(var))
  dplyr::select(dat, !!var)
}

example_fun(admiral_dm, USUBJID)

try(example_fun(admiral_dm))

try(example_fun(admiral_dm, "USUBJID"))

try(example_fun(admiral_dm, toupper(PARAMCD)))
```

 assert_terms

Asserts Requirements for Terms for Queries

Description

The function checks the requirements for terms for queries provided by the user. The terms could have been provided directly in the query definition or via a user provided function for accessing a SMQ or SDG database.

Usage

```
assert_terms(
  terms,
  expect_query_name = FALSE,
  expect_query_id = FALSE,
  source_text
)
```

Arguments

terms	Terms provided by user
expect_query_name	Is the QUERY_NAME column expected?
expect_query_id	Is the QUERY_ID column expected?
source_text	Text describing the source of the terms, e.g., "the data frame provided for the definition element".

Value

An error is issued if

- terms is not a data frame,
- terms has zero observations,
- the TERM_LEVEL variable is not in terms,
- neither the TERM_NAME nor the TERM_ID variable is in terms,
- expect_query_name == TRUE and the QUERY_NAME variable is not in terms,
- expect_query_id == TRUE and the QUERY_ID variable is not in terms,

Author(s)

Stefan Bundfuss

See Also

[create_query_data\(\)](#), [query\(\)](#)

Examples

```
try(
  assert_terms(
    terms = 42,
    source_text = "object provided by the `definition` element"
  )
)
```

 assert_unit

Asserts That a Parameter is Provided in the Expected Unit

Description

Checks if a parameter (PARAMCD) in a dataset is provided in the expected unit.

Usage

```
assert_unit(dataset, param, required_unit, get_unit_expr)
```

Arguments

dataset	A data.frame
param	Parameter code of the parameter to check
required_unit	Expected unit
get_unit_expr	Expression used to provide the unit of param

Value

The function throws an error if the unit variable differs from the unit for any observation of the parameter in the input dataset. Otherwise, the dataset is returned invisibly.

Author(s)

Stefan Bundfuss

Examples

```
data(admiral_advs)
assert_unit(admiral_advs, param = "WEIGHT", required_unit = "kg", get_unit_expr = VSSTRESU)
## Not run:
assert_unit(admiral_advs, param = "WEIGHT", required_unit = "g", get_unit_expr = VSSTRESU)

## End(Not run)
```

assert_valid_queries *Verify if a Dataset Has the Required Format as Queries Dataset.*

Description

Verify if a Dataset Has the Required Format as Queries Dataset.

Usage

```
assert_valid_queries(queries, queries_name)
```

Arguments

queries	A data.frame.
queries_name	Name of the queries dataset, a string.

Details

Check if the dataset has the following columns

- VAR_PREFIX, e.g., SMQ01, CQ12
- QUERY_NAME, non NA, must be unique per each VAR_PREFIX
- QUERY_ID, could be NA, must be unique per each VAR_PREFIX
- QUERY_SCOPE, 'BROAD', 'NARROW', or NA
- QUERY_SCOPE_NUM, 1, 2, or NA
- TERM_LEVEL, e.g., "AEDECOD", "AELLT", "AELLTCD", ...
- TERM_NAME, character, could be NA only at those observations where TERM_ID is non-NA
- TERM_ID, integer, could be NA only at those observations where TERM_NAME is non-NA

Value

The function throws an error if any of the requirements not met.

Author(s)

Shimeng Huang, Ondrej Slama

Examples

```
data("queries")
assert_valid_queries(queries, "queries")
```

assert_vars	<i>Is an Argument a List of Variables?</i>
-------------	--

Description

Checks if an argument is a valid list of variables created using `vars()`

Usage

```
assert_vars(arg, optional = FALSE)
```

Arguments

arg	A function argument to be checked
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not a list of variables created using `vars()` and returns the input invisibly otherwise.

Author(s)

Samia Kabi

Examples

```
example_fun <- function(by_vars) {  
  assert_vars(by_vars)  
}  
  
example_fun(vars(USUBJID, PARAMCD))  
  
try(example_fun(rlang::exprs(USUBJID, PARAMCD)))  
  
try(example_fun(c("USUBJID", "PARAMCD", "VISIT")))  
  
try(example_fun(vars(USUBJID, toupper(PARAMCD), desc(AVAL))))
```

assert_varval_list *Is an Argument a Variable-Value List?*

Description

Checks if the argument is a list of quosures where the expressions are variable-value pairs. The value can be a symbol, a string, a numeric, or NA. More general expression are not allowed.

Usage

```
assert_varval_list(  
  arg,  
  required_elements = NULL,  
  accept_expr = FALSE,  
  accept_var = FALSE,  
  optional = FALSE  
)
```

Arguments

arg	A function argument to be checked
required_elements	A character vector of names that must be present in arg
accept_expr	Should expressions on the right hand side be accepted?
accept_var	Should unnamed variable names (e.g. vars(USUBJID)) on the right hand side be accepted?
optional	Is the checked parameter optional? If set to FALSE and arg is NULL then an error is thrown.

Value

The function throws an error if arg is not a list of variable-value expressions. Otherwise, the input is returned invisibly.

Author(s)

Stefan Bundfuss, Thomas Neitmann

Examples

```
example_fun <- function(vars) {  
  assert_varval_list(vars)  
}  
example_fun(vars(DTHDOM = "AE", DTHSEQ = AESEQ))  
  
try(example_fun(vars("AE", DTSEQ = AESEQ)))
```

call_derivation	<i>Call a Single Derivation Multiple Times</i>
-----------------	--

Description

Call a single derivation multiple times with some parameters/arguments being fixed across iterations and others varying.

Usage

```
call_derivation(dataset = NULL, derivation, variable_params, ...)
```

Arguments

dataset	The input dataset
derivation	The derivation function to call
variable_params	A list of function arguments that are different across iterations. Each set of function arguments must be created using params() .
...	Any number of <i>named</i> function arguments that stay the same across iterations. If a function argument is specified both inside <code>variable_params</code> and <code>...</code> then the value in <code>variable_params</code> overwrites the one in <code>...</code>

Value

The input dataset with additional records/variables added depending on which derivation has been used.

Author(s)

Thomas Neitmann, Stefan Bundfuss, Tracey Wang

See Also

[params\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(admiral_adsl)

adae <-
  select(admiral_ae[sample(1:nrow(admiral_ae), 1000), ], USUBJID, AESTDTC, AEENDTC) %>%
  derive_vars_merged(
    dataset_add = admiral_adsl,
    new_vars = vars(TRTSOT, TRTEDT),
```

```

    by_vars = vars(USUBJID)
  )

## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
  variable_params = list(
    params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
    params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
  ),
  min_dates = vars(TRTSDT),
  max_dates = vars(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the following
adae %>%
  derive_vars_dt(
    new_vars_prefix = "AST",
    dtc = AESTDTC,
    date_imputation = "first",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  )

```

call_user_fun

Calls a Function Provided by the User

Description

Calls a function provided by the user and adds the function call to the error message if the call fails.

Usage

```
call_user_fun(call)
```

Arguments

call Call to be executed

Value

The return value of the function call

Author(s)

Stefan Bundfuss

Examples

```
call_user_fun(compute_bmi(
  height = 172,
  weight = 60
))

try(call_user_fun(compute_bmi(
  height = 172,
  weight = "hallo"
)))
```

censor_source

Create a censor_source Object

Description

censor_source objects are used to define censorings as input for the derive_param_tte() function.

Usage

```
censor_source(
  dataset_name,
  filter = NULL,
  date,
  censor = 1,
  set_values_to = NULL
)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte().
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points.
date	A variable providing the date of the event or censoring. A date, a datetime, or a character variable containing ISO 8601 dates can be specified. An unquoted symbol is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

censor	Censoring value CDISC strongly recommends using 0 for events and positive integers for censoring.
set_values_to	A named list returned by vars() defining the variables to be set for the event or censoring, e.g. vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, or NA.

Value

An object of class censor_source, inheriting from class tte_source

Author(s)

Stefan Bundfuss

See Also

[derive_param_tte\(\)](#), [event_source\(\)](#)

Examples

```
# Last study date known alive censor
censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = vars(
    EVNTDESC = "ALIVE",
    SRCDOM = "ADSL",
    SRCVAR = "LSTALVDT"
  )
)
```

compute_bmi

Compute Body Mass Index (BMI)

Description

Computes BMI from height and weight

Usage

```
compute_bmi(height, weight)
```

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm. Permitted Values: numeric vector
weight	WEIGHT value It is expected that WEIGHT is in kg. Permitted Values: numeric vector

Details

Usually this computation function can not be used with %>%.

Value

The BMI (Body Mass Index Area) in kg/m^2 .

Author(s)

Pavan Kumar

Examples

```
compute_bmi(height = 170, weight = 75)
```

compute_bsa

Compute Body Surface Area (BSA)

Description

Computes BSA from height and weight making use of the specified derivation method

Usage

```
compute_bsa(height = height, weight = weight, method)
```

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm. Permitted Values: numeric vector
weight	WEIGHT value It is expected that WEIGHT is in kg. Permitted Values: numeric vector

method Derivation method to use:
 Mosteller: $\sqrt{\text{height} * \text{weight} / 3600}$
 DuBois-DuBois: $0.20247 * (\text{height}/100) ^ 0.725 * \text{weight} ^ 0.425$
 Haycock: $0.024265 * \text{height} ^ 0.3964 * \text{weight} ^ 0.5378$
 Gehan-George: $0.0235 * \text{height} ^ 0.42246 * \text{weight} ^ 0.51456$
 Boyd: $0.0003207 * (\text{height} ^ 0.3) * (1000 * \text{weight}) ^ (0.7285 - (0.0188 * \log_{10}(1000 * \text{weight})))$
 Fujimoto: $0.008883 * \text{height} ^ 0.663 * \text{weight} ^ 0.444$
 Takahira: $0.007241 * \text{height} ^ 0.725 * \text{weight} ^ 0.425$
 Permitted Values: character value

Details

Usually this computation function can not be used with %>%.

Value

The BSA (Body Surface Area) in m².

Author(s)

Eric Simms

Examples

```
# Derive BSA by the Mosteller method
compute_bsa(
  height = 170,
  weight = 75,
  method = "Mosteller"
)

# Derive BSA by the DuBois & DuBois method
compute_bsa(
  height = c(170, 185),
  weight = c(75, 90),
  method = "DuBois-DuBois"
)
```

compute_dtf

Derive the Date Imputation Flag

Description

Derive the date imputation flag ('--DTF') comparing a date character vector ('--DTC') with a Date vector ('--DT').

Usage

```
compute_dtf(dtc, dt)
```

Arguments

dtc	The date character vector ('--DTC'). A character date is expected in a format like yyyy-mm-ddThh:mm:ss (partial or complete).
dt	The Date vector to compare. A date object is expected.

Details

Usually this computation function can not be used with %>%.

Value

The date imputation flag ('--DTF') (character value of 'D', 'M', 'Y' or NA)

Author(s)

Samia Kabi

Examples

```
compute_dtf(dtc = "2019-07", dt = as.Date("2019-07-18"))  
compute_dtf(dtc = "2019", dt = as.Date("2019-07-18"))
```

compute_duration	<i>Compute Duration</i>
------------------	-------------------------

Description

Compute duration between two dates, e.g., duration of an adverse event, relative day, age, ...

Usage

```
compute_duration(  
  start_date,  
  end_date,  
  in_unit = "days",  
  out_unit = "days",  
  floor_in = TRUE,  
  add_one = TRUE,  
  trunc_out = FALSE  
)
```

Arguments

start_date	<p>The start date</p> <p>A date or date-time object is expected.</p> <p>Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.</p>
end_date	<p>The end date</p> <p>A date or date-time object is expected.</p> <p>Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.</p>
in_unit	<p>Input unit</p> <p>See <code>floor_in</code> and <code>add_one</code> parameter for details.</p> <p>Default: 'days'</p> <p>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'</p>
out_unit	<p>Output unit</p> <p>The duration is derived in the specified unit</p> <p>Default: 'days'</p> <p>Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'</p>
floor_in	<p>Round down input dates?</p> <p>The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.</p> <p>Default: 'TRUE'</p> <p>Permitted Values: TRUE, FALSE</p>
add_one	<p>Add one input unit?</p> <p>If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.</p> <p>Default: TRUE</p> <p>Permitted Values: TRUE, FALSE</p>
trunc_out	<p>Return integer part</p> <p>The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.</p> <p>Default: FALSE</p> <p>Permitted Values: TRUE, FALSE</p>

Details

The output is a numeric vector providing the duration as time from start to end date in the specified unit. If the end date is before the start date, the duration is negative.

Value

The duration between the two date in the specified unit

Author(s)

Stefan Bundfuss

Examples

```
# derive duration in days (integer), i.e., relative day
compute_duration(
  start_date = lubridate::ymd_hms("2020-12-06T15:00:00"),
  end_date = lubridate::ymd_hms("2020-12-24T08:15:00")
)

# derive duration in days (float)
compute_duration(
  start_date = lubridate::ymd_hms("2020-12-06T15:00:00"),
  end_date = lubridate::ymd_hms("2020-12-24T08:15:00"),
  floor_in = FALSE,
  add_one = FALSE
)

# derive age
compute_duration(
  start_date = lubridate::ymd("1984-09-06"),
  end_date = lubridate::ymd("2020-02-24"),
  trunc_out = TRUE,
  out_unit = "years",
  add_one = FALSE
)
```

`compute_map`*Compute Mean Arterial Pressure (MAP)*

Description

Computes mean arterial pressure (MAP) based on diastolic and systolic blood pressure. Optionally heart rate can be used as well.

Usage

```
compute_map(diabp, sysbp, hr = NULL)
```

Arguments

<code>diabp</code>	Diastolic blood pressure A numeric vector is expected.
<code>sysbp</code>	Systolic blood pressure A numeric vector is expected.
<code>hr</code>	Heart rate A numeric vector or NULL is expected.

Details

$$\frac{2DIABP + SYSBP}{3}$$

if it is based on diastolic and systolic blood pressure and

$$DIABP + 0.01e^{4.14 - \frac{40.74}{HR}} (SYSBP - DIABP)$$

if it is based on diastolic, systolic blood pressure, and heart rate.

Usually this computation function can not be used with %>%.

Value

A numeric vector of MAP values

Author(s)

Stefan Bundfuss

Examples

```
# Compute MAP based on diastolic and systolic blood pressure
compute_map(diabp = 51, sysbp = 121)
```

```
# Compute MAP based on diastolic and systolic blood pressure and heart rate
compute_map(diabp = 51, sysbp = 121, hr = 59)
```

compute_qtc

Compute Corrected QT

Description

Computes corrected QT using Bazett's, Fridericia's or Sagie's formula.

Usage

```
compute_qtc(qt, rr, method)
```

Arguments

qt	QT interval A numeric vector is expected. It is expected that QT is measured in msec.
rr	RR interval A numeric vector is expected. It is expected that RR is measured in msec.
method	Method used to QT correction Permitted Values: "Bazett", "Fridericia", "Sagie"

Details

Depending on the chosen method one of the following formulae is used.

Bazett:

$$\frac{QT}{\sqrt{\frac{RR}{1000}}}$$

Fridericia:

$$\frac{QT}{\sqrt[3]{\frac{RR}{1000}}}$$

Sagie:

$$1000 \left(\frac{QT}{1000} + 0.154 \left(1 - \frac{RR}{1000} \right) \right)$$

Usually this computation function can not be used with %>%.

Value

QT interval in msec

Author(s)

Stefan Bundfuss

Examples

```
compute_qtc(qt = 350, rr = 56.54, method = "Bazett")
```

```
compute_qtc(qt = 350, rr = 56.54, method = "Fridericia")
```

```
compute_qtc(qt = 350, rr = 56.54, method = "Sagie")
```

compute_rr

Compute RR Interval From Heart Rate

Description

Computes RR interval from heart rate.

Usage

```
compute_rr(hr)
```

Arguments

hr Heart rate
A numeric vector is expected. It is expected that heart rate is measured in beats/min.

Details

Usually this computation function can not be used with %>%.

Value

RR interval in msec:
$$\frac{60000}{HR}$$

Author(s)

Stefan Bundfuss

Examples

```
compute_rr(hr = 70.14)
```

compute_tmf	<i>Derive the Time Imputation Flag</i>
-------------	--

Description

Derive the time imputation flag ('--TMF') comparing a date character vector ('--DTC') with a Datetime vector ('--DTM').

Usage

```
compute_tmf(dtc, dtm, ignore_seconds_flag = FALSE)
```

Arguments

dtc	The date character vector ('--DTC'). A character date is expected in a format like yyyy-mm-ddThh:mm:ss (partial or complete).
dtm	The Date vector to compare ('--DTM'). A datetime object is expected.
ignore_seconds_flag	ADaM IG states that given SDTM ('--DTC') variable, if only hours and minutes are ever collected, and seconds are imputed in ('--DTM') as 00, then it is not necessary to set ('--TMF') to 'S'. A user can set this to TRUE so the 'S' Flag is dropped from ('--TMF'). A logical value Default: FALSE

Details

Usually this computation function can not be used with %>%.

Value

The time imputation flag ('--TMF') (character value of 'H', 'M', 'S' or NA)

Author(s)

Samia Kabi

Examples

```
compute_tmf(dtc = "2019-07-18T15:25", dtm = as.POSIXct("2019-07-18T15:25:00"))
compute_tmf(dtc = "2019-07-18T15", dtm = as.POSIXct("2019-07-18T15:25:00"))
compute_tmf(dtc = "2019-07-18", dtm = as.POSIXct("2019-07-18"))
```

convert_blanks_to_na *Convert Blank Strings Into NAs*

Description

Turn SAS blank strings into proper R NAs.

Usage

```
convert_blanks_to_na(x)

## Default S3 method:
convert_blanks_to_na(x)

## S3 method for class 'character'
convert_blanks_to_na(x)

## S3 method for class 'list'
convert_blanks_to_na(x)

## S3 method for class 'data.frame'
convert_blanks_to_na(x)
```

Arguments

x Any R object

Details

The default methods simply returns its input unchanged. The character method turns every instance of "" into NA_character_ while preserving *all* attributes. When given a data frame as input the function keeps all non-character columns as is and applies the just described logic to character columns. Once again all attributes such as labels are preserved.

Value

An object of the same class as the input

Author(s)

Thomas Neitmann

Examples

```
convert_blanks_to_na(c("a", "b", "", "d", ""))

df <- tibble::tibble(
  a = structure(c("a", "b", "", "c"), label = "A"),
  b = structure(c(1, NA, 21, 9), label = "B"),
  c = structure(c(TRUE, FALSE, TRUE, TRUE), label = "C"),
  d = structure(c("", "", "s", "q"), label = "D")
)
print(df)
convert_blanks_to_na(df)
```

convert_date_to_dtm *Convert a Date into a Datetime Object*

Description

Convert a date (datetime, date, or date character) into a Date vector (usually '--DTM').

Usage

```
convert_date_to_dtm(
  dt,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dt The date to convert.
A date or character date is expected in a format like yyyy-mm-ddThh:mm:ss.

date_imputation The value to impute the day/month when a datepart is missing.
If NULL: no date imputation is performed and partial dates are returned as missing.
Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,
- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

time_imputation
The value to impute the time when a timepart is missing.
A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "FIRST", "LAST" to impute to the start/end of a day.

Default is "00:00:00".

min_dates
Minimum dates
A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  date_imputation = "first"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

max_dates
Maximum dates
A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

preserve
Preserve day if month is missing and day is present
For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").
Permitted Values: TRUE, FALSE
Default: FALSE

Details

Usually this computation function can not be used with %>%.

Value

A datetime object

Author(s)

Samia Kabi

Examples

```
convert_date_to_dtm("2019-07-18T15:25:00")
convert_date_to_dtm(Sys.time())
convert_date_to_dtm(as.Date("2019-07-18"), time_imputation = "23:59:59")
convert_date_to_dtm("2019-07-18", time_imputation = "23:59:59")
convert_date_to_dtm("2019-07-18")
```

convert_dtc_to_dt *Convert a Date Character Vector into a Date Object*

Description

Convert a date character vector (usually '-DTC') into a Date vector (usually '-DT').

Usage

```
convert_dtc_to_dt(
  dtc,
  date_imputation = NULL,
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dtc The -DTC date to convert.
A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. A partial date will return a NA date and a warning will be issued: 'All formats failed to parse. No formats found.'. Note: you can use impute_dtc function to build a complete date.

date_imputation
The value to impute the day/month when a datepart is missing.
If NULL: no date imputation is performed and partial dates are returned as missing.
Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,

- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

min_dates

Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  date_imputation = "first"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

max_dates

Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

preserve

Preserve day if month is missing and day is present

For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").

Permitted Values: TRUE, FALSE

Default: FALSE

Details

Usually this computation function can not be used with %>%.

Value

a date object

Author(s)

Samia Kabi

Examples

```
convert_dtc_to_dt("2019-07-18")
convert_dtc_to_dt("2019-07")
```

convert_dtc_to_dtm *Convert a Date Character Vector into a Datetime Object*

Description

Convert a date character vector (usually '--DTC') into a Date vector (usually '--DTM').

Usage

```
convert_dtc_to_dtm(
  dtc,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dtc The '--DTC' date to convert.
A character date is expected in a format like yyyy-mm-ddThh:mm:ss. A partial datetime will issue a warning. Note: you can use [impute_dtc\(\)](#) function to build a complete datetime.

date_imputation
The value to impute the day/month when a datepart is missing.
If NULL: no date imputation is performed and partial dates are returned as missing.
Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,
- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

time_imputation
The value to impute the time when a timepart is missing.
A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "FIRST", "LAST" to impute to the start/end of a day.

	Default is "00:00:00".
min_dates	<p>Minimum dates</p> <p>A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example</p> <pre> impute_dtc("2020-11", min_dates = list(ymd_hms("2020-12-06T12:12:12"), ymd_hms("2020-11-11T11:11:11")), date_imputation = "first") </pre> <p>returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).</p>
max_dates	<p>Maximum dates</p> <p>A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.</p>
preserve	<p>Preserve day if month is missing and day is present</p> <p>For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").</p> <p>Permitted Values: TRUE, FALSE</p> <p>Default: FALSE</p>

Details

Usually this computation function can not be used with %>%.

Value

A datetime object

Author(s)

Samia Kabi

Examples

```

convert_dtc_to_dtm("2019-07-18T15:25:00")
convert_dtc_to_dtm("2019-07-18T00:00:00") # note Time = 00:00:00 is not printed
convert_dtc_to_dtm("2019-07-18")

```

create_query_data	<i>Creates a queries dataset as input dataset to the dataset_queries argument in derive_vars_query()</i>
-------------------	--

Description

Creates a queries dataset as input dataset to the dataset_queries argument in the derive_vars_query() function as defined in the [Queries Dataset Documentation](#).

Usage

```

create_query_data(
  queries,
  meddra_version = NULL,
  whodd_version = NULL,
  get_smq_fun = NULL,
  get_sdg_fun = NULL
)

```

Arguments

queries	List of queries A list of query() objects is expected.
meddra_version	MedDRA version The MedDRA version used for coding the terms in the AE dataset should be specified. If any of the queries is a SMQ or a customized query including a SMQ, the parameter needs to be specified. <i>Permitted Values:</i> A character string (the expected format is company-specific)
whodd_version	WHO Drug Dictionary version The version of the WHO Drug Dictionary used for coding the terms in the CM dataset should be specified. If any of the queries is a SDG, the parameter needs to be specified. <i>Permitted Values:</i> A character string (the expected format is company-specific)
get_smq_fun	Function which returns the terms of an SMQ For each query specified for the queries parameter which refers to an SMQ (i.e., those where the definition field is set to a smq_select() object or a list which contains at least one smq_select() object) the specified function is called to retrieve the terms defining the query. This function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level. The function must return a dataset with all the terms defining the SMQ. The output dataset must contain the following variables.

- TERM_LEVEL: the variable to be used for defining a term of the SMQ, e.g., AEDECOD
- TERM_NAME: the name of the term if the variable TERM_LEVEL is referring to is character
- TERM_ID the numeric id of the term if the variable TERM_LEVEL is referring to is numeric
- QUERY_NAME: the name of the SMQ. The values must be the same for all observations.

The function must provide the following parameters

- smq_select: A smq_select() object.
- version: The MedDRA version. The value specified for the meddra_version in the create_query_data() call is passed to this parameter.
- keep_id: If set to TRUE, the output dataset must contain the QUERY_ID variable. The variable must be set to the numeric id of the SMQ.
- temp_env: A temporary environment is passed to this parameter. It can be used to store data which is used for all SMQs in the create_query_data() call. For example if the SMQs need to be read from a database all SMQs can be read and stored in the environment when the first SMQ is handled. For the other SMQs the terms can be retrieved from the environment instead of accessing the database again.

get_sdg_fun

Function which returns the terms of an SDG

For each query specified for the queries parameter which refers to an SDG the specified function is called to retrieve the terms defining the query. This function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level.

The function must return a dataset with all the terms defining the SDG. The output dataset must contain the following variables.

- TERM_LEVEL: the variable to be used for defining a term of the SDG, e.g., CMDECOD
- TERM_NAME: the name of the term if the variable TERM_LEVEL is referring to is character
- TERM_ID the numeric id of the term if the variable TERM_LEVEL is referring to is numeric
- QUERY_NAME: the name of the SDG. The values must be the same for all observations.

The function must provide the following parameters

- sdg_select: A sdg_select() object.
- version: The WHO drug dictionary version. The value specified for the whodd_version in the create_query_data() call is passed to this parameter.
- keep_id: If set to TRUE, the output dataset must contain the QUERY_ID variable. The variable must be set to the numeric id of the SDG.
- temp_env: A temporary environment is passed to this parameter. It can be used to store data which is used for all SDGs in the create_query_data() call. For example if the SDGs need to be read from a database all SDGs can

be read and stored in the environment when the first SDG is handled. For the other SDGs the terms can be retrieved from the environment instead of accessing the database again.

Details

For each `query()` object listed in the `queries` argument, the terms belonging to the query (`TERM_LEVEL`, `TERM_NAME`, `TERM_ID`) are determined with respect to the `definition` field of the query: if the `definition` field of the `query()` object is

- an `smq_select()` object, the terms are read from the SMQ database by calling the function specified for the `get_smq_fun` parameter.
- an `sdg_select()` object, the terms are read from the SDG database by calling the function specified for the `get_sdg_fun` parameter.
- a data frame, the terms stored in the data frame are used.
- a list of data frames and `smq_select()` objects, all terms from the data frames and all terms read from the SMQ database referenced by the `smq_select()` objects are collated.

The following variables (as described in [Queries Dataset Documentation](#)) are created:

- `VAR_PREFIX`: Prefix of the variables to be created by `derive_vars_query()` as specified by the `prefix` element.
- `QUERY_NAME`: Name of the query as specified by the `name` element.
- `QUERY_ID`: Id of the query as specified by the `id` element. If the `id` element is not specified for a query, the variable is set to NA. If the `id` element is not specified for any query, the variable is not created.
- `QUERY_SCOPE`: scope of the query as specified by the `scope` element of the `smq_select()` object. For queries not defined by a `smq_select()` object, the variable is set to NA. If none of the queries is defined by a `smq_select()` object, the variable is not created.
- `QUERY_SCOPE_NUM`: numeric scope of the query. It is set to 1 if the scope is broad. Otherwise it is set to '2'. If the `add_scope_num` element equals `FALSE`, the variable is set to NA. If the `add_scope_num` element equals `FALSE` for all SMQs or none of the queries is an SMQ, the variable is not created.
- `TERM_LEVEL`: Name of the variable used to identify the terms.
- `TERM_NAME`: Value of the term variable if it is a character variable.
- `TERM_ID`: Value of the term variable if it is a numeric variable.

Value

A dataset to be used as input dataset to the `dataset_queries` argument in `derive_vars_query()`

Author(s)

Stefan Bundfuss

See Also

[derive_vars_query\(\)](#), [query\(\)](#), [smq_select\(\)](#), [sdg_select\(\)](#), [Queries Dataset Documentation](#)

Examples

```

library(tibble)
library(magrittr, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
library(admiral)

# creating a query dataset for a customized query
cqterms <- tribble(
  ~TERM_NAME, ~TERM_ID,
  "APPLICATION SITE ERYTHEMA", 10003041L,
  "APPLICATION SITE PRURITUS", 10003053L
) %>%
  mutate(TERM_LEVEL = "AEDECOD")

cq <- query(
  prefix = "CQ01",
  name = "Application Site Issues",
  definition = cqterms
)

create_query_data(queries = list(cq))

# create a query dataset for SMQs
pregsmq <- query(
  prefix = "SMQ02",
  id = auto,
  definition = smq_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW"
  )
)

bilismq <- query(
  prefix = "SMQ04",
  definition = smq_select(
    id = 20000121L,
    scope = "BROAD"
  )
)

# The get_smq_terms function from admiral.test is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(pregsmq, bilismq),
  get_smq_fun = admiral.test::get_smq_terms,
  meddra_version = "20.1"
)

# create a query dataset for SDGs
sdg <- query(
  prefix = "SDG01",

```

```

    id = auto,
    definition = sdg_select(
      name = "5-aminosalicylates for ulcerative colitis"
    )
  )

# The get_sdg_terms function from admiral.test is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(sdg),
  get_sdg_fun = admiral.test::get_sdg_terms,
  whodd_version = "2019-09"
)

# creating a query dataset for a customized query including SMQs
# The get_smq_terms function from admiral.test is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(
    query(
      prefix = "CQ03",
      name = "Special issues of interest",
      definition = list(
        smq_select(
          name = "Pregnancy and neonatal topics (SMQ)",
          scope = "NARROW"
        ),
        cqterms
      )
    )
  ),
  get_smq_fun = admiral.test::get_smq_terms,
  meddra_version = "20.1"
)

```

```
create_single_dose_dataset
```

Create dataset of single doses

Description

Derives dataset of single dose from aggregate dose information. This may be necessary when e.g. calculating last dose before an adverse event in ADAE or deriving a total dose parameter in ADEX when EXDOSFRQ != ONCE.

Usage

```
create_single_dose_dataset(
  dataset,
  dose_freq = EXDOSFRQ,

```

```

start_date = ASTDT,
end_date = AENDT,
lookup_table = dose_freq_lookup,
lookup_column = CDISC_VALUE
)

```

Arguments

dataset	<p>Input dataset</p> <p>The columns specified by dose_freq, start_date and the end_date parameters are expected.</p>
dose_freq	<p>The dose frequency</p> <p>The aggregate dosing frequency used for multiple doses in a row.</p> <p>Default: EXDOSFRQ</p> <p>Permitted Values: defined by lookup table.</p>
start_date	<p>The start date</p> <p>A date or date-time object is expected. This object cannot contain NA values. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.</p> <p>Default: ASTDT</p>
end_date	<p>The end date</p> <p>A date or date-time object is expected. This object cannot contain NA values. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.</p> <p>Default: AENDT</p>
lookup_table	<p>The dose frequency value lookup table</p> <p>The table used to look up dose_freq values and determine the appropriate multiplier to be used for row generation. If a lookup table other than the default is used, it must have columns DOSE_WINDOW, DOSE_COUNT, and CONVERSION_FACTOR. The default table dose_freq_lookup is described in detail here.</p> <p>Default: dose_freq_lookup</p> <p>Permitted Values for DOSE_WINDOW: "MINUTE", "HOUR", "DAY", "WEEK", "MONTH", "YEAR"</p>
lookup_column	<p>The dose frequency value column in the lookup table</p> <p>The column of lookup_table.</p> <p>Default: CDISC_VALUE (column of dose_freq_lookup)</p>

Details

Each aggregate dose row is split into multiple rows which each represent a single dose. The number of completed dose periods between start_date and end_date is calculated with compute_duration and multiplied by DOSE_COUNT. For DOSE_WINDOW values of "WEEK", "MONTH", and "YEAR", CONVERSION_FACTOR is used to convert into days the time object to be added to start_date.

Value

The input dataset with a single dose per row.

Author(s)

Michael Thorpe, Andrew Smith

Examples

```
# Example with default lookup

library(lubridate)

data <- tibble::tribble(
  ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~AENDT,
  "P01", "Q2D", ymd("2021-01-01"), ymd("2021-01-07"),
  "P01", "Q3D", ymd("2021-01-08"), ymd("2021-01-15"),
  "P01", "EVERY 2 WEEKS", ymd("2021-01-15"), ymd("2021-01-29")
)

create_single_dose_dataset(data)

# Example with custom lookup

custom_lookup <- tibble::tribble(
  ~Value, ~DOSE_COUNT, ~DOSE_WINDOW, ~CONVERSION_FACTOR,
  "Q30MIN", (1 / 30), "MINUTE", 1,
  "Q90MIN", (1 / 90), "MINUTE", 1
)

data <- tibble::tribble(
  ~USUBJID, ~EXDOSFRQ, ~ASTDTM, ~AENDTM,
  "P01", "Q30MIN", ymd_hms("2021-01-01T06:00:00"), ymd_hms("2021-01-01T07:00:00"),
  "P02", "Q90MIN", ymd_hms("2021-01-01T06:00:00"), ymd_hms("2021-01-01T09:00:00")
)

create_single_dose_dataset(data,
  lookup_table = custom_lookup,
  lookup_column = Value,
  start_date = ASTDTM,
  end_date = AENDTM
)
```

dataset_vignette

Output a Dataset in a Vignette in the admiral Format

Description

Output a dataset in a vignette with the pre-specified admiral format.

Usage

```
dataset_vignette(dataset, display_vars = NULL, filter = NULL)
```

Arguments

dataset	Dataset to output in the vignette
display_vars	Variables selected to demonstrate the outcome of the derivation Permitted Values: list of variables Default is NULL If display_vars is not NULL, only the selected variables are visible in the vignette while the other variables are hidden. They can be made visible by clicking the Choose the columns to display button.
filter	Filter condition The specified condition is applied to the dataset before it is displayed. Permitted Values: a condition

Value

A HTML table

Examples

```
library(admiral)
library(DT)
library(dplyr)
library(admiral.test)
data("admiral_dm")

dataset_vignette(admiral_dm)
dataset_vignette(admiral_dm,
  display_vars = vars(USUBJID, RFSTDTC, DTHDTC),
  filter = ARMCD == "Pbo"
)
```

date_source

Create a date_source object

Description

Create a date_source object as input for derive_var_extreme_dt() and derive_var_extreme_dtm().

Usage

```
date_source(  
  dataset_name,  
  filter = NULL,  
  date,  
  date_imputation = NULL,  
  time_imputation = NULL,  
  preserve = FALSE,  
  traceability_vars = NULL  
)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the date.
filter	An unquoted condition for filtering dataset.
date	A variable providing a date. A date, a datetime, or a character variable containing ISO 8601 dates can be specified. An unquoted symbol is expected.
date_imputation	A string defining the date imputation for date. See date_imputation parameter of impute_dtc() for valid values.
time_imputation	A string defining the time imputation for date. See time_imputation parameter of impute_dtc() for valid values.
preserve	Should day be preserved if month is imputed for date. See preserve parameter of impute_dtc() for details.
traceability_vars	A named list returned by vars() defining the traceability variables, e.g. vars(LALVDOM = "AE", LALVSEQ = AESEQ, LALVVAR = "AESTDTC"). The values must be a symbol, a character string, a numeric, or NA.

Value

An object of class date_source.

Author(s)

Stefan Bundfuss

See Also

[derive_var_extreme_dtm\(\)](#), [derive_var_extreme_dt\(\)](#)

`death_event`*Pre-Defined Time-to-Event Source Objects*

Description

These pre-defined `tte_source` objects can be used as input to `derive_param_tte()`.

Usage`death_event``lastalive_censor``ae_event``ae_ser_event``ae_gr1_event``ae_gr2_event``ae_gr3_event``ae_gr4_event``ae_gr5_event``ae_gr35_event``ae_sev_event``ae_wd_event`**Format**

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `censor_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class `event_source` (inherits from `tte_source`, `source`, `list`) of length 5.

An object of class event_source (inherits from tte_source, source, list) of length 5.

An object of class event_source (inherits from tte_source, source, list) of length 5.

An object of class event_source (inherits from tte_source, source, list) of length 5.

Details

To see the definition of the various objects simply print the object in the R console, e.g. `print(death_event)`.

See Also

[derive_param_tte\(\)](#), [tte_source\(\)](#), [event_source\(\)](#), [censor_source\(\)](#)

default_qtc_paramcd *Get Default Parameter Code for Corrected QT*

Description

Get Default Parameter Code for Corrected QT

Usage

```
default_qtc_paramcd(method)
```

Arguments

method	Method used to QT correction Permitted Values: "Bazett", "Fridericia", "Sagie"
--------	---

Value

"QTCBR" if method is "Bazett", "QTCFR" if it's "Fridericia" or "QTLCR" if it's "Sagie". An error otherwise.

Author(s)

Thomas Neitmann

Examples

```
default_qtc_paramcd("Sagie")
```

derivation_slice *Create a derivation_slice Object*

Description

Create a derivation_slice object as input for slice_derivation().

Usage

```
derivation_slice(filter, args)
```

Arguments

filter	An unquoted condition for defining the observations of the slice
args	Arguments of the derivation to be used for the slice A params() object is expected.

Value

An object of class derivation_slice
An object of class slice.

Author(s)

Stefan Bundfuss

See Also

[slice_derivation\(\)](#), [params\(\)](#)

derive_derived_param *Adds a Parameter Computed from the Analysis Value of Other Parameters*

Description

Adds a parameter computed from the analysis value of other parameters. It is expected that the analysis value of the new parameter is defined by an expression using the analysis values of other parameters. For example mean arterial pressure (MAP) can be derived from systolic (SYSBP) and diastolic blood pressure (DIABP) with the formula

$$MAP = \frac{SYSBP + 2DIABP}{3}$$

Usage

```

derive_derived_param(
  dataset,
  by_vars,
  parameters,
  analysis_value,
  set_values_to,
  filter = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code>, and <code>AVAL</code> are expected.</p> <p>The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>parameters</code>.</p>
by_vars	<p>Grouping variables</p> <p>For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records.</p> <p><i>Permitted Values:</i> list of variables</p>
parameters	<p>Required parameter codes</p> <p>It is expected that all parameter codes (<code>PARAMCD</code>) which are required to derive the new parameter are specified for this parameter or the <code>constant_parameters</code> parameter.</p> <p><i>Permitted Values:</i> A character vector of <code>PARAMCD</code> values</p>
analysis_value	<p>Definition of the analysis value</p> <p>An expression defining the analysis value (<code>AVAL</code>) of the new parameter is expected. The analysis values of the parameters specified by <code>parameters</code> can be accessed using <code>AVAL.<parameter code></code>, e.g., <code>AVAL.SYSBP</code>.</p> <p><i>Permitted Values:</i> An unquoted expression</p>
set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter.</p> <p><i>Permitted Values:</i> List of variable-value pairs</p>
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

constant_by_vars

By variables for constant parameters

The constant parameters (parameters that are measured only once) are merged to the other parameters using the specified variables. (Refer to Example 2)

Permitted Values: list of variables

constant_parameters

Required constant parameter codes

It is expected that all the parameter codes (PARAMCD) which are required to derive the new parameter and are measured only once are specified here. For example if BMI should be derived and height is measured only once while weight is measured at each visit. Height could be specified in the constant_parameters parameter. (Refer to Example 2)

Permitted Values: A character vector of PARAMCD values

Details

For each group (with respect to the variables specified for the `by_vars` parameter) an observation is added to the output dataset if the filtered input dataset contains exactly one observation for each parameter code specified for parameters.

For the new observations AVAL is set to the value specified by `analysis_value` and the variables specified for `set_values_to` are set to the provided values. The values of the other variables of the input dataset are set to NA.

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Stefan Bundfuss

Examples

```
# Example 1: Derive MAP
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 51, "mmHg", "BASELINE",
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 50, "mmHg", "WEEK 2",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "mmHg", "BASELINE",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "mmHg", "WEEK 2",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 79, "mmHg", "BASELINE",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 80, "mmHg", "WEEK 2",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 130, "mmHg", "BASELINE",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 132, "mmHg", "WEEK 2"
)

derive_derived_param(
  advs,
  by_vars = vars(USUBJID, VISIT),
```

```

parameters = c("SYSBP", "DIABP"),
analysis_value = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
set_values_to = vars(
  PARAMCD = "MAP",
  PARAM = "Mean Arterial Pressure (mmHg)",
  AVALU = "mmHg"
)
)

# Example 2: Derive BMI where height is measured only once
adv <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 147, "cm", "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.0, "kg", "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.4, "kg", "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 53.1, "kg", "WEEK 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 163, "cm", "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 78.5, "kg", "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.3, "kg", "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.7, "kg", "WEEK 2"
)

derive_derived_param(
  adv,
  by_vars = vars(USUBJID, VISIT),
  parameters = "WEIGHT",
  analysis_value = AVAL.WEIGHT / (AVAL.HEIGHT / 100)^2,
  set_values_to = vars(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)",
    AVALU = "kg/m^2"
  ),
  constant_parameters = c("HEIGHT"),
  constant_by_vars = vars(USUBJID)
)

```

```
derive_extreme_records
```

Add the First or Last Observation for Each By Group as New Records

Description

Add the first or last observation for each by group as new observations. It can be used for example for adding the maximum or minimum value as a separate visit. All variables of the selected observation are kept. This distinguishes `derive_extreme_records()` from `derive_summary_records()`, where only the by variables are populated for the new records.

Usage

```
derive_extreme_records(
```

```

dataset,
by_vars = NULL,
order,
mode,
check_type = "warning",
filter = NULL,
set_values_to
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the order and the by_vars parameter are expected.</p>
by_vars	<p>Grouping variables</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> list of variables created by vars()</p>
order	<p>Sort order</p> <p>Within each by group the observations are ordered by the specified order.</p> <p><i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL))</p>
mode	<p>Selection mode (first or last)</p> <p>If "first" is specified, the first observation of each by group is added to the input dataset. If "last" is specified, the last observation of each by group is added to the input dataset.</p> <p><i>Permitted Values:</i> "first", "last"</p>
check_type	<p>Check uniqueness?</p> <p>If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.</p> <p><i>Default:</i> "warning"</p> <p><i>Permitted Values:</i> "none", "warning", "error"</p>
filter	<p>Filter for observations to consider</p> <p>Only observations fulfilling the specified condition are taken into account for selecting the first or last observation. If the parameter is not specified, all observations are considered.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> a condition</p>
set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. A list of variable name-value pairs is expected.</p> <ul style="list-style-type: none"> • LHS refers to a variable. • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or NA, e.g., vars(PARAMCD = "TDOSE", PARCAT1 = "OVERALL"). More general expression are not allowed.

Details

1. The input dataset is restricted as specified by the `filter` parameter.
2. For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is selected.
3. The variables specified by the `set_values_to` parameter are added to the selected observations.
4. The observations are added to input dataset.

Value

The input dataset with the first or last observation of each by group added as new observations.

Author(s)

Stefan Bundfuss

Examples

```
adlb <- tibble::tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~LBSEQ,
  "1",      1,        113,    1,
  "1",      2,        113,    2,
  "1",      3,        117,    3,
  "2",      1,        101,    1,
  "2",      2,        101,    2,
  "2",      3,         95,    3
)

# Add a new record for each USUBJID storing the minimum value (first AVAL).
# If multiple records meet the minimum criterion, take the first value by
# AVISITN. Set AVISITN = 97 and DTYPE = MINIMUM for these new records.
derive_extreme_records(
  adlb,
  by_vars = vars(USUBJID),
  order = vars(AVAL, AVISITN),
  mode = "first",
  filter = !is.na(AVAL),
  set_values_to = vars(
    AVISITN = 97,
    DTYPE = "MINIMUM"
  )
)

# Add a new record for each USUBJID storing the maximum value (last AVAL).
# If multiple records meet the maximum criterion, take the first value by
# AVISITN. Set AVISITN = 98 and DTYPE = MAXIMUM for these new records.
derive_extreme_records(
  adlb,
  by_vars = vars(USUBJID),
```

```

order = vars(desc(AVAL), AVISITN),
mode = "first",
filter = !is.na(AVAL),
set_values_to = vars(
  AVISITN = 98,
  DTYPE = "MAXIMUM"
)
)
)

# Add a new record for each USUBJID storing for the last value.
# Set AVISITN = 99 and DTYPE = LOV for these new records.
derive_extreme_records(
  adlb,
  by_vars = vars(USUBJID),
  order = vars(AVISITN),
  mode = "last",
  set_values_to = vars(
    AVISITN = 99,
    DTYPE = "LOV"
  )
)
)

```

derive_last_dose

Derive Last Dose Date(-time)

Description

[Deprecated]

Deprecated, please use `derive_var_last_dose_date()` instead.

Usage

```

derive_last_dose(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_start,
  dose_end,
  analysis_date,
  dataset_seq_var,
  new_var,
  output_datetime = TRUE,
  check_dates_only = FALSE,
  traceability_vars = NULL
)

```

Arguments

dataset	Input dataset.
dataset_ex	Input EX dataset.
filter_ex	Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL.
by_vars	Variables to join by (created by <code>dplyr::vars</code>).
dose_start	The dose start date variable.
dose_end	The dose end date variable.
analysis_date	The analysis date variable.
dataset_seq_var	The sequence variable (this together with <code>by_vars</code> creates the keys of dataset).
new_var	The output variable.
output_datetime	Logical. Should only date or date-time variable be returned? Defaults to TRUE (i.e. date-time variable).
check_dates_only	Logical. An assumption that start and end dates of treatment match is checked. By default (FALSE), the date as well as the time component is checked. If set to TRUE, then only the date component of those variables is checked.
traceability_vars	A named list returned by <code>vars()</code> listing the traceability variables, e.g. <code>vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ)</code> . The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Details

All date (date-time) variables can be characters in standard ISO format or of date / date-time class. For ISO format, see `impute_dtc` - parameter `dtc` for further details. Date-time imputations are done as follows:

- `dose_end`: no date imputation, time imputation to `00:00:00` if time is missing.
- `analysis_date`: no date imputation, time imputation to `23:59:59` if time is missing.

The last dose date is derived as follows:

1. The `dataset_ex` is filtered using `filter_ex`, if provided. This is useful for, for example, filtering for valid dose only.
2. The datasets `dataset` and `dataset_ex` are joined using `by_vars`.
3. The last dose date is derived: the last dose date is the maximum date where `dose_end` is lower to or equal to `analysis_date`, subject to both date values are non-NA. The last dose date is derived per `by_vars` and `dataset_seq_var`.
4. The last dose date is appended to the `dataset` and returned to the user.

Furthermore, the following assumption is checked: start and end dates (datetimes) need to match. Use `check_dates_only` to control whether only dates or whole date-times need to be equal.

Value

Input dataset with additional column new_var.

Author(s)

Ondrej Slama

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(ex_single)

admiral_ae %>%
  head(100) %>%
  derive_last_dose(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    dose_start = EXSTDTC,
    dose_end = EXENDTC,
    analysis_date = AESTDTC,
    dataset_seq_var = AESEQ,
    new_var = LDOSEDTM,
    output_datetime = TRUE,
    check_dates_only = FALSE
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDTM)

# or with traceability variables
admiral_ae %>%
  head(100) %>%
  derive_last_dose(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    dose_start = EXSTDTC,
    dose_end = EXENDTC,
    analysis_date = AESTDTC,
    dataset_seq_var = AESEQ,
    new_var = LDOSEDTM,
    output_datetime = TRUE,
    check_dates_only = FALSE,
    traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXSTDTC")
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDTM, LDOSEDOM, LDOSESEQ, LDOSEVAR)
```

derive_param_bmi	<i>Adds a Parameter for BMI</i>
------------------	---------------------------------

Description

Adds a record for BMI/Body Mass Index using Weight and Height each by group (e.g., subject and visit) where the source parameters are available.

Usage

```
derive_param_bmi(
  dataset,
  by_vars,
  set_values_to = vars(PARAMCD = "BMI"),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  get_unit_expr,
  filter = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code> , and <code>AVAL</code> are expected. The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>weight_code</code> and <code>height_code</code> .
by_vars	Grouping variables For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records. <i>Permitted Values:</i> list of variables
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter. <i>Permitted Values:</i> List of variable-value pairs
weight_code	WEIGHT parameter code The observations where <code>PARAMCD</code> equals the specified value are considered as the WEIGHT. It is expected that WEIGHT is measured in kg <i>Permitted Values:</i> character value
height_code	HEIGHT parameter code The observations where <code>PARAMCD</code> equals the specified value are considered as the HEIGHT. It is expected that HEIGHT is measured in cm <i>Permitted Values:</i> character value

get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters. Permitted Values: A variable of the input dataset or a function call
filter	Filter condition The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account. <i>Permitted Values:</i> a condition

Details

The analysis value of the new parameter is derived as

$$BMI = \frac{WEIGHT}{HEIGHT^2}$$

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Pavan Kumar

Examples

```
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 147, "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.0, "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.4, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 53.1, "WEEK 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 163, "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 78.5, "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.3, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.7, "WEEK 2"
)

derive_param_bmi(
  advs,
  by_vars = vars(USUBJID, AVISIT),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = vars(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

derive_param_bsa	<i>Adds a Parameter for BSA (Body Surface Area) Using the Specified Method</i>
------------------	--

Description

Adds a record for BSA (Body Surface Area) using the specified derivation method for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```
derive_param_bsa(
  dataset,
  by_vars,
  method,
  set_values_to = vars(PARAMCD = "BSA"),
  height_code = "HEIGHT",
  weight_code = "WEIGHT",
  get_unit_expr,
  filter = NULL
)
```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code>, and <code>AVAL</code> are expected.</p> <p>The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>HEIGHT</code> and <code>WEIGHT</code>.</p>
by_vars	<p>Grouping variables</p> <p>For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records.</p> <p><i>Permitted Values:</i> list of variables</p>
method	<p>Derivation method to use. Note that <code>HEIGHT</code> is expected in cm and <code>WEIGHT</code> is expected in kg:</p> <p>Mosteller: $\sqrt{\text{height} * \text{weight} / 3600}$</p> <p>DuBois-DuBois: $0.20247 * (\text{height}/100) ^ 0.725 * \text{weight} ^ 0.425$</p> <p>Haycock: $0.024265 * \text{height} ^ 0.3964 * \text{weight} ^ 0.5378$</p> <p>Gehan-George: $0.0235 * \text{height} ^ 0.42246 * \text{weight} ^ 0.51456$</p> <p>Boyd: $0.0003207 * (\text{height} ^ 0.3) * (1000 * \text{weight}) ^ (0.7285 - (0.0188 * \log_{10}(1000 * \text{weight})))$</p> <p>Fujimoto: $0.008883 * \text{height} ^ 0.663 * \text{weight} ^ 0.444$</p> <p>Takahira: $0.007241 * \text{height} ^ 0.725 * \text{weight} ^ 0.425$</p> <p><i>Permitted Values:</i> character value</p>

set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter.</p> <p><i>Permitted Values:</i> List of variable-value pairs</p>
height_code	<p>HEIGHT parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the HEIGHT assessments. It is expected that HEIGHT is measured in cm.</p> <p>Permitted Values: character value</p>
weight_code	<p>WEIGHT parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the WEIGHT assessments. It is expected that WEIGHT is measured in kg.</p> <p>Permitted Values: character value</p>
get_unit_expr	<p>An expression providing the unit of the parameter</p> <p>The result is used to check the units of the input parameters.</p> <p>Permitted Values: A variable of the input dataset or a function call</p>
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Eric Simms

Examples

```
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 170, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 75, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 78, "MONTH 1",
  "01-701-1015", "WEIGHT", "Weight (kg)", 80, "MONTH 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 185, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 90, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 88, "MONTH 1",
  "01-701-1028", "WEIGHT", "Weight (kg)", 85, "MONTH 2",
)

derive_param_bsa(
  advs,
  by_vars = vars(USUBJID, VISIT),
  method = "Mosteller",
)
```



```

    set_values_to = vars(
      PARAMCD = "BSA",
      PARAM = "Body Surface Area (m^2)"
    ),
    get_unit_expr = extract_unit(PARAM)
  )

  derive_param_bsa(
    advs,
    by_vars = vars(USUBJID, VISIT),
    method = "Fujimoto",
    set_values_to = vars(
      PARAMCD = "BSA",
      PARAM = "Body Surface Area (m^2)"
    ),
    get_unit_expr = extract_unit(PARAM)
  )
)

```

derive_param_doseint *Adds a Parameter for Dose Intensity*

Description

Adds a record for the dose intensity for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```

derive_param_doseint(
  dataset,
  by_vars,
  set_values_to = vars(PARAMCD = "TNDOSINT"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
  zero_doses = "Inf",
  filter = NULL
)

```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code> , and <code>AVAL</code> are expected. The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>tadm_code</code> and <code>padm_code</code> .
---------	--

by_vars	<p>Grouping variables</p> <p>Only variables specified in by_vars will be populated in the newly created records.</p> <p>Permitted Values: list of variables</p>
set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.</p> <p><i>Permitted Values:</i> List of variable-value pairs</p>
tadm_code	<p>Total Doses Administered parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the total dose administered. The AVAL associated with this PARAMCD will be the numerator of the dose intensity calculation.</p> <p>Permitted Values: character value</p>
tpadm_code	<p>Total Doses Planned parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the total planned dose. The AVAL associated with this PARAMCD will be the denominator of the dose intensity calculation.</p> <p>Permitted Values: character value</p>
zero_doses	<p>Flag indicating logic for handling 0 planned or administered doses for a by_vars group</p> <p>Default: Inf</p> <p>Permitted Values: Inf, 100</p> <p>No record is returned if either the planned (tpadm_code) or administered (tadm_code) AVAL are NA. No record is returned if a record does not exist for both tadm_code and tpadm_code for the specified by_var.</p> <p>If zero_doses = Inf:</p> <ol style="list-style-type: none"> 1. If the planned dose (tpadm_code) is 0 and administered dose (tadm_code) is 0, NaN is returned. 2. If the planned dose (tpadm_code) is 0 and the administered dose (tadm_code) is > 0, Inf is returned. <p>If zero_doses = 100 :</p> <ol style="list-style-type: none"> 1. If the planned dose (tpadm_code) is 0 and administered dose (tadm_code) is 0, 0 is returned. 2. If the planned dose (tpadm_code) is 0 and the administered dose (tadm_code) is > 0, 100 is returned.
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

Details

The analysis value of the new parameter is derived as Total Dose / Planned Dose * 100

Value

The input dataset with the new parameter rows added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Alice Ehmann

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)

adex <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~VISIT, ~ANL01FL, ~ASTDT, ~AENDT, ~AVAL,
  "P001", "TNDOSE", "V1", "Y", ymd("2020-01-01"), ymd("2020-01-30"), 59,
  "P001", "TSNDOSE", "V1", "Y", ymd("2020-01-01"), ymd("2020-02-01"), 96,
  "P001", "TNDOSE", "V2", "Y", ymd("2020-02-01"), ymd("2020-03-15"), 88,
  "P001", "TSNDOSE", "V2", "Y", ymd("2020-02-05"), ymd("2020-03-01"), 88,
  "P002", "TNDOSE", "V1", "Y", ymd("2021-01-01"), ymd("2021-01-30"), 0,
  "P002", "TSNDOSE", "V1", "Y", ymd("2021-01-01"), ymd("2021-02-01"), 0,
  "P002", "TNDOSE", "V2", "Y", ymd("2021-02-01"), ymd("2021-03-15"), 52,
  "P002", "TSNDOSE", "V2", "Y", ymd("2021-02-05"), ymd("2021-03-01"), 0
)

derive_param_doseint(
  adex,
  by_vars = vars(USUBJID, VISIT),
  set_values_to = vars(PARAMCD = "TNDOSINT"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE"
)

derive_param_doseint(
  adex,
  by_vars = vars(USUBJID, VISIT),
  set_values_to = vars(PARAMCD = "TNDOSINT2"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
  zero_doses = "100"
)
```

derive_param_exist_flag

Add an Existence Flag Parameter

Description

Add a new parameter indicating that a certain event exists in a dataset. AVALC and AVAL indicate if an event occurred or not. For example, the function can derive a parameter indicating if there is measurable disease at baseline.

Usage

```
derive_param_exist_flag(
  dataset = NULL,
  dataset_adsl,
  dataset_add,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL,
  aval_fun = yn_to_numeric,
  subject_keys = vars(STUDYID, USUBJID),
  set_values_to
)
```

Arguments

dataset	Input dataset The variables specified for subject_keys and the PARAMCD variable are expected.
dataset_adsl	ADSL input dataset The variables specified for subject_keys are expected. For each subject (as defined by subject_keys) from the specified dataset (dataset_adsl), the existence flag is calculated and added as a new observation to the input datasets (dataset)
dataset_add	Additional dataset The variables specified by the subject_keys parameter are expected. This dataset is used to check if an event occurred or not. Any observation in the dataset fulfilling the event condition (condition) is considered as an event.
condition	Event condition The condition is evaluated at the additional dataset (dataset_add). For all subjects where it evaluates as TRUE at least once AVALC is set to the true value (true_value) for the new observations. For all subjects where it evaluates as FALSE or NA for all observations AVALC is set to the false value (false_value). For all subjects not present in the additional dataset AVALC is set to the missing value (missing_value).
true_value	True value For all subjects with at least one observations in the additional dataset (dataset_add) fulfilling the event condition (condition), AVALC is set to the specified value (true_value).

	<i>Default:</i> "Y"
	<i>Permitted Value:</i> A character scalar
false_value	False value For all subjects with at least one observations in the additional dataset (dataset_add) but none of them is fulfilling the event condition (condition), AVALC is set to the specified value (false_value). <i>Default:</i> NA_character_ <i>Permitted Value:</i> A character scalar
missing_value	Values used for missing information For all subjects without an observation in the additional dataset (dataset_add), AVALC is set to the specified value (missing_value). <i>Default:</i> NA_character_ <i>Permitted Value:</i> A character scalar
filter_add	Filter for additional data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. <i>Permitted Values:</i> a condition
aval_fun	Function to map character analysis value (AVALC) to numeric analysis value (AVAL) The (first) argument of the function must expect a character vector and the function must return a numeric vector. <i>Default:</i> yn_to_numeric (see yn_to_numeric() for details)
subject_keys	Variables to uniquely identify a subject A list of symbols created using vars() is expected.
set_values_to	Variables to set A named list returned by vars() defining the variables to be set for the new parameter, e.g. vars(PARAMCD = "MDIS", PARAM = "Measurable Disease at Baseline") is expected. The values must be symbols, character strings, numeric values, or NA.

Details

1. The additional dataset (dataset_add) is restricted to the observations matching the filter_add condition.
2. For each subject in dataset_ads1 a new observation is created.
 - The AVALC variable is added and set to the true value (true_value) if for the subject at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE.
 - It is set to the false value (false_value) if for the subject at least one observation exists and for all observations the condition evaluates to FALSE or NA.
 - Otherwise, it is set to the missing value (missing_value), i.e., for those subject not in dataset_add.
3. The AVAL variable is added and set to aval_fun(AVALC).
4. The variables specified by the set_values_to parameter are added to the new observations.
5. The new observations are added to input dataset.

Value

The input dataset with a new parameter indicating if an event occurred (AVALC, AVAL, and the variables specified by subject_keys and set_value_to are populated for the new parameter)

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr)
library(lubridate)

# Derive a new parameter for measurable disease at baseline
adsl <- tibble::tribble(
  ~USUBJID,
  "1",
  "2",
  "3"
) %>%
  mutate(STUDYID = "XX1234")

tu <- tibble::tribble(
  ~USUBJID, ~VISIT, ~TUSTRESC,
  "1", "SCREENING", "TARGET",
  "1", "WEEK 1", "TARGET",
  "1", "WEEK 5", "TARGET",
  "1", "WEEK 9", "NON-TARGET",
  "2", "SCREENING", "NON-TARGET",
  "2", "SCREENING", "NON-TARGET"
) %>%
  mutate(
    STUDYID = "XX1234",
    TUTESTCD = "TUMIDENT"
  )

derive_param_exist_flag(
  dataset_adsl = adsl,
  dataset_add = tu,
  filter_add = TUTESTCD == "TUMIDENT" & VISIT == "SCREENING",
  condition = TUSTRESC == "TARGET",
  false_value = "N",
  missing_value = "N",
  set_values_to = vars(
    PARAMCD = "MDIS",
    PARAM = "Measurable Disease at Baseline"
  )
)
```

derive_param_exposure *Add an Aggregated Parameter and Derive the Associated Start and End Dates*

Description

Add a record computed from the aggregated analysis value of another parameter and compute the start (ASTDT(M)) and end date (AENDT(M)) as the minimum and maximum date by `by_vars`.

Usage

```
derive_param_exposure(  
  dataset,  
  by_vars,  
  input_code,  
  analysis_var,  
  summary_fun,  
  filter = NULL,  
  set_values_to = NULL  
)
```

Arguments

<code>dataset</code>	Input dataset <ul style="list-style-type: none"> • The variables specified by the <code>by_vars</code>, <code>analysis_var</code> parameters and <code>PARAMCD</code> are expected, • Either <code>ASTDTM</code> and <code>AENDTM</code> or <code>ASTDT</code> and <code>AENDT</code> are also expected.
<code>by_vars</code>	Grouping variables For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records. <i>Permitted Values:</i> list of variables
<code>input_code</code>	Required parameter code The observations where <code>PARAMCD</code> equals the specified value are considered to compute the summary record. <i>Permitted Values:</i> A character of <code>PARAMCD</code> value
<code>analysis_var</code>	Analysis variable.
<code>summary_fun</code>	Function that takes as an input the <code>analysis_var</code> and performs the calculation. This can include built-in functions as well as user defined functions, for example <code>mean</code> or <code>function(x) mean(x, na.rm = TRUE)</code> .
<code>filter</code>	Filter condition The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account. <i>Permitted Values:</i> a condition

`set_values_to` Variable-value pairs
 Set a list of variables to some specified value for the new observation(s)

- LHS refer to a variable. It is expected that at least PARAMCD is defined.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or NA. (e.g. `vars(PARAMCD = "TDOSE", PARCAT1 = "OVERALL")`). More general expression are not allowed.

Permitted Values: List of variable-value pairs

Details

For each group (with respect to the variables specified for the `by_vars` parameter), an observation is added to the output dataset and the defined values are set to the defined variables

Value

The input dataset with a new record added for each group (with respect to the variables specified for the `by_vars` parameter). That is, a variable will only be populated in this new record if it is specified in `by_vars`. For each new record,

- the variable specified `analysis_var` is computed as defined by `summary_fun`,
- the variable(s) specified on the LHS of `set_values_to` are set to their paired value (RHS). In addition, the start and end date are computed as the minimum/maximum dates by `by_vars`.

If the input datasets contains

- both `AxxDTM` and `AxxDT` then all `ASTDTM`, `AENDTM`, `ASTDT`, `AENDT` are computed
- only `AxxDTM` then `ASTDTM`, `AENDTM` are computed
- only `AxxDT` then `ASTDT`, `AENDT` are computed.

Author(s)

Samia Kabi

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(stringr, warn.conflicts = FALSE)
adex <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~AVALC, ~VISIT, ~ASTDT, ~AENDT,
  "1015", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "DOSE", 85, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "DOSE", 82, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1015", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "ADJ", NA, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1017", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
  "1017", "DOSE", 50, NA_character_, "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
  "1017", "DOSE", 65, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02"),
  "1017", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
```



```

"1017", "ADJ", NA, "ADVERSE EVENT", "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
"1017", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02")
) %>%
mutate(ASTDTM = ymd_hms(paste(ASTDT, "00:00:00")), AENDTM = ymd_hms(paste(AENDT, "00:00:00")))

# Cumulative dose
adex %>%
  derive_param_exposure(
    by_vars = vars(USUBJID),
    set_values_to = vars(PARAMCD = "TDOSE", PARCAT1 = "OVERALL"),
    input_code = "DOSE",
    analysis_var = AVAL,
    summary_fun = function(x) sum(x, na.rm = TRUE)
  ) %>%
  select(-ASTDTM, -AENDTM)

# average dose in w2-24
adex %>%
  derive_param_exposure(
    by_vars = vars(USUBJID),
    filter = VISIT %in% c("WEEK 2", "WEEK 24"),
    set_values_to = vars(PARAMCD = "AVDW224", PARCAT1 = "WEEK2-24"),
    input_code = "DOSE",
    analysis_var = AVAL,
    summary_fun = function(x) mean(x, na.rm = TRUE)
  ) %>%
  select(-ASTDTM, -AENDTM)

# Any dose adjustment?
adex %>%
  derive_param_exposure(
    by_vars = vars(USUBJID),
    set_values_to = vars(PARAMCD = "TADJ", PARCAT1 = "OVERALL"),
    input_code = "ADJ",
    analysis_var = AVALC,
    summary_fun = function(x) if_else(sum(!is.na(x)) > 0, "Y", NA_character_)
  ) %>%
  select(-ASTDTM, -AENDTM)

```

derive_param_first_event

Add a First Event Parameter

Description

Add a new parameter for the first event occurring in a dataset. AVALC and AVAL indicate if an event occurred and ADT is set to the date of the first event. For example, the function can derive a parameter for the first disease progression.

Usage

```

derive_param_first_event(
  dataset,
  dataset_adsl,
  dataset_source,
  filter_source,
  date_var,
  subject_keys = vars(STUDYID, USUBJID),
  set_values_to,
  check_type = "warning"
)

```

Arguments

dataset	Input dataset The PARAMCD variable is expected.
dataset_adsl	ADSL input dataset The variables specified for subject_keys are expected. For each observation of the specified dataset a new observation is added to the input dataset.
dataset_source	Source dataset All observations in the specified dataset fulfilling the condition specified by filter_source are considered as event. The variables specified by the subject_keys and date_var parameter are expected.
filter_source	Source filter All observations in dataset_source fulfilling the specified condition are considered as event. For subjects with at least one event AVALC is set to "Y", AVAL to 1, and ADT to the first date where the condition is fulfilled. For all other subjects AVALC is set to "N", AVAL to 0, and ADT to NA.
date_var	Date variable Date variable in the source dataset (dataset_source). The variable is used to sort the source dataset. ADT is set to the specified variable for events.
subject_keys	Variables to uniquely identify a subject A list of symbols created using vars() is expected.
set_values_to	Variables to set A named list returned by vars() defining the variables to be set for the new parameter, e.g. vars(PARAMCD = "PD", PARAM = "Disease Progression") is expected. The values must be symbols, character strings, numeric values, or NA.
check_type	Check uniqueness? If "warning" or "error" is specified, a message is issued if the observations of the input dataset restricted to the source parameter (source_param) are not unique with respect to the subject keys (subject_key parameter) and ADT. <i>Default:</i> "warning" <i>Permitted Values:</i> "none", "warning", "error"

Details

1. The input dataset is restricted to observations fulfilling `filter_source`.
2. For each subject (with respect to the variables specified for the `subject_keys` parameter) the first observation (with respect to `date_var`) where the event condition (`filter_source` parameter) is fulfilled is selected.
3. For each observation in `dataset_adsl` a new observation is created. For subjects with event AVALC is set to "Y", AVAL to 1, and ADT to the first date where the event condition is fulfilled. For all other subjects AVALC is set to "N", AVAL to 0, and ADT to NA.
4. The variables specified by the `set_values_to` parameter are added to the new observations.
5. The new observations are added to input dataset.

Value

The input dataset with a new parameter indicating if and when an event occurred

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr)
library(lubridate)

# Derive a new parameter for the first disease progression (PD)
adsl <- tibble::tribble(
  ~USUBJID, ~DTHDT,
  "1",      ymd("2022-05-13"),
  "2",      ymd(""),
  "3",      ymd("")
) %>%
  mutate(STUDYID = "XX1234")

adrs <- tibble::tribble(
  ~USUBJID, ~ADTC,      ~AVALC,
  "1",      "2020-01-02", "PR",
  "1",      "2020-02-01", "CR",
  "1",      "2020-03-01", "CR",
  "1",      "2020-04-01", "SD",
  "2",      "2021-06-15", "SD",
  "2",      "2021-07-16", "PD",
  "2",      "2021-09-14", "PD"
) %>%
  mutate(
    STUDYID = "XX1234",
    ADT = ymd(ADTC),
    PARAMCD = "OVR",
    PARAM = "Overall Response",
    ANL01FL = "Y"
  ) %>%
```

```

select(-ADTC)

derive_param_first_event(
  adrs,
  dataset_adsl = adsl,
  dataset_source = adrs,
  filter_source = PARAMCD == "OVR" & AVALC == "PD",
  date_var = ADT,
  set_values_to = vars(
    PARAMCD = "PD",
    PARAM = "Disease Progression",
    ANL01FL = "Y"
  )
)

# derive parameter indicating death
derive_param_first_event(
  dataset = adrs,
  dataset_adsl = adsl,
  dataset_source = adsl,
  filter_source = !is.na(DTHDT),
  date_var = DTHDT,
  set_values_to = vars(
    PARAMCD = "DEATH",
    PARAM = "Death",
    ANL01FL = "Y"
  )
)

```

derive_param_map	<i>Adds a Parameter for Mean Arterial Pressure</i>
------------------	--

Description

Adds a record for mean arterial pressure (MAP) for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```

derive_param_map(
  dataset,
  by_vars,
  set_values_to = vars(PARAMCD = "MAP"),
  sysbp_code = "SYSBP",
  diabp_code = "DIABP",
  hr_code = NULL,
  get_unit_expr,
  filter = NULL
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code>, and <code>AVAL</code> are expected.</p> <p>The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>sysbp_code</code>, <code>diabp_code</code> and <code>hr_code</code>.</p>
by_vars	<p>Grouping variables</p> <p>For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records.</p> <p><i>Permitted Values:</i> list of variables</p>
set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter.</p> <p><i>Permitted Values:</i> List of variable-value pairs</p>
sysbp_code	<p>Systolic blood pressure parameter code</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the systolic blood pressure assessments.</p> <p><i>Permitted Values:</i> character value</p>
diabp_code	<p>Diastolic blood pressure parameter code</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the diastolic blood pressure assessments.</p> <p><i>Permitted Values:</i> character value</p>
hr_code	<p>Heart rate parameter code</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the heart rate assessments.</p> <p><i>Permitted Values:</i> character value</p>
get_unit_expr	<p>An expression providing the unit of the parameter</p> <p>The result is used to check the units of the input parameters.</p> <p><i>Permitted Values:</i> A variable of the input dataset or a function call</p>
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

Details

The analysis value of the new parameter is derived as

$$\frac{2DIABP + SYSBP}{3}$$

if it is based on diastolic and systolic blood pressure and

$$DIABP + 0.01e^{4.14 - \frac{40.74}{HR}} (SYSBP - DIABP)$$

if it is based on diastolic, systolic blood pressure, and heart rate.

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflicts = FALSE)

advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-701-1015", "PULSE", "Pulse (beats/min)", 59, "BASELINE",
  "01-701-1015", "PULSE", "Pulse (beats/min)", 61, "WEEK 2",
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 51, "BASELINE",
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 50, "WEEK 2",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "BASELINE",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "WEEK 2",
  "01-701-1028", "PULSE", "Pulse (beats/min)", 62, "BASELINE",
  "01-701-1028", "PULSE", "Pulse (beats/min)", 77, "WEEK 2",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 79, "BASELINE",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 80, "WEEK 2",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 130, "BASELINE",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 132, "WEEK 2"
)

# Derive MAP based on diastolic and systolic blood pressure
advs %>%
  derive_param_map(
    by_vars = vars(USUBJID, VISIT),
    set_values_to = vars(
      PARAMCD = "MAP",
      PARAM = "Mean Arterial Pressure (mmHg)"
    ),
    get_unit_expr = extract_unit(PARAM)
  ) %>%
  filter(PARAMCD != "PULSE")

# Derive MAP based on diastolic and systolic blood pressure and heart rate
derive_param_map(
  advs,
  by_vars = vars(USUBJID, VISIT),
  hr_code = "PULSE",
  set_values_to = vars(
```

```

    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)"
  ),
  get_unit_expr = extract_unit(PARAM)
)

```

derive_param_qtc *Adds a Parameter for Corrected QT (an ECG measurement)*

Description

Adds a record for corrected QT using either Bazett's, Fridericia's or Sagie's formula for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```

derive_param_qtc(
  dataset,
  by_vars,
  method,
  set_values_to = default_qtc_paramcd(method),
  qt_code = "QT",
  rr_code = "RR",
  get_unit_expr,
  filter = NULL
)

```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> and the <code>unit_var</code> parameter, <code>PARAMCD</code> , and <code>AVAL</code> are expected. The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>qt_code</code> and <code>rr_code</code> .
by_vars	Grouping variables Only variables specified in <code>by_vars</code> will be populated in the newly created records. Permitted Values: list of variables
method	Method used to QT correction Permitted Values: "Bazett", "Fridericia", "Sagie"
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter. <i>Permitted Values:</i> List of variable-value pairs

qt_code	<p>QT parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the QT interval assessments. It is expected that QT is measured in msec.</p> <p>Permitted Values: character value</p>
rr_code	<p>RR parameter code</p> <p>The observations where PARAMCD equals the specified value are considered as the RR interval assessments. It is expected that RR is measured in msec.</p> <p>Permitted Values: character value</p>
get_unit_expr	<p>An expression providing the unit of the parameter</p> <p>The result is used to check the units of the input parameters.</p> <p>Permitted Values: A variable of the input dataset or a function call</p>
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Stefan Bundfuss

See Also

[compute_qtc\(\)](#)

Examples

```

adeg <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "HR", "Heart Rate (beats/min)", 70.14, "beats/min", "BASELINE",
  "01-701-1015", "QT", "QT Duration (msec)", 370, "msec", "WEEK 2",
  "01-701-1015", "HR", "Heart Rate (beats/min)", 62.66, "beats/min", "WEEK 1",
  "01-701-1015", "RR", "RR Duration (msec)", 710, "msec", "WEEK 2",
  "01-701-1028", "HR", "Heart Rate (beats/min)", 85.45, "beats/min", "BASELINE",
  "01-701-1028", "QT", "QT Duration (msec)", 480, "msec", "WEEK 2",
  "01-701-1028", "QT", "QT Duration (msec)", 350, "msec", "WEEK 3",
  "01-701-1028", "HR", "Heart Rate (beats/min)", 56.54, "beats/min", "WEEK 3",
  "01-701-1028", "RR", "RR Duration (msec)", 842, "msec", "WEEK 2",
)

derive_param_qtc(
  adeg,
  by_vars = vars(USUBJID, VISIT),
  method = "Bazett",
  set_values_to = vars(

```



```

    PARAMCD = "QTCBR",
    PARAM = "QTcB - Bazett's Correction Formula Rederived (msec)",
    AVALU = "msec"
  ),
  get_unit_expr = AVALU
)

derive_param_qtc(
  adeg,
  by_vars = vars(USUBJID, VISIT),
  method = "Fridericia",
  set_values_to = vars(
    PARAMCD = "QTCFR",
    PARAM = "QTcF - Fridericia's Correction Formula Rederived (msec)",
    AVALU = "msec"
  ),
  get_unit_expr = extract_unit(PARAM)
)

derive_param_qtc(
  adeg,
  by_vars = vars(USUBJID, VISIT),
  method = "Sagie",
  set_values_to = vars(
    PARAMCD = "QTLCR",
    PARAM = "QTlc - Sagie's Correction Formula Rederived (msec)",
    AVALU = "msec"
  ),
  get_unit_expr = extract_unit(PARAM)
)

```

derive_param_rr	<i>Adds a Parameter for Derived RR (an ECG measurement)</i>
-----------------	---

Description

Adds a record for derived RR based on heart rate for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```

derive_param_rr(
  dataset,
  by_vars,
  set_values_to = vars(PARAMCD = "RRR"),
  hr_code = "HR",
  get_unit_expr,
  filter = NULL
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter, <code>PARAMCD</code>, and <code>AVAL</code> are expected.</p> <p>The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset after restricting it by the filter condition (<code>filter</code> parameter) and to the parameters specified by <code>hr_code</code>.</p>
by_vars	<p>Grouping variables</p> <p>For each group defined by <code>by_vars</code> an observation is added to the output dataset. Only variables specified in <code>by_vars</code> will be populated in the newly created records.</p> <p><i>Permitted Values:</i> list of variables</p>
set_values_to	<p>Variables to be set</p> <p>The specified variables are set to the specified values for the new observations. For example <code>vars(PARAMCD = "MAP")</code> defines the parameter code for the new parameter.</p> <p><i>Permitted Values:</i> List of variable-value pairs</p>
hr_code	<p>HR parameter code</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the heart rate assessments.</p> <p><i>Permitted Values:</i> character value</p>
get_unit_expr	<p>An expression providing the unit of the parameter</p> <p>The result is used to check the units of the input parameters.</p> <p><i>Permitted Values:</i> A variable of the input dataset or a function call</p>
filter	<p>Filter condition</p> <p>The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.</p> <p><i>Permitted Values:</i> a condition</p>

Details

The analysis value of the new parameter is derived as

$$\frac{60000}{HR}$$

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

Author(s)

Stefan Bundfuss

Examples

```

adeg <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "HR", "Heart Rate", 70.14, "beats/min", "BASELINE",
  "01-701-1015", "QT", "QT Duration", 370, "msec", "WEEK 2",
  "01-701-1015", "HR", "Heart Rate", 62.66, "beats/min", "WEEK 1",
  "01-701-1015", "RR", "RR Duration", 710, "msec", "WEEK 2",
  "01-701-1028", "HR", "Heart Rate", 85.45, "beats/min", "BASELINE",
  "01-701-1028", "QT", "QT Duration", 480, "msec", "WEEK 2",
  "01-701-1028", "QT", "QT Duration", 350, "msec", "WEEK 3",
  "01-701-1028", "HR", "Heart Rate", 56.54, "beats/min", "WEEK 3",
  "01-701-1028", "RR", "RR Duration", 842, "msec", "WEEK 2"
)

derive_param_rr(
  adeg,
  by_vars = vars(USUBJID, VISIT),
  set_values_to = vars(
    PARAMCD = "RRR",
    PARAM = "RR Duration Rederived (msec)",
    AVALU = "msec"
  ),
  get_unit_expr = AVALU
)

```

 derive_param_tte

Derive a Time-to-Event Parameter

Description

Add a time-to-event parameter to the input dataset.

Usage

```

derive_param_tte(
  dataset = NULL,
  dataset_adsl,
  source_datasets,
  by_vars = NULL,
  start_date = TRTSDT,
  event_conditions,
  censor_conditions,
  create_datetime = FALSE,
  set_values_to,
  subject_keys = vars(STUDYID, USUBJID)
)

```

Arguments

dataset	Input dataset The PARAMCD variable is expected.
dataset_adsl	ADSL input dataset The variables specified for start_date, start_imputation_flag, and subject_keys are expected.
source_datasets	Source datasets A named list of datasets is expected. The dataset_name field of tte_source() refers to the dataset provided in the list.
by_vars	By variables If the parameter is specified, separate time to event parameters are derived for each by group. The by variables must be in at least one of the source datasets. Each source dataset must contain either all by variables or none of the by variables. The by variables are not included in the output dataset.
start_date	Time to event origin date The variable STARTDT is set to the specified date. The value is taken from the ADSL dataset. If the event or censoring date is before the origin date, ADT is set to the origin date. If the specified variable is imputed, the corresponding date imputation flag must be specified for start_imputation_flag.
event_conditions	Sources and conditions defining events A list of event_source() objects is expected.
censor_conditions	Sources and conditions defining censorings A list of censor_source() objects is expected.
create_datetime	Create datetime variables? If set to TRUE, variables ADTM and STARTDTM are created. Otherwise, variables ADT and STARTDT are created.
set_values_to	Variables to set A named list returned by vars() defining the variables to be set for the new parameter, e.g. vars(PARAMCD = "OS", PARAM = "Overall Survival") is expected. The values must be symbols, character strings, numeric values, expressions, or NA.
subject_keys	Variables to uniquely identify a subject A list of symbols created using vars() is expected.

Details

The following steps are performed to create the observations of the new parameter:

Deriving the events:

1. For each event source dataset the observations as specified by the `filter` element are selected. Then for each patient the first observation (with respect to date) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied. If the source variable is a character variable, it is converted to a date. If the date is incomplete, it is imputed as the first possible date.
3. The CNSR variable is added and set to the censor element.
4. The variables specified by the `set_values_to` element are added.
5. The selected observations of all event source datasets are combined into a single dataset.
6. For each patient the first observation (with respect to the ADT variable) from the single dataset is selected.

Deriving the censoring observations:

1. For each censoring source dataset the observations as specified by the `filter` element are selected. Then for each patient the last observation (with respect to date) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied. If the source variable is a character variable, it is converted to a date. If the date is incomplete, it is imputed as the first possible date.
3. The CNSR variable is added and set to the censor element.
4. The variables specified by the `set_values_to` element are added.
5. The selected observations of all censoring source datasets are combined into a single dataset.
6. For each patient the last observation (with respect to the ADT variable) from the single dataset is selected.

For each subject (as defined by the `subject_keys` parameter) an observation is selected. If an event is available, the event observation is selected. Otherwise the censoring observation is selected.

Finally

1. the variables specified for `start_date` and `start_imputation_flag` are joined from the ADSL dataset,
2. the variables as defined by the `set_values_to` parameter are added,
3. the ADT/ADTM variable is set to the maximum of ADT/ADTM and STARTDT/STARTDTM (depending on the `create_datetime` parameter), and
4. the new observations are added to the output dataset.

Value

The input dataset with the new parameter added

Author(s)

Stefan Bundfuss

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(lubridate)
data("admiral_adsl")

adsl <- admiral_adsl

death <- event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = vars(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)

last_alive_dt <- censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = vars(
    EVNTDESC = "LAST DATE KNOWN ALIVE",
    SRCDOM = "ADSL",
    SRCVAR = "LSTALVDT"
  )
)

derive_param_tte(
  dataset_adsl = adsl,
  event_conditions = list(death),
  censor_conditions = list(last_alive_dt),
  source_datasets = list(adsl = adsl),
  set_values_to = vars(
    PARAMCD = "OS",
    PARAM = "Overall Survival"
  )
) %>%
  select(-STUDYID) %>%
  filter(row_number() %in% 20:30)

# derive time to adverse event for each preferred term #
adsl <- tibble::tribble(
  ~USUBJID, ~TRTSDT, ~EOSDT,
  "01", ymd("2020-12-06"), ymd("2021-03-06"),
  "02", ymd("2021-01-16"), ymd("2021-02-03")
) %>%
  mutate(STUDYID = "AB42")

ae <- tibble::tribble(
  ~USUBJID, ~AESTDTC, ~AESEQ, ~AEDECOD,
  "01", "2021-01-03T10:56", 1, "Flu",

```

```

    "01",      "2021-03-04",    2,      "Cough",
    "01",      "2021",          3,      "Flu"
  ) %>%
  mutate(STUDYID = "AB42")

  ttae <- event_source(
    dataset_name = "ae",
    date = AESTDTC,
    set_values_to = vars(
      EVNTDESC = "AE",
      SRCDOM = "AE",
      SRCVAR = "AESTDTC",
      SRCSEQ = AESEQ
    )
  )

  eos <- censor_source(
    dataset_name = "adsl",
    date = EOSDT,
    set_values_to = vars(
      EVNTDESC = "END OF STUDY",
      SRCDOM = "ADSL",
      SRCVAR = "EOSDT"
    )
  )

  derive_param_tte(
    dataset_adsl = adsl,
    by_vars = vars(AEDECOD),
    start_date = TRTSDT,
    event_conditions = list(ttae),
    censor_conditions = list(eos),
    source_datasets = list(adsl = adsl, ae = ae),
    set_values_to = vars(
      PARAMCD = paste0("TTAE", as.numeric(as.factor(AEDECOD))),
      PARAM = paste("Time to First", AEDECOD, "Adverse Event"),
      PARCAT1 = "TTAE",
      PARCAT2 = AEDECOD
    )
  ) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)

```

derive_param_wbc_abs *Add a parameter for lab differentials converted to absolute values*

Description

Add a parameter by converting lab differentials from fraction or percentage to absolute values

Usage

```

derive_param_wbc_abs(
  dataset,
  by_vars,
  set_values_to,
  get_unit_expr,
  wbc_unit = "10^9/L",
  wbc_code = "WBC",
  diff_code,
  diff_type = "fraction"
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> argument, <code>PARAMCD</code>, and <code>AVAL</code> are expected to be present.</p> <p>The variable specified by <code>by_vars</code> and <code>PARAMCD</code> must be a unique key of the input dataset, and to the parameters specified by <code>wbc_code</code> and <code>diff_code</code>.</p>
by_vars	<p>Grouping variables</p> <p>Permitted Values: list of variables</p>
set_values_to	<p>Variables to set</p> <p>A named list returned by <code>vars()</code> defining the variables to be set for the new parameter, e.g. <code>vars(PARAMCD = "LYMPH", PARAM = "Lymphocytes Abs (10^9/L)")</code> is expected.</p>
get_unit_expr	<p>An expression providing the unit of the parameter</p> <p>The result is used to check the units of the input parameters.</p> <p>Permitted Values: a variable containing unit from the input dataset, or a function call, for example, <code>get_unit_expr = extract_unit(PARAM)</code>.</p>
wbc_unit	<p>A string containing the required unit of the WBC parameter</p> <p>Default: "10^9/L"</p>
wbc_code	<p>White Blood Cell (WBC) parameter</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the WBC absolute results to use for converting the differentials.</p> <p>Default: "WBC"</p> <p>Permitted Values: character value</p>
diff_code	<p>white blood differential parameter</p> <p>The observations where <code>PARAMCD</code> equals the specified value are considered as the white blood differential lab results in fraction or percentage value to be converted into absolute value.</p>
diff_type	<p>A string specifying the type of differential</p> <p>Permitted Values: "percent", "fraction" Default: fraction</p>

Details

If `diff_type` is "percent", the analysis value of the new parameter is derived as

$$\frac{WhiteBloodCellCount * PercentageValue}{100}$$

If `diff_type` is "fraction", the analysis value of the new parameter is derived as

$$WhiteBloodCellCount * FractionValue$$

New records are created for each group of records (grouped by `by_vars`) if 1) the white blood cell component in absolute value is not already available from the input dataset, and 2) the white blood cell absolute value (identified by `wbc_code`) and the white blood cell differential (identified by `diff_code`) are both present.

Value

The input dataset with the new parameter added

Author(s)

Annie Yang

Examples

```
library(dplyr, warn.conflicts = FALSE)
test_lb <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~PARAM, ~VISIT,
  "P01", "WBC", 33, "Leukocyte Count (10^9/L)", "CYCLE 1 DAY 1",
  "P01", "WBC", 38, "Leukocyte Count (10^9/L)", "CYCLE 2 DAY 1",
  "P01", "LYMLE", 0.90, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1",
  "P01", "LYMLE", 0.70, "Lymphocytes (fraction of 1)", "CYCLE 2 DAY 1",
  "P01", "ALB", 36, "Albumin (g/dL)", "CYCLE 2 DAY 1",
  "P02", "WBC", 33, "Leukocyte Count (10^9/L)", "CYCLE 1 DAY 1",
  "P02", "LYMPH", 29, "Lymphocytes Abs (10^9/L)", "CYCLE 1 DAY 1",
  "P02", "LYMLE", 0.87, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1",
  "P03", "LYMLE", 0.89, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1"
)

derive_param_wbc_abs(
  dataset = test_lb,
  by_vars = vars(USUBJID, VISIT),
  set_values_to = vars(
    PARAMCD = "LYMPH",
    PARAM = "Lymphocytes Abs (10^9/L)",
    DTYPE = "CALCULATION"
  ),
  get_unit_expr = extract_unit(PARAM),
  wbc_code = "WBC",
  diff_code = "LYMLE",
  diff_type = "fraction"
)
```

 derive_summary_records

Add New Records Within By Groups Using Aggregation Functions

Description

It is not uncommon to have an analysis need whereby one needs to derive an analysis value (AVAL) from multiple records. The ADaM basic dataset structure variable DTYPE is available to indicate when a new derived records has been added to a dataset.

Usage

```
derive_summary_records(
  dataset,
  by_vars,
  filter = NULL,
  analysis_var,
  summary_fun,
  set_values_to = NULL
)
```

Arguments

dataset	A data frame.
by_vars	Variables to consider for generation of groupwise summary records. Providing the names of variables in <code>vars()</code> will create a groupwise summary and generate summary records for the specified groups.
filter	Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within <code>by_vars</code> as this will help when an aggregating, lagging, or ranking function is involved. For example, <ul style="list-style-type: none"> • <code>filter = (AVAL > mean(AVAL, na.rm = TRUE))</code> will filter all AVAL values greater than mean of AVAL with in <code>by_vars</code>. • <code>filter = (dplyr::n() > 2)</code> will filter n count of <code>by_vars</code> greater than 2.
analysis_var	Analysis variable.
summary_fun	Function that takes as an input the <code>analysis_var</code> and performs the calculation. This can include built-in functions as well as user defined functions, for example <code>mean</code> or <code>function(x) mean(x, na.rm = TRUE)</code> .
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. A list of variable name-value pairs is expected. <ul style="list-style-type: none"> • LHS refers to a variable. • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or NA, e.g., <code>vars(PARAMCD = "TDOSE", PARCAT1 = "OVERALL")</code>. More general expression are not allowed.

Details

When all records have same values within `by_vars` then this function will retain those common values in the newly derived records. Otherwise new value will be set to NA.

Value

A data frame with derived records appended to original dataset.

Author(s)

Vignesh Thanikachalam, Ondrej Slama

Examples

```
library(dplyr, warn.conflicts = FALSE)
adeg <- tibble::tribble(
  ~USUBJID, ~EGSEQ, ~PARAM, ~AVISIT, ~EGDTC, ~AVAL, ~TRTA,
  "XYZ-1001", 1, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385, "",
  "XYZ-1001", 2, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399, "",
  "XYZ-1001", 3, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396, "",
  "XYZ-1001", 4, "QTcF Int. (msec)", "Visit 2", "2016-03-08T09:45", 384, "Placebo",
  "XYZ-1001", 5, "QTcF Int. (msec)", "Visit 2", "2016-03-08T09:48", 393, "Placebo",
  "XYZ-1001", 6, "QTcF Int. (msec)", "Visit 2", "2016-03-08T09:51", 388, "Placebo",
  "XYZ-1001", 7, "QTcF Int. (msec)", "Visit 3", "2016-03-22T10:45", 385, "Placebo",
  "XYZ-1001", 8, "QTcF Int. (msec)", "Visit 3", "2016-03-22T10:48", 394, "Placebo",
  "XYZ-1001", 9, "QTcF Int. (msec)", "Visit 3", "2016-03-22T10:51", 402, "Placebo",
  "XYZ-1002", 1, "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 399, "",
  "XYZ-1002", 2, "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410, "",
  "XYZ-1002", 3, "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392, "",
  "XYZ-1002", 4, "QTcF Int. (msec)", "Visit 2", "2016-03-06T09:50", 401, "Active 20mg",
  "XYZ-1002", 5, "QTcF Int. (msec)", "Visit 2", "2016-03-06T09:53", 407, "Active 20mg",
  "XYZ-1002", 6, "QTcF Int. (msec)", "Visit 2", "2016-03-06T09:56", 400, "Active 20mg",
  "XYZ-1002", 7, "QTcF Int. (msec)", "Visit 3", "2016-03-24T10:50", 412, "Active 20mg",
  "XYZ-1002", 8, "QTcF Int. (msec)", "Visit 3", "2016-03-24T10:53", 414, "Active 20mg",
  "XYZ-1002", 9, "QTcF Int. (msec)", "Visit 3", "2016-03-24T10:56", 402, "Active 20mg",
)

# Summarize the average of the triplicate ECG interval values (AVAL)
derive_summary_records(
  adeg,
  by_vars = vars(USUBJID, PARAM, AVISIT),
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = vars(DTYPE = "AVERAGE")
)

advs <- tibble::tribble(
  ~USUBJID, ~VSSEQ, ~PARAM, ~AVAL, ~VSSTRESU, ~VISIT, ~VSDTC,
  "XYZ-001-001", 1164, "Weight", 99, "kg", "Screening", "2018-03-19",
  "XYZ-001-001", 1165, "Weight", 101, "kg", "Run-In", "2018-03-26",
  "XYZ-001-001", 1166, "Weight", 100, "kg", "Baseline", "2018-04-16",
  "XYZ-001-001", 1167, "Weight", 94, "kg", "Week 24", "2018-09-30",
)
```

```

"XYZ-001-001", 1168, "Weight", 92, "kg", "Week 48", "2019-03-17",
"XYZ-001-001", 1169, "Weight", 95, "kg", "Week 52", "2019-04-14",
)

# Set new values to any variable. Here, `DTYPE = MAXIMUM` refers to `max()` records
# and `DTYPE = AVERAGE` refers to `mean()` records.
derive_summary_records(
  advs,
  by_vars = vars(USUBJID, PARAM),
  analysis_var = AVAL,
  summary_fun = max,
  set_values_to = vars(DTYPE = "MAXIMUM")
) %>%
  derive_summary_records(
    by_vars = vars(USUBJID, PARAM),
    analysis_var = AVAL,
    summary_fun = mean,
    set_values_to = vars(DTYPE = "AVERAGE")
  )

# Sample ADEG dataset with triplicate record for only AVISIT = 'Baseline'
adeg <- tibble::tribble(
  ~USUBJID, ~EGSEQ, ~PARAM, ~AVISIT, ~EGDTC, ~AVAL, ~TRTA,
  "XYZ-1001", 1, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385, "",
  "XYZ-1001", 2, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399, "",
  "XYZ-1001", 3, "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396, "",
  "XYZ-1001", 4, "QTcF Int. (msec)", "Visit 2", "2016-03-08T09:48", 393, "Placebo",
  "XYZ-1001", 5, "QTcF Int. (msec)", "Visit 2", "2016-03-08T09:51", 388, "Placebo",
  "XYZ-1001", 6, "QTcF Int. (msec)", "Visit 3", "2016-03-22T10:48", 394, "Placebo",
  "XYZ-1001", 7, "QTcF Int. (msec)", "Visit 3", "2016-03-22T10:51", 402, "Placebo",
  "XYZ-1002", 1, "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 399, "",
  "XYZ-1002", 2, "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410, "",
  "XYZ-1002", 3, "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392, "",
  "XYZ-1002", 4, "QTcF Int. (msec)", "Visit 2", "2016-03-06T09:53", 407, "Active 20mg",
  "XYZ-1002", 5, "QTcF Int. (msec)", "Visit 2", "2016-03-06T09:56", 400, "Active 20mg",
  "XYZ-1002", 6, "QTcF Int. (msec)", "Visit 3", "2016-03-24T10:53", 414, "Active 20mg",
  "XYZ-1002", 7, "QTcF Int. (msec)", "Visit 3", "2016-03-24T10:56", 402, "Active 20mg",
)

# Compute the average of AVAL only if there are more than 2 records within the
# by group
derive_summary_records(
  adeg,
  by_vars = vars(USUBJID, PARAM, AVISIT),
  filter = dplyr::n() > 2,
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = vars(DTYPE = "AVERAGE")
)

```

Description

Derives analysis age (AAGE) and analysis age unit (AAGEU)

Usage

```
derive_vars_age(  
  dataset,  
  start_date = BRTHDT,  
  end_date = RANDDT,  
  unit = "years"  
)
```

Arguments

dataset	Input dataset The columns specified by the start_date and the end_date parameter are expected.
start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Default: BRTHDT
end_date	The end date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Default: RANDDT
unit	Unit The age is derived in the specified unit Default: 'years' Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'

Details

The age is derived as the integer part of the duration from start to end date in the specified unit.

Value

The input dataset with AAGE and AAGEU added

Author(s)

Stefan Bundfuss

See Also

[derive_vars_duration\(\)](#)

Examples

```
data <- tibble::tribble(
  ~BRTHDT, ~RANDDT,
  lubridate::ymd("1984-09-06"), lubridate::ymd("2020-02-24")
)

derive_vars_aage(data)
```

derive_vars_atc	<i>Derive ATC Class Variables</i>
-----------------	-----------------------------------

Description

Add Anatomical Therapeutic Chemical class variables from FACM to ACDM

Usage

```
derive_vars_atc(
  dataset,
  dataset_facm,
  by_vars = vars(USUBJID, CMREFID = FAREFID)
)
```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter are required
dataset_facm	FACM dataset The variables specified by the <code>by_vars</code> parameter, <code>FAGRPID</code> , <code>FATESTCD</code> and <code>FASTRESC</code> are required
by_vars	Keys used to merge <code>dataset_facm</code> with <code>dataset</code> <i>Permitted Values:</i> list of variables

Value

The input dataset with ATC variables added

Author(s)

Thomas Neitmann

Examples

```

cm <- tibble::tribble(
  ~USUBJID, ~CMGRPID, ~CMREFID, ~CMDECOD,
  "BP40257-1001", "14", "1192056", "PARACETAMOL",
  "BP40257-1001", "18", "2007001", "SOLUMEDROL",
  "BP40257-1002", "19", "2791596", "SPIRONOLACTONE"
)
facm <- tibble::tribble(
  ~USUBJID, ~FAGRPID, ~FAREFID, ~FATESTCD, ~FASTRESC,
  "BP40257-1001", "1", "1192056", "CMATC1CD", "N",
  "BP40257-1001", "1", "1192056", "CMATC2CD", "N02",
  "BP40257-1001", "1", "1192056", "CMATC3CD", "N02B",
  "BP40257-1001", "1", "1192056", "CMATC4CD", "N02BE",
  "BP40257-1001", "1", "2007001", "CMATC1CD", "D",
  "BP40257-1001", "1", "2007001", "CMATC2CD", "D10",
  "BP40257-1001", "1", "2007001", "CMATC3CD", "D10A",
  "BP40257-1001", "1", "2007001", "CMATC4CD", "D10AA",
  "BP40257-1001", "2", "2007001", "CMATC1CD", "D",
  "BP40257-1001", "2", "2007001", "CMATC2CD", "D07",
  "BP40257-1001", "2", "2007001", "CMATC3CD", "D07A",
  "BP40257-1001", "2", "2007001", "CMATC4CD", "D07AA",
  "BP40257-1001", "3", "2007001", "CMATC1CD", "H",
  "BP40257-1001", "3", "2007001", "CMATC2CD", "H02",
  "BP40257-1001", "3", "2007001", "CMATC3CD", "H02A",
  "BP40257-1001", "3", "2007001", "CMATC4CD", "H02AB",
  "BP40257-1002", "1", "2791596", "CMATC1CD", "C",
  "BP40257-1002", "1", "2791596", "CMATC2CD", "C03",
  "BP40257-1002", "1", "2791596", "CMATC3CD", "C03D",
  "BP40257-1002", "1", "2791596", "CMATC4CD", "C03DA"
)

derive_vars_atc(cm, facm)

```

```
derive_vars_disposition_reason
```

Derive a Disposition Reason at a Specific Timepoint

Description

Derive a disposition reason from the the relevant records in the disposition domain.

Usage

```

derive_vars_disposition_reason(
  dataset,
  dataset_ds,
  new_var,
  reason_var,
  new_var_spe = NULL,

```

```

reason_var_spe = NULL,
format_new_vars = format_reason_default,
filter_ds,
subject_keys = vars(STUDYID, USUBJID)
)

```

Arguments

dataset	Input dataset
dataset_ds	Dataset containing the disposition information (e.g. ds) The dataset must contain: <ul style="list-style-type: none"> • STUDYID, USUBJID, • The variable(s) specified in the reason_var (and reason_var_spe, if required) • The variables used in filter_ds.
new_var	Name of the disposition reason variable A variable name is expected (e.g. DCSREAS).
reason_var	The variable used to derive the disposition reason A variable name is expected (e.g. DSDECOD).
new_var_spe	Name of the disposition reason detail variable A variable name is expected (e.g. DCSREASP). If new_var_spe is specified, it is expected that reason_var_spe is also specified, otherwise an error is issued. Default: NULL
reason_var_spe	The variable used to derive the disposition reason detail A variable name is expected (e.g. DSTERM). If new_var_spe is specified, it is expected that reason_var_spe is also specified, otherwise an error is issued. Default: NULL
format_new_vars	The function used to derive the reason(s) This function is used to derive the disposition reason(s) and must follow the below conventions <ul style="list-style-type: none"> • If only the main reason for discontinuation needs to be derived (i.e. new_var_spe is NULL), the function must have at least one character vector argument, e.g. <code>format_reason <- function(reason)</code> and new_var will be derived as <code>new_var = format_reason(reason_var)</code>. Typically, the content of the function would return reason_var or NA depending on the value (e.g. <code>if_else (reason != "COMPLETED" & !is.na(reason), reason, NA_character_)</code>). <code>DCSREAS = format_reason(DSDECOD)</code> returns <code>DCSREAS = DSDECOD</code> when <code>DSDECOD</code> is not 'COMPLETED' nor NA, NA otherwise. • If both the main reason and the details needs to be derived (new_var_spe is specified) the function must have two character vectors argument, e.g. <code>format_reason2 <- function(reason, reason_spe)</code> and new_var will be derived as <code>new_var = format_reason(reason_var)</code>, new_var_spe will be derived as <code>new_var_spe = format_reason(reason_var, reason_var_spe)</code>. Typically, the content of the function would return reason_var_spe or NA

depending on the reason_var value (e.g. if_else (reason == "OTHER", reason_spe, NA_character_)). DCSREASP = format_reason(DSDECOD, DSTERM) returns DCSREASP = DSTERM when DSDECOD is equal to 'OTHER'.

Default: format_reason_default, see [format_reason_default\(\)](#) for details.

filter_ds	Filter condition for the disposition data. Filter used to select the relevant disposition data. It is expected that the filter restricts dataset_ds such that there is at most one observation per patient. An error is issued otherwise. Permitted Values: logical expression.
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by vars() is expected.

Details

This functions returns the main reason for discontinuation (e.g. DCSREAS or DCTREAS). The reason for discontinuation is derived based on reason_var (e.g. DSDECOD) and format_new_vars. If new_var_spe is not NULL, then the function will also return the details associated with the reason for discontinuation (e.g. DCSREASP). The details associated with the reason for discontinuation are derived based on reason_var_spe (e.g. DSTERM), reason_var and format_new_vars.

Value

the input dataset with the disposition reason(s) (new_var and if required new_var_spe) added.

Author(s)

Samia Kabi

See Also

[format_reason_default\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ds")

# Derive DCSREAS using the default format
admiral_dm %>%
  derive_vars_disposition_reason(
    dataset_ds = admiral_ds,
    new_var = DCSREAS,
    reason_var = DSDECOD,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, DCSREAS)
```

```

# Derive DCSREAS and DCSREASP using a study-specific format
format_dcsreas <- function(x, y = NULL) {
  if (is.null(y)) {
    if_else(!x %in% c("COMPLETED", "SCREEN FAILURE") & !is.na(x), x, NA_character_)
  } else {
    if_else(x == "OTHER", y, NA_character_)
  }
}
}
admiral_dm %>%
  derive_vars_disposition_reason(
    dataset_ds = admiral_ds,
    new_var = DCSREAS,
    reason_var = DSDECOD,
    new_var_spe = DCSREASP,
    reason_var_spe = DSTERM,
    format_new_vars = format_dcsreas,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, DCSREAS, DCSREASP)

```

 derive_vars_dt

Derive/Impute a Date from a Date Character Vector

Description

Derive a date ('--DT') from a date character vector ('--DTC'). The date can be imputed (see `date_imputation` parameter) and the date imputation flag ('--DTF') can be added.

Usage

```

derive_vars_dt(
  dataset,
  new_vars_prefix,
  dtc,
  date_imputation = NULL,
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)

```

Arguments

dataset	Input dataset.
	The date character vector (dtc) must be present.

new_vars_prefix	<p>Prefix used for the output variable(s).</p> <p>A character scalar is expected. For the date variable "DT" is appended to the specified prefix and for the date imputation flag "DTF". I.e., for new_vars_prefix = "AST" the variables ASTDT and ASTDTF are created.</p>
dtc	<p>The '--DTC' date to impute</p> <p>A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. If the year part is not recorded (missing date), no imputation is performed.</p>
date_imputation	<p>The value to impute the day/month when a datepart is missing.</p> <p>If NULL: no date imputation is performed and partial dates are returned as missing.</p> <p>Otherwise, a character value is expected, either as a</p> <ul style="list-style-type: none"> • format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June, • or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month. <p>Default is NULL.</p>
flag_imputation	<p>Whether the date imputation flag must also be derived.</p> <p>If "auto" is specified, the date imputation flag is derived if the date_imputation parameter is not null.</p> <p><i>Default:</i> "auto"</p> <p><i>Permitted Values:</i> "auto", "date" or "none"</p>
min_dates	<p>Minimum dates</p> <p>A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example</p> <pre> impute_dtc("2020-11", min_dates = list(ymd_hms("2020-12-06T12:12:12"), ymd_hms("2020-11-11T11:11:11")), date_imputation = "first") </pre> <p>returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).</p>

max_dates	<p>Maximum dates</p> <p>A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.</p>
preserve	<p>Preserve day if month is missing and day is present</p> <p>For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").</p> <p>Permitted Values: TRUE, FALSE</p> <p>Default: FALSE</p>

Details

The presence of a '--DTF' variable is checked and if it already exists in the input dataset, a warning is issued and '--DTF' will be overwritten.

Value

The input dataset with the date '--DT' (and the date imputation flag '--DTF' if requested) added.

Author(s)

Samia Kabi

Examples

```
library(lubridate)

mhdt <- tibble::tribble(
  ~MHSTDTC,
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

# Create ASTDT and ASTDTF
# no imputation for partial date
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC
)

# Create ASTDT and ASTDTF
# Impute partial dates to first day/month
derive_vars_dt(
  mhdt,
```

```

    new_vars_prefix = "AST",
    dtc = MHSTDTC,
    date_imputation = "FIRST"
  )

# Impute partial dates to 6th of April
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  date_imputation = "04-06"
)

# Create AENDT and AENDTF
# Impute partial dates to last day/month
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AEN",
  dtc = MHSTDTC,
  date_imputation = "LAST"
)

# Create BIRTHDT
# Impute partial dates to 15th of June. No DTF
derive_vars_dt(
  mhdt,
  new_vars_prefix = "BIRTH",
  dtc = MHSTDTC,
  date_imputation = "MID",
  flag_imputation = "none"
)

# Impute AE start date to the first date and ensure that the imputed date
# is not before the treatment start date
adae <- tibble::tribble(
  ~AESTDTC, ~TRTSDTM,
  "2020-12", ymd_hms("2020-12-06T12:12:12"),
  "2020-11", ymd_hms("2020-12-06T12:12:12")
)

derive_vars_dt(
  adae,
  dtc = AESTDTC,
  new_vars_prefix = "AST",
  date_imputation = "first",
  min_dates = vars(TRTSDTM)
)

# A user imputing dates as middle month/day, i.e. date_imputation = "MID" can
# use preserve argument to "preserve" partial dates. For example, "2019--07",
# will be displayed as "2019-06-07" rather than 2019-06-15 with preserve = TRUE

derive_vars_dt(

```

```

mhd,
new_vars_prefix = "AST",
dtc = MHSTDTC,
date_imputation = "MID",
preserve = TRUE
)

```

derive_vars_dtm *Derive/Impute a Datetime from a Date Character Vector*

Description

Derive a datetime object ('--DTM') from a date character vector ('--DTC'). The date and time can be imputed (see date_imputation/time_imputation parameters) and the date/time imputation flag ('--DTF', '--TMF') can be added.

Usage

```

derive_vars_dtm(
  dataset,
  new_vars_prefix,
  dtc,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE,
  ignore_seconds_flag = FALSE
)

```

Arguments

dataset	Input dataset The date character vector (dtc) must be present.
new_vars_prefix	Prefix used for the output variable(s). A character scalar is expected. For the date variable "DT" is appended to the specified prefix, for the date imputation flag "DTF", and for the time imputation flag "TMF". I.e., for new_vars_prefix = "AST" the variables ASTDT, ASTDTF, and ASTTMF are created.
dtc	The '--DTC' date to impute A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. If the year part is not recorded (missing date), no imputation is performed.

date_imputation

The value to impute the day/month when a datepart is missing.

If NULL: no date imputation is performed and partial dates are returned as missing.

Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,
- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "FIRST", "LAST" to impute to the start/end of a day.

Default is "00:00:00".

flag_imputation

Whether the date/time imputation flag(s) must also be derived.

If "auto" is specified, the date imputation flag is derived if the date_imputation parameter is not null and the time imputation flag is derived if the time_imputation parameter is not null

Default: "auto"

Permitted Values: "auto", "date", "time", "both", or "none"

min_dates

Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  date_imputation = "first"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

max_dates	<p>Maximum dates</p> <p>A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.</p>
preserve	<p>Preserve day if month is missing and day is present</p> <p>For example "2019---07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").</p> <p>Permitted Values: TRUE, FALSE</p> <p>Default: FALSE</p>
ignore_seconds_flag	<p>ADaM IG states that given SDTM ('--DTC') variable, if only hours and minutes are ever collected, and seconds are imputed in ('--DTM') as 00, then it is not necessary to set ('--TMF') to 'S'. A user can set this to TRUE so the 'S' Flag is dropped from ('--TMF').</p> <p>A logical value</p> <p>Default: FALSE</p>

Details

The presence of a '--DTF' variable is checked and the variable is not derived if it already exists in the input dataset. However, if '--TMF' already exists in the input dataset, a warning is issued and '--TMF' will be overwritten.

Value

The input dataset with the datetime '--DTM' (and the date/time imputation flag '--DTF', '--TMF') added.

Author(s)

Samia Kabi

Examples

```
library(lubridate)

mhdt <- tibble::tribble(
  ~MHSTDTC,
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dtm(
```



```

    mhdt,
    new_vars_prefix = "AST",
    dtc = MHSTDTC,
    date_imputation = "FIRST",
    time_imputation = "FIRST"
  )

# Impute AE end date to the last date and ensure that the imputed date is not
# after the death or data cut off date
adae <- tibble::tribble(
  ~AEENDTC, ~DTHDT, ~DCUTDT,
  "2020-12", ymd("2020-12-06"), ymd("2020-12-24"),
  "2020-11", ymd("2020-12-06"), ymd("2020-12-24")
)

derive_vars_dtm(
  adae,
  dtc = AEENDTC,
  new_vars_prefix = "AEN",
  date_imputation = "last",
  time_imputation = "last",
  max_dates = vars(DTHDT, DCUTDT)
)

# Seconds has been removed from the input dataset. Function now uses
# ignore_seconds_flag to remove the 'S' from the --TMF variable.
mhdt <- tibble::tribble(
  ~MHSTDTC,
  "2019-07-18T15:25",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  date_imputation = "FIRST",
  time_imputation = "FIRST",
  ignore_seconds_flag = TRUE
)

# A user imputing dates as middle month/day, i.e. date_imputation = "MID" can
# use preserve argument to "preserve" partial dates. For example, "2019---07",
# will be displayed as "2019-06-07" rather than 2019-06-15 with preserve = TRUE

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",

```

```

dtc = MHSTDTC,
date_imputation = "MID",
preserve = TRUE
)

```

derive_vars_dtm_to_dt *Derive Date Variables from Datetime Variables*

Description

This function creates date(s) as output from datetime variable(s)

Usage

```
derive_vars_dtm_to_dt(dataset, source_vars)
```

Arguments

dataset	Input dataset
source_vars	A list of datetime variables created using vars() from which dates are to be extracted

Value

A data frame containing the input dataset with the corresponding date (--DT) variable(s) of all datetime variables (--DTM) specified in source_vars.

Author(s)

Teckla Akinyi

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adcm <- tibble::tribble(
  ~USUBJID, ~TRTSDTM,          ~ASTDTM,          ~AENDTM,
  "PAT01",  "2012-02-25 23:00:00", "2012-02-28 19:00:00", "2012-02-25 23:00:00",
  "PAT01",  NA,                "2012-02-28 19:00:00", NA,
  "PAT01",  "2017-02-25 23:00:00", "2013-02-25 19:00:00", "2014-02-25 19:00:00",
  "PAT01",  "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-03-25 23:00:00",
  "PAT01",  "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-04-29 14:00:00",
) %>%
mutate(
  TRTSDTM = as_datetime(TRTSDTM),
  ASTDTM = as_datetime(ASTDTM),
  AENDTM = as_datetime(AENDTM)
)

```

```
adcm %>%
  derive_vars_dtm_to_dt(vars(TRTSDTM, ASTDTM, AENDTM)) %>%
  select(USUBJID, starts_with("TRT"), starts_with("AST"), starts_with("AEN"))
```

derive_vars_dtm_to_tm *Derive Time Variables from Datetime Variables*

Description

This function creates time variable(s) as output from datetime variable(s)

Usage

```
derive_vars_dtm_to_tm(dataset, source_vars)
```

Arguments

dataset	Input dataset
source_vars	A list of datetime variables created using vars() from which time is to be extracted

Details

The names of the newly added variables are automatically set by replacing the --DTM suffix of the source_vars with --TM. The --TM variables are created using the hms package.

Value

A data frame containing the input dataset with the corresponding time (--TM) variable(s) of all datetime variables (--DTM) specified in source_vars with the correct name.

Author(s)

Teckla Akinyi

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adcm <- tibble::tribble(
  ~USUBJID, ~TRTSDTM, ~ASTDTM, ~AENDTM,
  "PAT01", "2012-02-25 23:41:10", "2012-02-28 19:03:00", "2013-02-25 23:32:16",
  "PAT01", "", "2012-02-28 19:00:00", "",
  "PAT01", "2017-02-25 23:00:02", "2013-02-25 19:00:15", "2014-02-25 19:00:56",
  "PAT01", "2017-02-25 16:00:00", "2017-02-25 14:25:00", "2017-03-25 23:00:00",
  "PAT01", "2017-02-25 16:05:17", "2017-02-25 14:20:00", "2018-04-29 14:06:45",
) %>%
```

```

mutate(
  TRTSDTM = as_datetime(TRTSDTM),
  ASTDTM = as_datetime(ASTDTM),
  AENDTM = as_datetime(AENDTM)
)

adcm %>%
  derive_vars_dtm_to_tm(vars(TRTSDTM)) %>%
  select(USUBJID, starts_with("TRT"), everything())

adcm %>%
  derive_vars_dtm_to_tm(vars(TRTSDTM, ASTDTM, AENDTM)) %>%
  select(USUBJID, starts_with("TRT"), starts_with("AS"), starts_with("AE"))

```

derive_vars_duration *Derive Duration*

Description

Derives duration between two dates, specified by the variables present in input dataset e.g., duration of adverse events, relative day, age, ...

Usage

```

derive_vars_duration(
  dataset,
  new_var,
  new_var_unit = NULL,
  start_date,
  end_date,
  in_unit = "days",
  out_unit = "days",
  floor_in = TRUE,
  add_one = TRUE,
  trunc_out = FALSE
)

```

Arguments

dataset	Input dataset The variables specified by the start_date and the end_date parameter are expected.
new_var	Name of variable to create
new_var_unit	Name of the unit variable If the parameter is not specified, no variable for the unit is created.

start_date	<p>The start date</p> <p>A date or date-time variable is expected. This variable must be present in specified input dataset.</p> <p>Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.</p>
end_date	<p>The end date</p> <p>A date or date-time variable is expected. This variable must be present in specified input dataset.</p> <p>Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.</p>
in_unit	<p>Input unit</p> <p>See <code>floor_in</code> and <code>add_one</code> parameter for details.</p> <p>Default: 'days'</p> <p>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'</p>
out_unit	<p>Output unit</p> <p>The duration is derived in the specified unit</p> <p>Default: 'days'</p> <p>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'</p>
floor_in	<p>Round down input dates?</p> <p>The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.</p> <p>Default: 'TRUE'</p> <p>Permitted Values: TRUE, FALSE</p>
add_one	<p>Add one input unit?</p> <p>If the duration is non-negative, one input unit is added. I.e., the duration can not be zero.</p> <p>Default: TRUE Permitted Values: TRUE, FALSE</p>
trunc_out	<p>Return integer part</p> <p>The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.</p> <p>Default: FALSE</p> <p>Permitted Values: TRUE, FALSE</p>

Details

The duration is derived as time from start to end date in the specified output unit. If the end date is before the start date, the duration is negative. The start and end date variable must be present in the specified input dataset.

Value

The input dataset with the duration and unit variable added

Author(s)

Stefan Bundfuss

See Also[compute_duration\(\)](#)**Examples**

```

data <- tibble::tribble(
  ~BRTHDT, ~RANDDT,
  lubridate::ymd("1984-09-06"), lubridate::ymd("2020-02-24")
)

derive_vars_duration(data,
  new_var = AAGE,
  new_var_unit = AAGEU,
  start_date = BRTHDT,
  end_date = RANDDT,
  out_unit = "years",
  add_one = FALSE,
  trunc_out = TRUE
)

```

`derive_vars_dy`*Derive Relative Day Variables*

Description

Adds relative day variables (--DY) to the dataset, e.g., ASTDY and AENDY.

Usage

```
derive_vars_dy(dataset, reference_date, source_vars)
```

Arguments

<code>dataset</code>	Input dataset The columns specified by the <code>reference_date</code> and the <code>source_vars</code> parameter are expected.
<code>reference_date</code>	The start date column, e.g., date of first treatment A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.
<code>source_vars</code>	A list of datetime or date variables created using <code>vars()</code> from which dates are to be extracted. This can either be a list of date(time) variables or named --DY variables and corresponding -DT(M) variables e.g. <code>vars(TRTSDTM, ASTDTM, AENDT)</code> or <code>vars(TRTSDT, ASTDTM, AENDT, DEATHDY = DTHDT)</code> . If the source variable does not end in -DT(M), a name for the resulting --DY variable must be provided.

Details

The relative day is derived as number of days from the reference date to the end date. If it is nonnegative, one is added. I.e., the relative day of the reference date is 1. Unless a name is explicitly specified, the name of the resulting relative day variable is generated from the source variable name by replacing DT (or DTM as appropriate) with DY.

Value

The input dataset with --DY corresponding to the --DTM or --DT source variable(s) added

Author(s)

Teckla Akinyi

Examples

```
library(lubridate)
library(dplyr)

datain <- tibble::tribble(
  ~TRTSDTM, ~ASTDTM, ~AENDT,
  "2014-01-17T23:59:59", "2014-01-18T13:09:09", "2014-01-20"
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDT = ymd(AENDT)
  )

derive_vars_dy(
  datain,
  reference_date = TRTSDTM,
  source_vars = vars(TRTSDTM, ASTDTM, AENDT)
)

# specifying name of new variables
datain <- tibble::tribble(
  ~TRTSDT, ~DTHDT,
  "2014-01-17", "2014-02-01"
) %>%
  mutate(
    TRTSDT = ymd(TRTSDT),
    DTHDT = ymd(DTHDT)
  )

derive_vars_dy(
  datain,
  reference_date = TRTSDT,
  source_vars = vars(TRTSDT, DEATHDY = DTHDT)
)
```

 derive_vars_last_dose *Derive Last Dose*

Description

Add EX source variables from last dose to the input dataset.

Usage

```
derive_vars_last_dose(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_vars = NULL,
  traceability_vars = NULL
)
```

Arguments

dataset	Input dataset. The variables specified by the <code>by_vars</code> and <code>analysis_date</code> parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the <code>by_vars</code> , <code>dose_date</code> , <code>new_vars</code> parameters, and source variables from <code>traceability_vars</code> parameter are expected.
filter_ex	Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to <code>NULL</code> .
by_vars	Variables to join by (created by <code>dplyr::vars</code>).
dose_id	Variables to identify unique dose (created by <code>dplyr::vars</code>). Defaults to empty <code>vars()</code> .
dose_date	The EX dose date variable. A date or date-time object is expected.
analysis_date	The analysis date variable. A date or date-time object is expected.
single_dose_condition	The condition for checking if <code>dataset_ex</code> is single dose. An error is issued if the condition is not true. Defaults to <code>(EXDOSFRQ == "ONCE")</code> .
new_vars	Variables to keep from <code>dataset_ex</code> , with the option to rename. Can either be variables created by <code>dplyr::vars</code> (e.g. <code>vars(VISIT)</code>), or named list returned by <code>vars()</code> (e.g. <code>vars(LSTEXVIS = VISIT)</code>). If set to <code>NULL</code> , then all variables from <code>dataset_ex</code> are kept without renaming. Defaults to <code>NULL</code> .

traceability_vars

A named list returned by `vars()` listing the traceability variables, e.g. `vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ)`. The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Details

All date (date-time) variables can be characters in standard ISO format or of date / date-time class. For ISO format, see `impute_dtc` - parameter `dtc` for further details. When doing date comparison to identify last dose, date-time imputations are done as follows:

- `dose_date`: no date imputation, time imputation to `00:00:00` if time is missing.
- `analysis_date`: no date imputation, time imputation to `23:59:59` if time is missing.

The last dose records are identified as follows:

1. The `dataset_ex` is filtered using `filter_ex`, if provided. This is useful for, for example, filtering for valid dose only.
2. The datasets `dataset` and `dataset_ex` are joined using `by_vars`.
3. The last dose is identified: the last dose is the EX record with maximum date where `dose_date` is lower to or equal to `analysis_date`, subject to both date values are non-NA. The last dose is identified per `by_vars`. If multiple EX records exist for the same `dose_date`, then either `dose_id` needs to be supplied (e.g. `dose_id = vars(EXSEQ)`) to identify unique records, or an error is issued. When `dose_id` is supplied, the last EX record from the same `dose_date` sorted by `dose_id` will be used to identify last dose.
4. The EX source variables (as specified in `new_vars`) from last dose are appended to the dataset and returned to the user.

This function only works correctly for EX dataset with a structure of single dose per row. If your study EX dataset has multiple doses per row, use `expansion_function_name??` to transform the EX dataset into single dose per row structure before calling `derive_vars_last_dose`.

If variables (other than those specified in `by_vars`) exist in both `dataset` and `dataset_ex`, then join cannot be performed properly and an error is issued. To resolve the error, use `new_vars` to either keep variables unique to `dataset_ex`, or use this option to rename variables from `dataset_ex` (e.g. `new_vars = vars(LSTEXVIS = VISIT)`).

Value

Input dataset with EX source variables from last dose added.

Author(s)

Ondrej Slama, Annie Yang

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(ex_single)

admiral_ae %>%
  head(100) %>%
  derive_vars_last_dose(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    new_vars = vars(EXDOSE, EXTRT, EXSEQ, EXENDTC, VISIT),
    dose_date = EXENDTC,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC)
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, EXDOSE, EXTRT, EXENDTC, EXSEQ, VISIT)

# or with traceability variables
admiral_ae %>%
  head(100) %>%
  derive_vars_last_dose(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    new_vars = vars(EXDOSE, EXTRT, EXSEQ, EXENDTC, VISIT),
    dose_date = EXENDTC,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC),
    traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXENDTC")
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, EXDOSE, EXTRT, EXENDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR)

```

derive_vars_merged	<i>Add New Variable(s) to the Input Dataset Based on Variables from Another Dataset</i>
--------------------	---

Description

Add new variable(s) to the input dataset based on variables from another dataset. The observations to merge can be selected by a condition (`filter_add` argument) and/or selecting the first or last observation for each by group (`order` and `mode` argument).

Usage

```

derive_vars_merged(
  dataset,
  dataset_add,

```

```

    by_vars,
    order = NULL,
    new_vars = NULL,
    mode = NULL,
    filter_add = NULL,
    match_flag = NULL,
    check_type = "warning",
    duplicate_msg = NULL
)

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter are expected.</p>
dataset_add	<p>Additional dataset</p> <p>The variables specified by the <code>by_vars</code>, the <code>new_vars</code>, and the <code>order</code> parameter are expected.</p>
by_vars	<p>Grouping variables</p> <p>The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations.</p> <p><i>Permitted Values:</i> list of variables created by <code>vars()</code></p>
order	<p>Sort order</p> <p>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> list of variables or <code>desc(<variable>)</code> function calls created by <code>vars()</code>, e.g., <code>vars(ADT, desc(AVAL))</code> or NULL</p>
new_vars	<p>Variables to add</p> <p>The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., <code>new_vars = vars(<new name> = <old name>)</code>. For example <code>new_vars = vars(var1, var2)</code> adds variables <code>var1</code> and <code>var2</code> from <code>dataset_add</code> to the input dataset.</p> <p>And <code>new_vars = vars(var1, new_var2 = old_var2)</code> takes <code>var1</code> and <code>old_var2</code> from <code>dataset_add</code> and adds them to the input dataset renaming <code>old_var2</code> to <code>new_var2</code>.</p> <p>If the parameter is not specified or set to NULL, all variables from the additional dataset (<code>dataset_add</code>) are added.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> list of variables created by <code>vars()</code></p>
mode	<p>Selection mode</p> <p>Determines if the first or last observation is selected. If the <code>order</code> parameter is specified, <code>mode</code> must be non-null.</p> <p>If the <code>order</code> parameter is not specified, the <code>mode</code> parameter is ignored.</p>

	<i>Default:</i> NULL
	<i>Permitted Values:</i> "first", "last", NULL
filter_add	Filter for additional dataset (dataset_add) Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered. <i>Default:</i> NULL <i>Permitted Values:</i> a condition
match_flag	Match flag If the parameter is specified (e.g., match_flag = FLAG), the specified variable (e.g., FLAG) is added to the input dataset. This variable will be TRUE for all selected records from dataset_add which are merged into the input dataset, and NA otherwise. <i>Default:</i> NULL <i>Permitted Values:</i> Variable name
check_type	Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order. <i>Default:</i> "warning" <i>Permitted Values:</i> "none", "warning", "error"
duplicate_msg	Message of unique check If the uniqueness check fails, the specified message is displayed. <i>Default:</i> <pre>paste("Dataset `dataset_add` contains duplicate records with respect to", enumerate(vars2chr(by_vars)))</pre>

Details

1. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
2. If order is specified, for each by group the first or last observation (depending on mode) is selected.
3. The variables specified for new_vars are renamed (if requested) and merged to the input dataset using left_join(). I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the additional dataset the new variables are set to NA. Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars from the additional dataset (dataset_add).

Author(s)

Stefan Bundfuss

Examples

```

library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_vs")
data("admiral_dm")

# merging all dm variables to vs
derive_vars_merged(
  admiral_vs,
  dataset_add = select(admiral_dm, -DOMAIN),
  by_vars = vars(STUDYID, USUBJID)
) %>%
  select(STUDYID, USUBJID, VSTESTCD, VISIT, VSTPT, VSSTRESN, AGE, AGEU)

# merge last weight to adsl
data("admiral_adsl")
derive_vars_merged(
  admiral_adsl,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  order = vars(VSDTC),
  mode = "last",
  new_vars = vars(LASTWGT = VSSTRESN, LASTWGTU = VSSTRESU),
  filter_add = VSTESTCD == "WEIGHT",
  match_flag = vsdatafl
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, LASTWGT, LASTWGTU, vsdatafl)

```

derive_vars_merged_dt *Merge a (Imputed) Date Variable*

Description

Merge a imputed date variable and date imputation flag from a dataset to the input dataset. The observations to merge can be selected by a condition and/or selecting the first or last observation for each by group.

Usage

```

derive_vars_merged_dt(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars_prefix,
  filter_add = NULL,
  mode = NULL,
  dtc,
  date_imputation = NULL,

```

```

    flag_imputation = "auto",
    min_dates = NULL,
    max_dates = NULL,
    preserve = FALSE,
    check_type = "warning",
    duplicate_msg = NULL
  )

```

Arguments

dataset	<p>Input dataset</p> <p>The variables specified by the <code>by_vars</code> parameter are expected.</p>
dataset_add	<p>Additional dataset</p> <p>The variables specified by the <code>by_vars</code>, the <code>dtc</code>, and the <code>order</code> parameter are expected.</p>
by_vars	<p>Grouping variables</p> <p>The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations.</p> <p><i>Permitted Values:</i> list of variables created by <code>vars()</code></p>
order	<p>Sort order</p> <p>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. The imputed date variable can be specified as well (see examples below). Please note that NA is considered as the last value. I.e., if a order variable is NA and <code>mode = "last"</code>, this observation is chosen while for <code>mode = "first"</code> the observation is chosen only if there are no observations where the variable is not 'NA'.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> list of variables or <code>desc(<variable>)</code> function calls created by <code>vars()</code>, e.g., <code>vars(ADT, desc(AVAL))</code> or NULL</p>
new_vars_prefix	<p>Prefix used for the output variable(s).</p> <p>A character scalar is expected. For the date variable "DT" is appended to the specified prefix and for the date imputation flag "DTF". I.e., for <code>new_vars_prefix = "AST"</code> the variables <code>ASTDT</code> and <code>ASTDTF</code> are created.</p>
filter_add	<p>Filter for additional dataset (<code>dataset_add</code>)</p> <p>Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> a condition</p>
mode	<p>Selection mode</p> <p>Determines if the first or last observation is selected. If the <code>order</code> parameter is specified, <code>mode</code> must be non-null.</p> <p>If the <code>order</code> parameter is not specified, the <code>mode</code> parameter is ignored.</p>

	<i>Default:</i> NULL
	<i>Permitted Values:</i> "first", "last", NULL
dtc	The '--DTC' date to impute A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. If the year part is not recorded (missing date), no imputation is performed.
date_imputation	The value to impute the day/month when a datepart is missing. If NULL: no date imputation is performed and partial dates are returned as missing. Otherwise, a character value is expected, either as a <ul style="list-style-type: none"> • format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June, • or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.
flag_imputation	Default is NULL. Whether the date imputation flag must also be derived. If "auto" is specified, the date imputation flag is derived if the date_imputation parameter is not null. <i>Default:</i> "auto" <i>Permitted Values:</i> "auto", "date" or "none"
min_dates	Minimum dates A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example <pre> impute_dtc("2020-11", min_dates = list(ymd_hms("2020-12-06T12:12:12"), ymd_hms("2020-11-11T11:11:11")), date_imputation = "first") </pre> returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).
max_dates	Maximum dates A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

preserve	<p>Preserve day if month is missing and day is present For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID"). Permitted Values: TRUE, FALSE Default: FALSE</p>
check_type	<p>Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order. <i>Default: "warning"</i> <i>Permitted Values: "none", "warning", "error"</i></p>
duplicate_msg	<p>Message of unique check If the uniqueness check fails, the specified message is displayed. <i>Default:</i> <pre>paste("Dataset `dataset_add` contains duplicate records with respect to", enumerate(vars2chr(by_vars)))</pre></p>

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The date variable and if requested, the date imputation flag is added to the additional dataset.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The date and flag variables are merged to the input dataset.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable <new_vars_prefix>DT and optionally the variable <new_vars_prefix>DTF derived from the additional dataset (dataset_add).

Author(s)

Stefan Bundfuss

Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_ex")

# derive treatment start date (TRTSDT)
derive_vars_merged_dt(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = admiral_ex,
  by_vars = vars(STUDYID, USUBJID),
```



```

    new_vars_prefix = "TRTS",
    dtc = EXSTDTC,
    date_imputation = "first",
    order = vars(TRTSDT),
    mode = "first"
  )

# derive treatment end date (TRTEDT) (without imputation)
derive_vars_merged_dt(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = admiral_ex,
  by_vars = vars(STUDYID, USUBJID),
  new_vars_prefix = "TRTE",
  dtc = EXENDTC,
  order = vars(TRTEDT),
  mode = "last"
)

```

```
derive_vars_merged_dtm
```

Merge a (Imputed) Datetime Variable

Description

Merge a imputed datetime variable, date imputation flag, and time imputation flag from a dataset to the input dataset. The observations to merge can be selected by a condition and/or selecting the first or last observation for each by group.

Usage

```

derive_vars_merged_dtm(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars_prefix,
  filter_add = NULL,
  mode = NULL,
  dtc,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE,
  check_type = "warning",
  duplicate_msg = NULL
)

```

Arguments

dataset	Input dataset The variables specified by the by_vars parameter are expected.
dataset_add	Additional dataset The variables specified by the by_vars, the dtc, and the order parameter are expected.
by_vars	Grouping variables The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. <i>Permitted Values:</i> list of variables created by vars()
order	Sort order If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. The imputed datetime variable can be specified as well (see examples below). <i>Default:</i> NULL <i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) or NULL
new_vars_prefix	Prefix used for the output variable(s). A character scalar is expected. For the date variable "DT" is appended to the specified prefix, for the date imputation flag "DTF", and for the time imputation flag "TMF". I.e., for new_vars_prefix = "AST" the variables ASTDT, ASTDTF, and ASTTMF are created.
filter_add	Filter for additional dataset (dataset_add) Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered. <i>Default:</i> NULL <i>Permitted Values:</i> a condition
mode	Selection mode Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null. If the order parameter is not specified, the mode parameter is ignored. <i>Default:</i> NULL <i>Permitted Values:</i> "first", "last", NULL
dtc	The '--DTC' date to impute A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. If the year part is not recorded (missing date), no imputation is performed.
date_imputation	The value to impute the day/month when a datepart is missing. If NULL: no date imputation is performed and partial dates are returned as missing. Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,
- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "FIRST", "LAST" to impute to the start/end of a day.

Default is "00:00:00".

flag_imputation

Whether the date/time imputation flag(s) must also be derived.

If "auto" is specified, the date imputation flag is derived if the date_imputation parameter is not null and the time imputation flag is derived if the time_imputation parameter is not null

Default: "auto"

Permitted Values: "auto", "date", "time", "both", or "none"

min_dates

Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  date_imputation = "first"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

max_dates

Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

preserve	<p>Preserve day if month is missing and day is present For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID"). Permitted Values: TRUE, FALSE Default: FALSE</p>
check_type	<p>Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order. <i>Default: "warning"</i> <i>Permitted Values: "none", "warning", "error"</i></p>
duplicate_msg	<p>Message of unique check If the uniqueness check fails, the specified message is displayed. <i>Default:</i></p> <pre>paste("Dataset `dataset_add` contains duplicate records with respect to", enumerate(vars2chr(by_vars)))</pre>

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The datetime variable and if requested, the date imputation flag and time imputation flag is added to the additional dataset.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The date and flag variables are merged to the input dataset.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable <new_vars_prefix>DT and optionally the variables <new_vars_prefix>DTF and <new_vars_prefix>TMF derived from the additional dataset (dataset_add).

Author(s)

Stefan Bundfuss

Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_ex")

# derive treatment start datetime (TRTSDTM)
derive_vars_merged_dtm(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = admiral_ex,
```

```

    by_vars = vars(STUDYID, USUBJID),
    new_vars_prefix = "TRTS",
    dtc = EXSTDTC,
    date_imputation = "first",
    time_imputation = "first",
    order = vars(TRTSDTM),
    mode = "first"
  )

# derive treatment end datetime (TRTEDTM) (without date imputation)
derive_vars_merged_dtm(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = admiral_ex,
  by_vars = vars(STUDYID, USUBJID),
  new_vars_prefix = "TRTE",
  dtc = EXENDTC,
  time_imputation = "last",
  order = vars(TRTEDTM),
  mode = "last"
)

```

derive_vars_query *Derive Query Variables*

Description

Derive Query Variables

Usage

```
derive_vars_query(dataset, dataset_queries)
```

Arguments

dataset Input dataset.

dataset_queries

A dataset containing required columns VAR_PREFIX, QUERY_NAME, TERM_LEVEL, TERM_NAME, TERM_ID, and optional columns QUERY_ID, QUERY_SCOPE, QUERY_SCOPE_NUM.

The content of the dataset will be verified by [assert_valid_queries\(\)](#).

[create_query_data\(\)](#) can be used to create the dataset.

Details

For each unique element in VAR_PREFIX, the corresponding "NAM" variable will be created. For each unique VAR_PREFIX, if QUERY_ID is not "" or NA, then the corresponding "CD" variable is created; similarly, if QUERY_SCOPE is not "" or NA, then the corresponding "SC" variable will be created; if QUERY_SCOPE_NUM is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in dataset, the "NAM" variable takes the value of QUERY_NAME if the value of TERM_NAME or TERM_ID in dataset_queries matches the value of the respective TERM_LEVEL in dataset. Note that TERM_NAME in dataset_queries dataset may be NA only when TERM_ID is non-NA and vice versa. The "CD", "SC", and "SCN" variables are derived accordingly based on QUERY_ID, QUERY_SCOPE, and QUERY_SCOPE_NUM respectively, whenever not missing.

Value

The input dataset with query variables derived.

Author(s)

Ondrej Slama, Shimeng Huang

See Also

[create_query_data\(\)](#) [assert_valid_queries\(\)](#)

Examples

```
data("queries")
adae <- tibble::tribble(
  ~USUBJID, ~ASTDTM, ~AETERM, ~AESEQ, ~AEDECOD, ~AELLT, ~AELLTCD,
  "01", "2020-06-02 23:59:59", "ALANINE AMINOTRANSFERASE ABNORMAL",
  3, "Alanine aminotransferase abnormal", NA_character_, NA_integer_,
  "02", "2020-06-05 23:59:59", "BASEDOW'S DISEASE",
  5, "Basedow's disease", NA_character_, 1L,
  "03", "2020-06-07 23:59:59", "SOME TERM",
  2, "Some query", "Some term", NA_integer_,
  "05", "2020-06-09 23:59:59", "ALVEOLAR PROTEINOSIS",
  7, "Alveolar proteinosis", NA_character_, NA_integer_
)
derive_vars_query(adae, queries)
```

derive_vars_suppqual *Join Supplementary Qualifier Variables into the Parent SDTM Domain*

Description

[Deprecated]

Deprecated, please use `metatools::combine_supp()` instead.

The SDTM does not allow any new variables beside ones assigned to each SDTM domain. So, Supplemental Qualifier is introduced to supplement each SDTM domain to contain non standard variables. `dataset_suppqual` can be either a single SUPPQUAL dataset or separate supplementary data sets (SUPP) such as SUPPDM, SUPPAE, and SUPPEX. When a `dataset_suppqual` is a single SUPPQUAL dataset, specify two characterdomain value.

`derive_vars_suppqual()` expects USUBJID, RDOMAIN, IDVAR, IDVARVAL, QNAM, QLABEL, and QVAL variables to exist in `dataset_suppqual`.

Usage

```
derive_vars_suppqual(dataset, dataset_suppqual, domain = NULL)
```

Arguments

dataset	A SDTM domain data set.
dataset_suppqual	A Supplemental Qualifier (SUPPQUAL) data set.
domain	Two letter domain value. Used when supplemental data set is common across multiple SDTM domain.

Value

A data frame with SUPPQUAL variables appended to parent data set.

Author(s)

Vignesh Thanikachalam

derive_vars_transposed

Derive Variables by Transposing and Merging a Second Dataset

Description

Adds variables from a vertical dataset after transposing it into a wide one.

Usage

```
derive_vars_transposed(
  dataset,
  dataset_merge,
  by_vars,
  key_var,
  value_var,
  filter = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars parameter are required
dataset_merge	Dataset to transpose and merge The variables specified by the by_vars, key_var and value_var parameters are expected
by_vars	Keys used to merge dataset_merge with dataset

key_var	The variable of dataset_merge containing the names of the transposed variables
value_var	The variable of dataset_merge containing the values of the transposed variables
filter	Expression used to restrict the records of dataset_merge prior to transposing

Details

After filtering dataset_merge based upon the condition provided in filter, this dataset is transposed and subsequently merged onto dataset using by_vars as keys.

Value

The input dataset with transposed variables from dataset_merge added

Author(s)

Thomas Neitmann

Examples

```
library(dplyr, warn.conflicts = FALSE)

cm <- tibble::tribble(
  ~USUBJID, ~CMGRPID, ~CMREFID, ~CMDECOD,
  "BP40257-1001", "14", "1192056", "PARACETAMOL",
  "BP40257-1001", "18", "2007001", "SOLUMEDROL",
  "BP40257-1002", "19", "2791596", "SPIRONOLACTONE"
)

facm <- tibble::tribble(
  ~USUBJID, ~FAGRPID, ~FAREFID, ~FATESTCD, ~FASTRESC,
  "BP40257-1001", "1", "1192056", "CMATC1CD", "N",
  "BP40257-1001", "1", "1192056", "CMATC2CD", "N02",
  "BP40257-1001", "1", "1192056", "CMATC3CD", "N02B",
  "BP40257-1001", "1", "1192056", "CMATC4CD", "N02BE",
  "BP40257-1001", "1", "2007001", "CMATC1CD", "D",
  "BP40257-1001", "1", "2007001", "CMATC2CD", "D10",
  "BP40257-1001", "1", "2007001", "CMATC3CD", "D10A",
  "BP40257-1001", "1", "2007001", "CMATC4CD", "D10AA",
  "BP40257-1001", "2", "2007001", "CMATC1CD", "D",
  "BP40257-1001", "2", "2007001", "CMATC2CD", "D07",
  "BP40257-1001", "2", "2007001", "CMATC3CD", "D07A",
  "BP40257-1001", "2", "2007001", "CMATC4CD", "D07AA",
  "BP40257-1001", "3", "2007001", "CMATC1CD", "H",
  "BP40257-1001", "3", "2007001", "CMATC2CD", "H02",
  "BP40257-1001", "3", "2007001", "CMATC3CD", "H02A",
  "BP40257-1001", "3", "2007001", "CMATC4CD", "H02AB",
  "BP40257-1002", "1", "2791596", "CMATC1CD", "C",
  "BP40257-1002", "1", "2791596", "CMATC2CD", "C03",
  "BP40257-1002", "1", "2791596", "CMATC3CD", "C03D",
  "BP40257-1002", "1", "2791596", "CMATC4CD", "C03DA"
)
```



```

)
cm %>%
  derive_vars_transposed(
    facm,
    by_vars = vars(USUBJID, CMREFID = FAREFID),
    key_var = FATESTCD,
    value_var = FASTRESC
  ) %>%
  select(USUBJID, CMDECOD, starts_with("CMATC"))

```

 derive_var_ady

Derive Analysis Study Day

Description

[Deprecated]

This function is *deprecated*, please use `derive_vars_dy()` instead.

Adds the analysis study day (ADY) to the dataset, i.e., study day of analysis date.

Usage

```
derive_var_ady(dataset, reference_date = TRTSDT, date = ADT)
```

Arguments

dataset	Input dataset The columns specified by the <code>reference_date</code> and the <code>date</code> parameter are expected.
reference_date	The start date column, e.g., date of first treatment A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object. The default is TRTSDT.
date	The end date column for which the study day should be derived A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object. The default is ADT

Details

The study day is derived as number of days from the start date to the end date. If it is non-negative, one is added. I.e., the study day of the start date is 1.

Value

The input dataset with ADY column added

Author(s)

Stefan Bundfuss

derive_var_aendy	<i>Derive Analysis End Relative Day</i>
------------------	---

Description**[Deprecated]**

This function is *deprecated*, please use `derive_vars_dy()` instead.

Adds the analysis end relative day (AENDY) to the dataset, i.e. study day of analysis end date

Usage

```
derive_var_aendy(dataset, reference_date = TRTSDT, date = AENDT)
```

Arguments

dataset	Input dataset The columns specified by the <code>reference_date</code> and the <code>date</code> parameter are expected.
reference_date	The start date column, e.g., date of first treatment A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object. The default is TRTSDT.
date	The end date column for which the study day should be derived A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object. The default is AENDT

Details

The study day is derived as number of days from the start date to the end date. If it is nonnegative, one is added. I.e., the study day of the start date is 1.

Value

The input dataset with AENDY column added

Author(s)

Stefan Bundfuss

 derive_var_agegr_fda *Derive Age Groups*

Description

Functions for deriving standardized age groups.

Usage

```
derive_var_agegr_fda(dataset, age_var, age_unit = NULL, new_var)
```

```
derive_var_agegr_ema(dataset, age_var, age_unit = NULL, new_var)
```

Arguments

dataset	Input dataset.
age_var	AGE variable.
age_unit	AGE unit variable. The AGE unit variable is used to convert AGE to 'years' so that grouping can occur. This is only used when the age_var variable does not have a corresponding unit in the dataset. Default: NULL Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'
new_var	New variable to be created.

Details

derive_var_agegr_fda() derives age groups according to FDA guidance. age_var will be split in categories: <18, 18-64, >=65.

derive_var_agegr_ema() derives age groups according to EMA guidance. age_var will be split into categories: 0-27 days (Newborns), 28 days to 23 months (Infants and Toddlers), 2-11 (Children), 12-17 (Adolescents), 18-64, 65-84, >=85.

Value

dataset with new column new_var of class factor.

Author(s)

Ondrej Slama

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_dm)

admiral_dm %>%
  derive_var_agegr_fda(age_var = AGE, new_var = AGEGR1) %>%
  select(SUBJID, AGE, AGEGR1)

data <- tibble::tribble(
  ~BRTHDT, ~RANDDT,
  lubridate::ymd("1984-09-06"), lubridate::ymd("2020-02-24")
)

data %>%
  derive_vars_aage(unit = "months") %>%
  derive_var_agegr_fda(AAGE, age_unit = NULL, AGEGR1)

data.frame(AGE = 1:100) %>%
  derive_var_agegr_fda(age_var = AGE, age_unit = "years", new_var = AGEGR1)
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_dm)

admiral_dm %>%
  derive_var_agegr_ema(age_var = AGE, new_var = AGEGR1) %>%
  select(SUBJID, AGE, AGEGR1)

data.frame(AGE = 1:100) %>%
  derive_var_agegr_ema(age_var = AGE, age_unit = "years", new_var = AGEGR1)

data.frame(AGE = 1:20) %>%
  derive_var_agegr_ema(age_var = AGE, age_unit = "years", new_var = AGEGR1)

```

derive_var_age_years *Derive Age in Years*

Description

Derive Age in Years

Usage

```
derive_var_age_years(dataset, age_var, age_unit = NULL, new_var)
```

Arguments

dataset	Input dataset.
age_var	AGE variable.

age_unit	AGE unit variable. The AGE unit variable is used to convert AGE to 'years' so that grouping can occur. This is only used when the age_var variable does not have a corresponding unit in the dataset. Default: NULL Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'
new_var	New AGE variable to be created in years.

Details

This function is used to convert age variables into years. These can then be used to create age groups.

Value

The input dataset with new_var parameter added in years.

Author(s)

Michael Thorpe

Examples

```
library(dplyr, warn.conflicts = FALSE)

data <- data.frame(
  AGE = c(27, 24, 3, 4, 1),
  AGEU = c("days", "months", "years", "weeks", "years")
)

data %>%
  derive_var_age_years(., AGE, new_var = AAGE)

data.frame(AGE = c(12, 24, 36, 48)) %>%
  derive_var_age_years(., AGE, age_unit = "months", new_var = AAGE)
```

derive_var_analysis_ratio

Derive Ratio Variable

Description

Derives a ratio variable for a BDS dataset based on user specified variables.

Usage

```
derive_var_analysis_ratio(dataset, numer_var, denom_var, new_var = NULL)
```

Arguments

dataset	Input dataset
numer_var	Variable containing numeric values to be used in the numerator of the ratio calculation.
denom_var	Variable containing numeric values to be used in the denominator of the ratio calculation.
new_var	A user-defined variable that will be appended to the dataset. The default behavior will take the denominator variable and prefix it with R2 and append to the dataset. Using this argument will override this default behavior. Default is NULL.

Details

A user wishing to calculate a Ratio to Baseline, AVAL / BASE will have returned a new variable R2BASE that will be appended to the input dataset. Ratio to Analysis Range Lower Limit AVAL / ANRLO will return a new variable R2ANRLO, and Ratio to Analysis Range Upper Limit AVAL / ANRHI will return a new variable R2ANRLO. Please note how the denominator variable has the prefix R2----. A user can override the default returned variables by using the new_var argument. Also, values of 0 in the denominator will return NA in the derivation.

Reference CDISC ADaM Implementation Guide Version 1.1 Section 3.3.4 Analysis Parameter Variables for BDS Datasets

Value

The input dataset with a ratio variable appended

Author(s)

Ben Straub

Examples

```
library(dplyr, warn.conflicts = FALSE)

data <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~SEQ, ~AVAL, ~BASE, ~ANRLO, ~ANRHI,
  "P01", "ALT", 1, 27, 27, 6, 34,
  "P01", "ALT", 2, 41, 27, 6, 34,
  "P01", "ALT", 3, 17, 27, 6, 34,
  "P02", "ALB", 1, 38, 38, 33, 49,
  "P02", "ALB", 2, 39, 38, 33, 49,
  "P02", "ALB", 3, 37, 38, 33, 49
)

# Returns "R2" prefixed variables
data %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = BASE) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRLO) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRHI)
```

```
# Returns user-defined variables
data %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = BASE, new_var = R01BASE) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRLO, new_var = R01ANRLO) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRHI, new_var = R01ANRHI)
```

derive_var_anrind	<i>Derive Reference Range Indicator</i>
-------------------	---

Description

Derive Reference Range Indicator

Usage

```
derive_var_anrind(dataset)
```

Arguments

dataset	The input dataset
---------	-------------------

Details

ANRIND is set to

- "NORMAL" if AVAL is greater or equal ANRLO and less than or equal ANRHI; or if AVAL is greater than or equal ANRLO and ANRHI is missing; or if AVAL is less than or equal ANRHI and ANRLO is missing
- "LOW" if AVAL is less than ANRLO and either A1LO is missing or AVAL is greater than or equal A1LO
- "HIGH" if AVAL is greater than ANRHI and either A1HI is missing or AVAL is less than or equal A1HI
- "LOW LOW" if AVAL is less than A1LO
- "HIGH HIGH" if AVAL is greater than A1HI

Value

The input dataset with additional column ANRIND

Author(s)

Thomas Neitmann

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_vs)

ref_ranges <- tibble::tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI, ~A1LO, ~A1HI,
  "DIABP",   60,     80,   40,   90,
  "PULSE",   60,    100,   40,  110
)

admiral_vs %>%
  mutate(
    PARAMCD = VSTESTCD,
    AVAL = VSSTRESN
  ) %>%
  filter(PARAMCD %in% c("PULSE", "DIABP")) %>%
  derive_vars_merged(ref_ranges, by_vars = vars(PARAMCD)) %>%
  derive_var_anrind() %>%
  select(USUBJID, PARAMCD, AVAL, ANRLO:ANRIND)

```

 derive_var_astdy

Derive Analysis Start Relative Day

Description**[Deprecated]**

This function is *deprecated*, please use `derive_vars_dy()` instead.

Adds the analysis start relative day (ASTDY) to the dataset, i.e., study day of analysis start date.

Usage

```
derive_var_astdy(dataset, reference_date = TRTSDT, date = ASTDT)
```

Arguments

dataset	Input dataset The columns specified by the <code>reference_date</code> and the <code>date</code> parameter are expected.
reference_date	The start date column, e.g., date of first treatment A date or date-time object column is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object. The default is TRTSDT.

date The end date column for which the study day should be derived
 A date or date-time object column is expected.
 Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
 The default is ASTDT

Details

The study day is derived as number of days from the start date to the end date. If it is nonnegative, one is added. I.e., the study day of the start date is 1.

Value

The input dataset with ASTDY column added

Author(s)

Stefan Bundfuss

derive_var_atirel *Derive Time Relative to Reference*

Description

[Deprecated]

This function is *deprecated*, as it is deemed as too specific for admiral. Derivations like this can be implemented calling `mutate()` and `case_when()`.

Derives the variable ATIREL to CONCOMITANT, PRIOR, PRIOR_CONCOMITANT or NULL based on the relationship of cm Analysis start/end date/times to treatment start date/time

Usage

```
derive_var_atirel(dataset, flag_var, new_var)
```

Arguments

dataset Input dataset The variables TRTSDTM, ASTDTM, AENDTM are expected
 flag_var Name of the variable with Analysis Start Date Imputation Flag
 new_var Name of variable to create

Details

ATIREL is set to:

- null, if Datetime of First Exposure to Treatment is missing,
- "CONCOMITANT", if the Analysis Start Date/Time is greater than or equal to Datetime of First Exposure to Treatment,
- "PRIOR", if the Analysis End Date/Time is not missing and less than the Datetime of First Exposure to Treatment,
- "CONCOMITANT" if the date part of Analysis Start Date/Time is equal to the date part of Datetime of First Exposure to Treatment and the Analysis Start Time Imputation Flag is 'H' or 'M',
- otherwise it is set to "PRIOR_CONCOMITANT".

Value

A dataset containing all observations and variables of the input dataset and additionally the variable specified by the new_var parameter.

Author(s)

Teckla Akinyi

derive_var_base	<i>Derive Baseline Variables</i>
-----------------	----------------------------------

Description

Derive baseline variables, e.g. BASE or BNRIND, in a BDS dataset

Usage

```
derive_var_base(
  dataset,
  by_vars,
  source_var = AVAL,
  new_var = BASE,
  filter = ABLFL == "Y"
)
```

Arguments

dataset	The input dataset
by_vars	Grouping variables uniquely identifying a set of records for which to calculate new_var
source_var	The column from which to extract the baseline value, e.g. AVAL
new_var	The name of the newly created baseline column, e.g. BASE
filter	The condition used to filter dataset for baseline records. By default ABLFL == "Y"

Details

For each `by_vars` group the baseline record is identified by filtering using the condition specified by `filter` which defaults to `ABLFL == "Y"`. Subsequently, every value of the `new_var` variable for the `by_vars` group is set to the value of the `source_var` variable of the baseline record. In case there are multiple baseline records within `by_vars` an error is issued.

Value

A new data.frame containing all records and variables of the input dataset plus the `new_var` variable

Author(s)

Thomas Neitmann

Examples

```
dataset <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~PARAMCD, ~AVAL, ~AVALC, ~AVISIT, ~ABLFL,
  "TEST01", "PAT01", "PARAM01", 10.12, NA, "Baseline", "Y",
  "TEST01", "PAT01", "PARAM01", 9.700, NA, "Day 7", "N",
  "TEST01", "PAT01", "PARAM01", 15.01, NA, "Day 14", "N",
  "TEST01", "PAT01", "PARAM02", 8.350, NA, "Baseline", "Y",
  "TEST01", "PAT01", "PARAM02", NA, NA, "Day 7", "N",
  "TEST01", "PAT01", "PARAM02", 8.350, NA, "Day 14", "N",
  "TEST01", "PAT01", "PARAM03", NA, "LOW", "Baseline", "Y",
  "TEST01", "PAT01", "PARAM03", NA, "LOW", "Day 7", "N",
  "TEST01", "PAT01", "PARAM03", NA, "MEDIUM", "Day 14", "N",
  "TEST01", "PAT01", "PARAM04", NA, "HIGH", "Baseline", "Y",
  "TEST01", "PAT01", "PARAM04", NA, "HIGH", "Day 7", "N",
  "TEST01", "PAT01", "PARAM04", NA, "MEDIUM", "Day 14", "N"
)

## Derive `BASE` variable from `AVAL`
derive_var_base(
  dataset,
  by_vars = vars(USUBJID, PARAMCD),
  source_var = AVAL,
  new_var = BASE
)

## Derive `BASEC` variable from `AVALC`
derive_var_base(
  dataset,
  by_vars = vars(USUBJID, PARAMCD),
  source_var = AVALC,
  new_var = BASEC
)

## Derive `BNRIND` variable from `ANRIND`
if (FALSE) {
```

```

derive_var_base(
  dataset,
  by_vars = vars(USUBJID, PARAMCD),
  source_var = ANRIND,
  new_var = BNRIND
)
}

```

derive_var_basetype *Derive BASETYPE Variable*

Description

Baseline Type BASETYPE is needed when there is more than one definition of baseline for a given Analysis Parameter PARAM in the same dataset. For a given parameter, if Baseline Value BASE is populated, and there is more than one definition of baseline, then BASETYPE must be non-null on all records of any type for that parameter. Each value of BASETYPE refers to a definition of baseline that characterizes the value of BASE on that row. Please see section 4.2.1.6 of the ADaM Implementation Guide, version 1.3 for further background.

Usage

```
derive_var_basetype(dataset, basetypes)
```

Arguments

dataset	Input dataset The columns specified in the expressions inside basetypes are required.
basetypes	A <i>named</i> list of expressions created using the <code>rlang::exprs()</code> function The names corresponds to the values of the newly created BASETYPE variables and the expressions are used to subset the input dataset.

Details

Adds the BASETYPE variable to a dataset and duplicates records based upon the provided conditions. For each element of basetypes the input dataset is subset based upon the provided expression and the BASETYPE variable is set to the name of the expression. Then, all subsets are stacked. Records which do not match any condition are kept and BASETYPE is set to NA.

Value

The input dataset with variable BASETYPE added

Author(s)

Thomas Neitmann

Examples

```

bds <- tibble::tribble(
  ~USUBJID, ~EPOCH,      ~PARAMCD, ~ASEQ, ~AVAL,
  "P01",    "RUN-IN",    "PARAM01", 1, 10.0,
  "P01",    "RUN-IN",    "PARAM01", 2, 9.8,
  "P01",    "DOUBLE-BLIND", "PARAM01", 3, 9.2,
  "P01",    "DOUBLE-BLIND", "PARAM01", 4, 10.1,
  "P01",    "OPEN-LABEL", "PARAM01", 5, 10.4,
  "P01",    "OPEN-LABEL", "PARAM01", 6, 9.9,
  "P02",    "RUN-IN",    "PARAM01", 1, 12.1,
  "P02",    "DOUBLE-BLIND", "PARAM01", 2, 10.2,
  "P02",    "DOUBLE-BLIND", "PARAM01", 3, 10.8,
  "P02",    "OPEN-LABEL", "PARAM01", 4, 11.4,
  "P02",    "OPEN-LABEL", "PARAM01", 5, 10.8
)

bds_with_basetype <- derive_var_basetype(
  dataset = bds,
  basetypes = rlang::exprs(
    "RUN-IN" = EPOCH %in% c("RUN-IN", "STABILIZATION", "DOUBLE-BLIND", "OPEN-LABEL"),
    "DOUBLE-BLIND" = EPOCH %in% c("DOUBLE-BLIND", "OPEN-LABEL"),
    "OPEN-LABEL" = EPOCH == "OPEN-LABEL"
  )
)

# Below print statement will print all 23 records in the data frame
# bds_with_basetype
print(bds_with_basetype, n = Inf)

dplyr::count(bds_with_basetype, BASETYPE, name = "Number of Records")

# An example where all parameter records need to be included for 2 different
# baseline type derivations (such as LAST and WORST)
bds <- tibble::tribble(
  ~USUBJID, ~EPOCH,      ~PARAMCD, ~ASEQ, ~AVAL,
  "P01",    "RUN-IN",    "PARAM01", 1, 10.0,
  "P01",    "RUN-IN",    "PARAM01", 2, 9.8,
  "P01",    "DOUBLE-BLIND", "PARAM01", 3, 9.2,
  "P01",    "DOUBLE-BLIND", "PARAM01", 4, 10.1
)

bds_with_basetype <- derive_var_basetype(
  dataset = bds,
  basetypes = rlang::exprs(
    "LAST" = TRUE,
    "WORST" = TRUE
  )
)

print(bds_with_basetype, n = Inf)

```

```
dplyr::count(bds_with_basetype, BASETYPE, name = "Number of Records")
```

derive_var_chg	<i>Derive Change from Baseline</i>
----------------	------------------------------------

Description

Derive change from baseline (CHG) in a BDS dataset

Usage

```
derive_var_chg(dataset)
```

Arguments

dataset The input dataset. Required variables are AVAL and BASE.

Details

Change from baseline is calculated by subtracting the baseline value from the analysis value.

Value

The input dataset with an additional column named CHG

Author(s)

Thomas Neitmann

See Also

[derive_var_pchg\(\)](#)

Examples

```
advs <- tibble::tribble(  
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BASE,  
  "P01",    "WEIGHT", 80,    "Y",    80,  
  "P01",    "WEIGHT", 80.8,  "",    80,  
  "P01",    "WEIGHT", 81.4,  "",    80,  
  "P02",    "WEIGHT", 75.3,  "Y",   75.3,  
  "P02",    "WEIGHT", 76,    "",    75.3  
)  
derive_var_chg(advs)
```

derive_var_disposition_dt
Derive a Disposition Date

Description

[Deprecated]

This function is *deprecated*, please use `derive_vars_merged_dt()` instead.

Derive a disposition status date from the the relevant records in the disposition domain.

Usage

```
derive_var_disposition_dt(
  dataset,
  dataset_ds,
  new_var,
  dtc,
  filter_ds,
  date_imputation = NULL,
  preserve = FALSE,
  subject_keys = vars(STUDYID, USUBJID)
)
```

Arguments

dataset	Input dataset
dataset_ds	Datasets containing the disposition information (e.g.: ds) It must contain: <ul style="list-style-type: none"> • STUDYID, USUBJID, • The variable(s) specified in the dtc • The variables used in filter_ds.
new_var	Name of the disposition date variable a variable name is expected
dtc	The character date used to derive/impute the disposition date A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. If the year part is not recorded (missing date), no imputation is performed.
filter_ds	Filter condition for the disposition data. Filter used to select the relevant disposition data. It is expected that the filter restricts dataset_ds such that there is at most one observation per patient. An error is issued otherwise. Permitted Values: logical expression.

date_imputation	<p>The value to impute the day/month when a datepart is missing.</p> <p>If NULL: no date imputation is performed and partial dates are returned as missing.</p> <p>Otherwise, a character value is expected, either as a</p> <ul style="list-style-type: none"> • format with day and month specified as 'mm-dd': e.g. '06-15' for the 15th of June • or as a keyword: 'FIRST', 'MID', 'LAST' to impute to the first/mid/last day/month. <p>Default is NULL</p>
preserve	<p>Preserve day if month is missing and day is present</p> <p>For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").</p> <p>Permitted Values: TRUE, FALSE</p> <p>Default: FALSE</p>
subject_keys	<p>Variables to uniquely identify a subject</p> <p>A list of quosures where the expressions are symbols as returned by vars() is expected.</p>

Value

the input dataset with the disposition date (new_var) added

Author(s)

Samia Kabi

derive_var_disposition_status

Derive a Disposition Status at a Specific Timepoint

Description

Derive a disposition status from the the relevant records in the disposition domain.

Usage

```
derive_var_disposition_status(
  dataset,
  dataset_ds,
  new_var,
  status_var,
  format_new_var = format_eoxxstt_default,
  filter_ds,
  subject_keys = vars(STUDYID, USUBJID)
)
```


Arguments

dataset	Input dataset.
dataset_ds	Dataset containing the disposition information (e.g.: ds). It must contain: <ul style="list-style-type: none"> • STUDYID, USUBJID, • The variable(s) specified in the status_var • The variables used in filter_ds.
new_var	Name of the disposition status variable. A variable name is expected (e.g. EOSSTT).
status_var	The variable used to derive the disposition status. A variable name is expected (e.g. DSDECOD).
format_new_var	The format used to derive the status. Default: format_eoxxstt_default() defined as: <pre>format_eoxxstt_default <- function(status) { case_when(status == "COMPLETED" ~ "COMPLETED", status != "COMPLETED" & !is.na(status) ~ "DISCONTINUED", TRUE ~ "ONGOING") }</pre> <p>where status is the status_var .</p>
filter_ds	Filter condition for the disposition data. one observation per patient. An error is issued otherwise. Permitted Values: logical expression.
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by vars() is expected.

Value

The input dataset with the disposition status (new_var) added. new_var is derived based on the values given in status_var and according to the format defined by format_new_var (e.g. when the default format is used, the function will derive new_var as: "COMPLETED" if status_var == "COMPLETED", "DISCONTINUED" if status_var is not "COMPLETED" nor NA, "ONGOING" otherwise).

Author(s)

Samia Kabi

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ds")

# Default derivation: EOSSTT =
#- COMPLETED when status_var = COMPLETED
#- DISCONTINUED when status_var is not COMPLETED nor NA
#- ONGOING otherwise

admiral_dm %>%
  derive_var_disposition_status(
    dataset_ds = admiral_ds,
    new_var = EOSSTT,
    status_var = DSDECOD,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, EOSSTT)

# Specific derivation: EOSSTT =
#- COMPLETED when status_var = COMPLETED
#- DISCONTINUED DUE TO AE when status_var = ADVERSE EVENT
#- DISCONTINUED NOT DUE TO AE when status_var != ADVERSE EVENT nor COMPLETED nor missing
#- ONGOING otherwise

format_eoxxstt1 <- function(x) {
  case_when(
    x == "COMPLETED" ~ "COMPLETED",
    x == "ADVERSE EVENT" ~ "DISCONTINUED DUE TO AE",
    !(x %in% c("ADVERSE EVENT", "COMPLETED")) & !is.na(x) ~ "DISCONTINUED NOT DUE TO AE",
    TRUE ~ "ONGOING"
  )
}

admiral_dm %>%
  derive_var_disposition_status(
    dataset_ds = admiral_ds,
    new_var = EOSSTT,
    status_var = DSDECOD,
    format_new_var = format_eoxxstt1,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, EOSSTT)

```

derive_var_dthcaus *Derive Death Cause*

Description

Derive death cause (DTHCAUS) and add traceability variables if required.

Usage

```
derive_var_dthcaus(
  dataset,
  ...,
  source_datasets,
  subject_keys = vars(STUDYID, USUBJID)
)
```

Arguments

dataset	Input dataset. The variables specified by subject_keys are required.
...	Objects of class "dthcaus_source" created by dthcaus_source() .
source_datasets	A named list containing datasets in which to search for the death cause
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by vars() is expected.

Details

This function derives DTHCAUS along with the user-defined traceability variables, if required. If a subject has death info from multiple sources, the one from the source with the earliest death date will be used. If dates are equivalent, the first source will be kept, so the user should provide the inputs in the preferred order.

Value

The input dataset with DTHCAUS variable added.

Author(s)

Shimeng Huang, Samia Kabi, Thomas Neitmann

See Also

[dthcaus_source\(\)](#)

Examples

```
adsl <- tibble::tribble(
  ~STUDYID, ~USUBJID,
  "STUDY01", "PAT01",
  "STUDY01", "PAT02",
  "STUDY01", "PAT03"
)
ae <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~AESEQ, ~AEDECOD, ~AEOUT, ~AEDTHDTC,
  "STUDY01", "PAT01", 12, "SUDDEN DEATH", "FATAL", "2021-04-04"
```

```

)
ds <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~DSSEQ, ~DSDECOD, ~DSTERM, ~DSSTDTC,
  "STUDY01", "PAT02", 1, "INFORMED CONSENT OBTAINED", "INFORMED CONSENT OBTAINED", "2021-04-03",
  "STUDY01", "PAT02", 2, "RANDOMIZATION", "RANDOMIZATION", "2021-04-11",
  "STUDY01", "PAT02", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
  "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
)

# Derive `DTHCAUS` only - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDTC,
  mode = "first",
  dthcaus = AEDECOD
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDTC,
  mode = "first",
  dthcaus = DSTERM
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive `DTHCAUS` and add traceability variables - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDTC,
  mode = "first",
  dthcaus = AEDECOD,
  traceability_vars = vars(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDTC,
  mode = "first",
  dthcaus = DSTERM,
  traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive `DTHCAUS` as above - now including post-study deaths with different `DTHCAUS` value
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",

```

```

    date = AEDTHDTC,
    mode = "first",
    dthcaus = AEDECOD,
    traceability_vars = vars(DTHDOM = "AE", DTHSEQ = AESEQ)
  )

  src_ds <- dthcaus_source(
    dataset_name = "ds",
    filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
    date = DSSTDTC,
    mode = "first",
    dthcaus = DSTERM,
    traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
  )

  src_ds_post <- dthcaus_source(
    dataset_name = "ds",
    filter = DSDECOD == "DEATH" & DSTERM == "POST STUDY REPORTING OF DEATH",
    date = DSSTDTC,
    mode = "first",
    dthcaus = "POST STUDY: UNKNOWN CAUSE",
    traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
  )

  derive_var_dthcaus(adsl, src_ae, src_ds, src_ds_post, source_datasets = list(ae = ae, ds = ds))

```

derive_var_extreme_dt *Derive First or Last Date from Multiple Sources*

Description

Add the first or last date from multiple sources to the dataset, e.g., the last known alive date (LSTALVDT).

Usage

```

derive_var_extreme_dt(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = vars(STUDYID, USUBJID)
)

```

Arguments

dataset	Input dataset
	The variables specified by subject_keys are required.

new_var	Name of variable to create
...	Source(s) of dates. One or more <code>date_source()</code> objects are expected.
source_datasets	A named list containing datasets in which to search for the first or last date
mode	Selection mode (first or last) If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected. Permitted Values: "first", "last"
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by <code>vars()</code> is expected.

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the `filter` element are selected. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable specified by the `date` element. If the date variable is a date variable, the time is imputed as `time_imputation = "first"`. If the source variable is a character variable, it is converted to a datetime. If the date is incomplete, it is imputed as specified by the `date_imputation` element and with `time_imputation = "first"`.
3. The variables specified by the `traceability_vars` element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.
6. The time part is removed from the new variable.

Value

The input dataset with the new variable added.

Author(s)

Stefan Bundfuss, Thomas Neitmann

See Also

[date_source\(\)](#), [derive_var_extreme_dtm\(\)](#), [derive_vars_merged_dt\(\)](#), [derive_vars_merged_dtm\(\)](#), [derive_vars_merged\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ae")
```

```
data("admiral_lb")
data("admiral_adsl")

# derive last known alive date (LSTALVDT)
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first",
)
ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first",
)
lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10,
)
adsl_date <- date_source(dataset_name = "adsl", date = TRTEDT)

admiral_dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = admiral_ae, lb = admiral_lb
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT)

# derive last alive date and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)
```

```

)
)
lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10,
  traceability_vars = vars(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDT,
  traceability_vars = vars(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDT"
  )
)

admiral_dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = admiral_ae, lb = admiral_lb
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT, LALVDOM, LALVSEQ, LALVVAR)

```

```
derive_var_extreme_dtm
```

Derive First or Last Datetime from Multiple Sources

Description

Add the first or last datetime from multiple sources to the dataset, e.g., the last known alive datetime (LSTALVDTM).

Usage

```

derive_var_extreme_dtm(
  dataset,
  new_var,
  ...,

```



```

    source_datasets,
    mode,
    subject_keys = vars(STUDYID, USUBJID)
)

```

Arguments

dataset	Input dataset The variables specified by subject_keys are required.
new_var	Name of variable to create
...	Source(s) of dates. One or more date_source() objects are expected.
source_datasets	A named list containing datasets in which to search for the first or last date
mode	Selection mode (first or last) If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected. Permitted Values: "first", "last"
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by vars() is expected.

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the filter element are selected. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable specified by the date element. If the date variable is a date variable, the time is imputed as specified by the time_imputation element. If the source variable is a character variable, it is converted to a datetime. If the date is incomplete, it is imputed as specified by the date_imputation and time_imputation element.
3. The variables specified by the traceability_vars element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.

Value

The input dataset with the new variable added.

Author(s)

Stefan Bundfuss, Thomas Neitmann

See Also

[date_source\(\)](#), [derive_var_extreme_dt\(\)](#), [derive_vars_merged_dt\(\)](#), [derive_vars_merged_dtm\(\)](#), [derive_vars_merged\(\)](#)

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ae")
data("admiral_lb")
data("admiral_adsl")

# derive last known alive datetime (LSTALVDTM)
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first",
  time_imputation = "first"
)
ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first",
  time_imputation = "first"
)
lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10,
  time_imputation = "first"
)
adsl_date <- date_source(dataset_name = "adsl", date = TRTEDTM)

admiral_dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = admiral_ae, lb = admiral_lb
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM)

# derive last alive datetime and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first",
  time_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

```

```

)

ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first",
  time_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)

lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10,
  time_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDTM,
  traceability_vars = vars(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDTM"
  )
)

admiral_dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = admiral_ae, lb = admiral_lb
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM, LALVDOM, LALVSEQ, LALVVAR)

```

derive_var_extreme_flag

Add a Variable Flagging the First or Last Observation Within Each By Group

Description

Add a variable flagging the first or last observation within each by group

Usage

```
derive_var_extreme_flag(
  dataset,
  by_vars,
  order,
  new_var,
  mode,
  filter = deprecated(),
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars parameter are expected.
by_vars	Grouping variables Permitted Values: list of variables
order	Sort order The first or last observation is determined with respect to the specified order. Permitted Values: list of variables or functions of variables
new_var	Variable to add The specified variable is added to the output dataset. It is set to "Y" for the first or last observation (depending on the mode) of each by group. Permitted Values: list of name-value pairs
mode	Flag mode Determines of the first or last observation is flagged. Permitted Values: "first", "last"
filter	Deprecated, please use restrict_derivation() instead (see examples).
check_type	Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order. Default: "warning" Permitted Values: "none", "warning", "error"

Details

For each group (with respect to the variables specified for the by_vars parameter), new_var is set to "Y" for the first or last observation (with respect to the order specified for the order parameter and the flag mode specified for the mode parameter). Only observations included by the filter parameter are considered for flagging. Otherwise, new_var is set to NA. Thus, the direction of "worst" is considered fixed for all parameters in the dataset depending on the order and the mode, i.e. for every parameter the first or last record will be flagged across the whole dataset.

Value

The input dataset with the new flag variable added

Author(s)

Stefan Bundfuss

See Also

[derive_var_worst_flag\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_vs")

# Flag last value for each patient, test, and visit, baseline observations are ignored
admiral_vs %>%
  restrict_derivation(
    derivation = derive_var_extreme_flag,
    args = params(
      by_vars = vars(USUBJID, VSTESTCD, VISIT),
      order = vars(VSTPTNUM),
      new_var = LASTFL,
      mode = "last"
    ),
    filter = VISIT != "BASELINE"
  ) %>%
  arrange(USUBJID, VSTESTCD, VISITNUM, VSTPTNUM) %>%
  select(USUBJID, VSTESTCD, VISIT, VSTPTNUM, VSSTRESN, LASTFL)

# Baseline (ABLFL) examples:

input <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~PARAMCD, ~AVISIT, ~ADT, ~AVAL, ~DTYPE,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-27"), 15.0, NA,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-25"), 14.0, NA,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-23"), 15.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM01", "WEEK 1", as.Date("2021-04-27"), 10.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM01", "WEEK 2", as.Date("2021-04-30"), 12.0, NA,
  "TEST01", "PAT02", "PARAM01", "SCREENING", as.Date("2021-04-27"), 15.0, "AVERAGE",
  "TEST01", "PAT02", "PARAM01", "BASELINE", as.Date("2021-04-25"), 14.0, "AVERAGE",
  "TEST01", "PAT02", "PARAM01", "BASELINE", as.Date("2021-04-23"), 15.0, "AVERAGE",
  "TEST01", "PAT02", "PARAM01", "WEEK 1", as.Date("2021-04-27"), 10.0, "AVERAGE",
  "TEST01", "PAT02", "PARAM01", "WEEK 2", as.Date("2021-04-30"), 12.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-27"), 15.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-25"), 14.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-23"), 15.0, NA,
  "TEST01", "PAT01", "PARAM02", "BASELINE", as.Date("2021-04-27"), 10.0, "AVERAGE",
  "TEST01", "PAT01", "PARAM02", "WEEK 2", as.Date("2021-04-30"), 12.0, NA,
  "TEST01", "PAT02", "PARAM02", "SCREENING", as.Date("2021-04-27"), 15.0, NA,
```

```

"TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-25"), 14.0, NA,
"TEST01", "PAT02", "PARAM02", "WEEK 1", as.Date("2021-04-23"), 15.0, NA,
"TEST01", "PAT02", "PARAM02", "WEEK 1", as.Date("2021-04-27"), 10.0, NA,
"TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-30"), 12.0, NA
)

# Last observation
restrict_derivation(
  input,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD),
    order = vars(ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = High
restrict_derivation(
  input,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD),
    order = vars(AVAL, ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = Lo
restrict_derivation(
  input,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD),
    order = vars(desc(AVAL), ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
)

# Average observation
restrict_derivation(
  input,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD),
    order = vars(ADT, desc(AVAL)),
    new_var = ABLFL,

```

```

      mode = "last"
    ),
    filter = AVISIT == "BASELINE" & DTYPE == "AVERAGE"
  )

# OCCURDS Examples
data("admiral_ae")

# Most severe AE first occurrence per patient
admiral_ae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = vars(USUBJID),
    order = vars(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first"
  ) %>%
  arrange(USUBJID, AESTDY, AESEQ) %>%
  select(USUBJID, AEDECOD, AESEV, AESTDY, AESEQ, AOCCIFL)

# Most severe AE first occurrence per patient per body system
admiral_ae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCSIFL,
    by_vars = vars(USUBJID, AEBODSYS),
    order = vars(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first"
  ) %>%
  arrange(USUBJID, AESTDY, AESEQ) %>%
  select(USUBJID, AEBODSYS, AESEV, AESTDY, AOCCSIFL)

```

```
derive_var_last_dose_amt
```

Derive Last Dose Amount

Description

Add a variable for dose amount from the last dose to the input dataset.

Usage

```
derive_var_last_dose_amt(
  dataset,
```

```

dataset_ex,
filter_ex = NULL,
by_vars = vars(STUDYID, USUBJID),
dose_id = vars(),
dose_date,
analysis_date,
single_dose_condition = (EXDOSFRQ == "ONCE"),
new_var,
dose_var = EXDOSE,
traceability_vars = NULL
)

```

Arguments

dataset	Input dataset. The variables specified by the <code>by_vars</code> and <code>analysis_date</code> parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the <code>by_vars</code> , <code>dose_date</code> , <code>new_vars</code> parameters, and source variables from <code>traceability_vars</code> parameter are expected.
filter_ex	Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to <code>NULL</code> .
by_vars	Variables to join by (created by <code>dplyr::vars</code>).
dose_id	Variables to identify unique dose (created by <code>dplyr::vars</code>). Defaults to empty <code>vars()</code> .
dose_date	The EX dose date variable. A date or date-time object is expected.
analysis_date	The analysis date variable. A date or date-time object is expected.
single_dose_condition	The condition for checking if <code>dataset_ex</code> is single dose. An error is issued if the condition is not true. Defaults to <code>(EXDOSFRQ == "ONCE")</code> .
new_var	The new variable added to dataset.
dose_var	The EX source dose amount variable. Defaults to <code>EXDOSE</code> .
traceability_vars	A named list returned by <code>vars()</code> listing the traceability variables, e.g. <code>vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ)</code> . The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Details

The last dose amount is derived as the dose amount where the maximum `dose_date` is lower to or equal to the `analysis_date` per `by_vars` for each observation in dataset.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function `create_single_dose_dataset()` can be used to generate single doses from aggregate dose information and satisfy `single_dose_condition`.

Value

Input dataset with additional column new_var.

Author(s)

Annie Yang

See Also

[derive_vars_last_dose\(\)](#), [create_single_dose_dataset\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(ex_single)

admiral_ae %>%
  head(100) %>%
  derive_var_last_dose_amt(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    dose_date = EXENDTC,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC),
    new_var = LDOSE,
    dose_var = EXDOSE
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSE)

# or with traceability variables
admiral_ae %>%
  head(100) %>%
  derive_var_last_dose_amt(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    dose_date = EXENDTC,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC),
    new_var = LDOSE,
    dose_var = EXDOSE,
    traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXDOSE")
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR, LDOSE)
```

 derive_var_last_dose_date

Derive Last Dose Date-Time

Description

Add a variable for the dose date or datetime of the last dose to the input dataset.

Usage

```

derive_var_last_dose_date(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_var,
  output_datetime = TRUE,
  traceability_vars = NULL
)

```

Arguments

dataset	Input dataset. The variables specified by the <code>by_vars</code> and <code>analysis_date</code> parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the <code>by_vars</code> , <code>dose_date</code> , <code>new_vars</code> parameters, and source variables from <code>traceability_vars</code> parameter are expected.
filter_ex	Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL.
by_vars	Variables to join by (created by <code>dplyr::vars</code>).
dose_id	Variables to identify unique dose (created by <code>dplyr::vars</code>). Defaults to empty <code>vars()</code> .
dose_date	The EX dose date variable. A date or date-time object is expected.
analysis_date	The analysis date variable. A date or date-time object is expected.
single_dose_condition	The condition for checking if <code>dataset_ex</code> is single dose. An error is issued if the condition is not true. Defaults to <code>(EXDOSFRQ == "ONCE")</code> .
new_var	The new date or datetime variable added to dataset.
output_datetime	Display <code>new_var</code> as datetime or as date only. Defaults to TRUE.

traceability_vars

A named list returned by `vars()` listing the traceability variables, e.g. `vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ)`. The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Details

The last dose date is derived as the maximum dose date where the `dose_date` is lower to or equal to the `analysis_date` per `by_vars` for each observation in dataset. When `output_datetime` is `TRUE` and time is missing, then the last dose date time is imputed to `00:00:00`. However, if date is missing, then no imputation is done.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function `create_single_dose_dataset()` can be used to generate single doses from aggregate dose information and satisfy `single_dose_condition`.

Value

Input dataset with additional column `new_var`.

Author(s)

Ben Straub

See Also

[derive_vars_last_dose\(\)](#), [create_single_dose_dataset\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(ex_single)

admiral_ae %>%
  head(100) %>%
  derive_var_last_dose_date(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    dose_date = EXENDTC,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC),
    new_var = LDOSEDTM,
    traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXDOSE")
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR, LDOSEDTM)
```

 derive_var_last_dose_grp

Derive Last Dose with User-Defined Groupings

Description

Add a variable for user-defined dose grouping of the last dose to the input dataset.

Usage

```

derive_var_last_dose_grp(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_var,
  grp_brks,
  grp_lbls,
  include_lowest = TRUE,
  right = TRUE,
  dose_var = EXDOSE,
  traceability_vars = NULL
)

```

Arguments

dataset	Input dataset. The variables specified by the <code>by_vars</code> and <code>analysis_date</code> parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the <code>by_vars</code> , <code>dose_date</code> , <code>new_vars</code> parameters, and source variables from <code>traceability_vars</code> parameter are expected.
filter_ex	Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL.
by_vars	Variables to join by (created by <code>dplyr::vars</code>).
dose_id	Variables to identify unique dose (created by <code>dplyr::vars</code>). Defaults to empty <code>vars()</code> .
dose_date	The EX dose date variable. A date or date-time object is expected.
analysis_date	The analysis date variable. A date or date-time object is expected.
single_dose_condition	The condition for checking if <code>dataset_ex</code> is single dose. An error is issued if the condition is not true. Defaults to <code>(EXDOSFRQ == "ONCE")</code> .

<code>new_var</code>	The output variable defined by the user.
<code>grp_brks</code>	User supplied breaks to apply to groups. Refer to <code>breaks</code> parameter in <code>cut()</code> for details.
<code>grp_lbls</code>	User supplied labels to apply to groups. Refer to <code>labels</code> parameter in <code>cut()</code> for details.
<code>include_lowest</code>	logical, indicating if a value equal to the lowest (or highest, for <code>right = FALSE</code>) 'breaks' value should be included. Refer to <code>include.lowest</code> parameter in <code>cut()</code> for details.
<code>right</code>	Logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. Refer to <code>right</code> parameter in <code>cut()</code> for details.
<code>dose_var</code>	The source dose amount variable. Defaults to <code>EXDOSE</code> .
<code>traceability_vars</code>	A named list returned by <code>vars()</code> listing the traceability variables, e.g. <code>vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ)</code> . The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Details

Last dose is the dose with maximum `dose_date` that is lower to or equal to the `analysis_date` per `by_vars` for each observation in dataset. The last dose group is then derived by user-defined grouping, which groups `dose_var` as specified in `grp_brks`, and returns `grp_lbls` as the values for `new_var`.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function `create_single_dose_dataset()` can be used to generate single doses from aggregate dose information and satisfy `single_dose_condition`.

Value

Input dataset with additional column `new_var`.

Author(s)

Ben Straub

See Also

[derive_vars_last_dose\(\)](#), [cut\(\)](#), [create_single_dose_dataset\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(ex_single)
```

```

admiral_ae %>%
  head(100) %>%
  derive_var_last_dose_grp(
    head(ex_single, 100),
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      nchar(EXENDTC) >= 10,
    by_vars = vars(STUDYID, USUBJID),
    dose_date = EXSTDTC,
    new_var = LDGRP,
    grp_brks = c(0, 20, 40, 60),
    grp_lbls = c("Low", "Medium", "High"),
    include_lowest = TRUE,
    right = TRUE,
    dose_var = EXDOSE,
    analysis_date = AESTDTC,
    single_dose_condition = (EXSTDTC == EXENDTC),
    traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXENDTC")
  ) %>%
  select(USUBJID, LDGRP, LDOSEDOM, LDOSESEQ, LDOSEVAR)

```

derive_var_lstalvdt *Derive Last Known Alive Date*

Description

[Deprecated]

Deprecated, please use `derive_var_extreme_dt()` instead. Add the last known alive date (LSTALVDT) to the dataset.

Usage

```

derive_var_lstalvdt(
  dataset,
  ...,
  source_datasets,
  subject_keys = vars(STUDYID, USUBJID)
)

```

Arguments

dataset	Input dataset The variables specified by <code>subject_keys</code> are required.
...	Source(s) of known alive dates. One or more <code>lstalvdt_source()</code> objects are expected.
source_datasets	A named list containing datasets in which to search for the last known alive date

subject_keys Variables to uniquely identify a subject
 A list of quosures where the expressions are symbols as returned by vars() is expected.

Value

The input dataset with the LSTALVDT variable added.

Author(s)

Stefan Bundfuss, Thomas Neitmann

derive_var_merged_cat *Merge a Categorization Variable*

Description

Merge a categorization variable from a dataset to the input dataset. The observations to merge can be selected by a condition and/or selecting the first or last observation for each by group.

Usage

```
derive_var_merged_cat(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  source_var,
  cat_fun,
  filter_add = NULL,
  mode = NULL,
  missing_value = NA_character_
)
```

Arguments

dataset Input dataset
 The variables specified by the by_vars parameter are expected.

dataset_add Additional dataset
 The variables specified by the by_vars, the source_var, and the order parameter are expected.

by_vars Grouping variables
 The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations.
Permitted Values: list of variables created by vars()

order	<p>Sort order</p> <p>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) or NULL</p>
new_var	<p>New variable</p> <p>The specified variable is added to the additional dataset and set to the categorized values, i.e., cat_fun(<source variable>).</p>
source_var	Source variable
cat_fun	<p>Categorization function</p> <p>A function must be specified for this parameter which expects the values of the source variable as input and returns the categorized values.</p>
filter_add	<p>Filter for additional dataset (dataset_add)</p> <p>Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> a condition</p>
mode	<p>Selection mode</p> <p>Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null.</p> <p>If the order parameter is not specified, the mode parameter is ignored.</p> <p><i>Default:</i> NULL</p> <p><i>Permitted Values:</i> "first", "last", NULL</p>
missing_value	<p>Values used for missing information</p> <p>The new variable is set to the specified value for all by groups without observations in the additional dataset.</p> <p><i>Default:</i> NA_character_</p>

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The categorization variable is added to the additional dataset.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The categorization variable is merged to the input dataset.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for new_var derived from the additional dataset (dataset_add).

Author(s)

Stefan Bundfuss

Examples

```

library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_vs")

wgt_cat <- function(wgt) {
  case_when(
    wgt < 50 ~ "low",
    wgt > 90 ~ "high",
    TRUE ~ "normal"
  )
}

derive_var_merged_cat(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  order = vars(VSDTC, VSSEQ),
  filter_add = VSTESTCD == "WEIGHT" & substr(VISIT, 1, 9) == "SCREENING",
  new_var = WGTBLCAT,
  source_var = VSSTRESN,
  cat_fun = wgt_cat,
  mode = "last"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WGTBLCAT)

# defining a value for missing VS data
derive_var_merged_cat(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  order = vars(VSDTC, VSSEQ),
  filter_add = VSTESTCD == "WEIGHT" & substr(VISIT, 1, 9) == "SCREENING",
  new_var = WGTBLCAT,
  source_var = VSSTRESN,
  cat_fun = wgt_cat,
  mode = "last",
  missing_value = "MISSING"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WGTBLCAT)

```

 derive_var_merged_character

Merge a Character Variable

Description

Merge a character variable from a dataset to the input dataset. The observations to merge can be selected by a condition and/or selecting the first or last observation for each by group.

Usage

```
derive_var_merged_character(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  source_var,
  case = NULL,
  filter_add = NULL,
  mode = NULL,
  missing_value = NA_character_
)
```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter are expected.
dataset_add	Additional dataset The variables specified by the <code>by_vars</code> , the <code>source_var</code> , and the <code>order</code> parameter are expected.
by_vars	Grouping variables The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. <i>Permitted Values:</i> list of variables created by <code>vars()</code>
order	Sort order If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. <i>Default:</i> NULL <i>Permitted Values:</i> list of variables or <code>desc(<variable>)</code> function calls created by <code>vars()</code> , e.g., <code>vars(ADT, desc(AVAL))</code> or NULL
new_var	New variable The specified variable is added to the additional dataset and set to the transformed value with respect to the <code>case</code> parameter.
source_var	Source variable
case	Change case Changes the case of the values of the new variable. <i>Default:</i> NULL <i>Permitted Values:</i> NULL, "lower", "upper", "title"

filter_add	Filter for additional dataset (dataset_add) Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered. <i>Default:</i> NULL <i>Permitted Values:</i> a condition
mode	Selection mode Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null. If the order parameter is not specified, the mode parameter is ignored. <i>Default:</i> NULL <i>Permitted Values:</i> "first", "last", NULL
missing_value	Values used for missing information The new variable is set to the specified value for all by groups without observations in the additional dataset. <i>Default:</i> NA_character_ <i>Permitted Value:</i> A character scalar

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The (transformed) character variable is added to the additional dataset.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The character variable is merged to the input dataset.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for new_var derived from the additional dataset (dataset_add).

Author(s)

Stefan Bundfuss

Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_ds")

derive_var_merged_character(
  admiral_dm,
  dataset_add = admiral_ds,
  by_vars = vars(STUDYID, USUBJID),
  new_var = DISPSTAT,
  filter_add = DSCAT == "DISPOSITION EVENT",
```

```

source_var = DSDECOD,
case = "title"
) %>%
select(STUDYID, USUBJID, AGE, AGEU, DISPSTAT)

```

derive_var_merged_exist_flag

Merge an Existence Flag

Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in another dataset fulfilling a certain condition.

Usage

```

derive_var_merged_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  new_var,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL
)

```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter are expected.
dataset_add	Additional dataset The variables specified by the <code>by_vars</code> parameter are expected.
by_vars	Grouping variables <i>Permitted Values:</i> list of variables
new_var	New variable The specified variable is added to the input dataset.
condition	Condition The condition is evaluated at the additional dataset (<code>dataset_add</code>). For all by groups where it evaluates as TRUE at least once the new variable is set to the true value (<code>true_value</code>). For all by groups where it evaluates as FALSE or NA for all observations the new variable is set to the false value (<code>false_value</code>). The new variable is set to the missing value (<code>missing_value</code>) for by groups not present in the additional dataset.

true_value	True value <i>Default:</i> "Y"
false_value	False value <i>Default:</i> NA_character_
missing_value	Values used for missing information The new variable is set to the specified value for all by groups without observations in the additional dataset. <i>Default:</i> NA_character_ <i>Permitted Value:</i> A character scalar
filter_add	Filter for additional data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. <i>Permitted Values:</i> a condition

Details

1. The additional dataset is restricted to the observations matching the `filter_add` condition.
2. The new variable is added to the input dataset and set to the true value (`true_value`) if for the by group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE. It is set to the false value (`false_value`) if for the by group at least one observation exists and for all observations the condition evaluates to FALSE or NA. Otherwise, it is set to the missing value (`missing_value`).

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for `new_var` derived from the additional dataset (`dataset_add`).

Author(s)

Stefan Bundfuss

Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_ae")
derive_var_merged_exist_flag(
  admiral_dm,
  dataset_add = admiral_ae,
  by_vars = vars(STUDYID, USUBJID),
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, AERELFL)
```

```

data("admiral_vs")
derive_var_merged_exist_flag(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  false_value = "N",
  missing_value = "M"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WTBLHIFL)

```

derive_var_obs_number *Adds a Variable Numbering the Observations Within Each By Group*

Description

Adds a variable numbering the observations within each by group

Usage

```

derive_var_obs_number(
  dataset,
  by_vars = NULL,
  order = NULL,
  new_var = ASEQ,
  check_type = "none"
)

```

Arguments

dataset	Input dataset The variables specified by the order and the by_vars parameter are expected.
by_vars	Grouping variables Permitted Values: list of variables
order	Sort order Within each by group the observations are ordered by the specified order. Permitted Values: list of variables or functions of variables
new_var	Name of variable to create The new variable is set to the observation number for each by group. The numbering starts with 1. Default: ASEQ

check_type Check uniqueness?
 If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.
 Default: "none"
 Permitted Values: "none", "warning", "error"

Details

For each group (with respect to the variables specified for the by_vars parameter) the first or last observation (with respect to the order specified for the order parameter and the mode specified for the mode parameter) is included in the output dataset.

Value

A dataset containing all observations and variables of the input dataset and additionally the variable specified by the new_var parameter.

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_vs")

admiral_vs %>%
  select(USUBJID, VSTESTCD, VISITNUM, VSTPTNUM) %>%
  filter(VSTESTCD %in% c("HEIGHT", "WEIGHT")) %>%
  derive_var_obs_number(
    by_vars = vars(USUBJID, VSTESTCD),
    order = vars(VISITNUM, VSTPTNUM)
  )
```

derive_var_ontrtfl *Derive On-Treatment Flag Variable*

Description

Derive on-treatment flag (ONTRTFL) in an ADaM dataset with a single assessment date (e.g ADT) or event start and end dates (e.g. ASTDT/AENDT).

Usage

```

derive_var_ontrtfl(
  dataset,
  new_var = ONTRTFL,
  start_date,
  end_date = NULL,
  ref_start_date,
  ref_end_date = NULL,
  ref_end_window = 0,
  filter_pre_timepoint = NULL,
  span_period = NULL
)

```

Arguments

dataset	Input dataset. Required columns are start_date, end_date, ref_start_date and ref_end_date.
new_var	On-treatment flag variable name to be created. Default is ONTRTFL.
start_date	The start date (e.g. AESDT) or assessment date (e.g. ADT) Required; A date or date-time object column is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
end_date	The end date of assessment/event (e.g. AENDT) A date or date-time object column is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Optional; Default is null. If the used and date value is missing on an observation, it is assumed the medication is ongoing and ONTRTFL is set to "Y".
ref_start_date	The lower bound of the on-treatment period Required; A date or date-time object column is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
ref_end_date	The upper bound of the on-treatment period A date or date-time object column is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Optional; This can be null and everything after ref_start_date will be considered on-treatment. Default is NULL.
ref_end_window	A window to add to the upper bound ref_end_date measured in days (e.g. 7 if 7 days should be added to the upper bound) Optional; default is 0.
filter_pre_timepoint	An expression to filter observations as not on-treatment when date = ref_start_date. For example, if observations where VSTPT = PRE should not be considered on-treatment when date = ref_start_date, filter_pre_timepoint should be

	used to denote when the on-treatment flag should be set to null. Optional; default is NULL.
span_period	A "Y" scalar character. If "Y", events that started prior to the ref_start_date and are ongoing or end after the ref_start_date are flagged as "Y". Optional; default is NULL.

Details

On-Treatment is calculated by determining whether the assessment date or start/stop dates fall between 2 dates. The following logic is used to assign on-treatment = "Y":

1. start_date is missing and ref_start_date is non-missing
2. No timepoint filter is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date
3. A timepoint is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date and the filter provided in filter_pre_timepoint is not true.
4. ref_end_date is not provided and ref_start_date < start_date
5. ref_end_date is provided and ref_start_date < start_date <= ref_end_date + ref_end_window.

If the end_date is provided and the end_date < ref_start_date then the ONTRTFL is set to NULL. This would be applicable to cases where the start_date is missing and ONTRTFL has been assigned as "Y" above.

If the span_period is specified as "Y", this allows the user to assign ONTRTFL as "Y" to cases where the record started prior to the ref_start_date and was ongoing or ended after the ref_start_date.

Any date imputations needed should be done prior to calling this function.

Value

The input dataset with an additional column named ONTRTFL with a value of "Y" or NA

Author(s)

Alice Ehmann, Teckla Akinyi

Examples

```
library(dplyr)
library(lubridate, warn.conflict = FALSE)

advs <- tibble::tribble(
  ~USUBJID, ~ADT, ~TRTSDT, ~TRTEDT,
  "P01", ymd("2020-02-24"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P02", ymd("2020-01-01"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P03", ymd("2019-12-31"), ymd("2020-01-01"), ymd("2020-03-01")
)
derive_var_ontrtfl(
  advs,
  start_date = ADT,
```

```

    ref_start_date = TRTSDT,
    ref_end_date = TRTEDT
  )

  advs <- tibble::tribble(
    ~USUBJID, ~ADT,           ~TRTSDT,           ~TRTEDT,
    "P01",    ymd("2020-07-01"), ymd("2020-01-01"), ymd("2020-03-01"),
    "P02",    ymd("2020-04-30"), ymd("2020-01-01"), ymd("2020-03-01"),
    "P03",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01")
  )
  derive_var_ontrtfl(
    advs,
    start_date = ADT,
    ref_start_date = TRTSDT,
    ref_end_date = TRTEDT,
    ref_end_window = 60
  )

  advs <- tibble::tribble(
    ~USUBJID, ~ADTM,           ~TRTSDTM,           ~TRTEDTM,
    "P01",    ymd("2020-01-02T12:00"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
    "P02",    ymd("2020-01-01"),        ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
    "P03",    ymd("2019-12-31"),        ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
  ) %>%
  mutate(TPT = c(NA, "PRE", NA))
  derive_var_ontrtfl(
    advs,
    start_date = ADTM,
    ref_start_date = TRTSDTM,
    ref_end_date = TRTEDTM,
    filter_pre_timepoint = TPT == "PRE"
  )

  advs <- tibble::tribble(
    ~USUBJID, ~ASTDT,           ~TRTSDT,           ~TRTEDT,           ~AENDT,
    "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
    "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
    "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
  )
  derive_var_ontrtfl(
    advs,
    start_date = ASTDT,
    end_date = AENDT,
    ref_start_date = TRTSDT,
    ref_end_date = TRTEDT,
    ref_end_window = 60,
    span_period = "Y"
  )

  advs <- tibble::tribble(
    ~USUBJID, ~ASTDT,           ~AP01SDT,           ~AP01EDT,           ~AENDT,
    "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
    "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
  )

```

```
    "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
  )
  derive_var_ontrtfl(
    advs,
    new_var = ONTR01FL,
    start_date = ASTDT,
    end_date = AENDT,
    ref_start_date = AP01SDT,
    ref_end_date = AP01EDT,
    span_period = "Y"
  )
```

derive_var_pchg	<i>Derive Percent Change from Baseline</i>
-----------------	--

Description

Derive percent change from baseline (PCHG) in a BDS dataset

Usage

```
derive_var_pchg(dataset)
```

Arguments

dataset The input dataset. Required variables are AVAL and BASE.

Details

Percent change from baseline is calculated by dividing change from baseline by the absolute value of the baseline value and multiplying the result by 100.

Value

The input dataset with an additional column named PCHG

Author(s)

Thomas Neitmann

See Also

[derive_var_chg\(\)](#)

Examples

```
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BASE,
  "P01",    "WEIGHT", 80,    "Y",    80,
  "P01",    "WEIGHT", 80.8,  "",    80,
  "P01",    "WEIGHT", 81.4,  "",    80,
  "P02",    "WEIGHT", 75.3,  "Y",   75.3,
  "P02",    "WEIGHT", 76,    "",   75.3
)
derive_var_pchg(advs)
```

derive_var_shift	<i>Derive Shift</i>
------------------	---------------------

Description

Derives a character shift variable containing concatenated shift in values based on user-defined pairing, e.g., shift from baseline to analysis value, shift from baseline grade to analysis grade, ...

Usage

```
derive_var_shift(
  dataset,
  new_var,
  from_var,
  to_var,
  na_val = "NULL",
  sep_val = " to "
)
```

Arguments

dataset	Input dataset The columns specified by from_var and the to_var parameters are expected.
new_var	Name of the character shift variable to create.
from_var	Variable containing value to shift from.
to_var	Variable containing value to shift to.
na_val	Character string to replace missing values in from_var or to_var. Default: "NULL"
sep_val	Character string to concatenate values of from_var and to_var. Default: " to "

Details

new_var is derived by concatenating the values of from_var to values of to_var (e.g. "NORMAL to HIGH"). When from_var or to_var has missing value, the missing value is replaced by na_val (e.g. "NORMAL to NULL").

Value

The input dataset with the character shift variable added

Author(s)

Annie Yang

Examples

```
library(dplyr, warn.conflicts = FALSE)

data <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BNRIND, ~ANRIND,
  "P01", "ALB", 33, "Y", "LOW", "LOW",
  "P01", "ALB", 38, NA, "LOW", "NORMAL",
  "P01", "ALB", NA, NA, "LOW", NA,
  "P02", "ALB", 37, "Y", "NORMAL", "NORMAL",
  "P02", "ALB", 49, NA, "NORMAL", "HIGH",
  "P02", "SODIUM", 147, "Y", "HIGH", "HIGH"
)

data %>%
  convert_blanks_to_na() %>%
  derive_var_shift(
    new_var = SHIFT1,
    from_var = BNRIND,
    to_var = ANRIND
  )

# or only populate post-baseline records
data %>%
  convert_blanks_to_na() %>%
  restrict_derivation(
    derivation = derive_var_shift,
    args = params(
      new_var = SHIFT1,
      from_var = BNRIND,
      to_var = ANRIND
    ),
    filter = is.na(ABLFL)
  )
```

derive_var_trtdurd	<i>Derive Total Treatment Duration (Days)</i>
--------------------	---

Description

Derives total treatment duration (days) (TRTDURD)

Usage

```
derive_var_trtdurd(dataset, start_date = TRTSDT, end_date = TRTEDT)
```

Arguments

dataset	Input dataset The columns specified by the start_date and the end_date parameter are expected.
start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Default: TRTSDT
end_date	The end date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Default: TRTEDT

Details

The total treatment duration is derived as the number of days from start to end date plus one.

Value

The input dataset with TRTDURD added

Author(s)

Stefan Bundfuss

See Also

[derive_vars_duration\(\)](#)

Examples

```
data <- tibble::tribble(  
  ~TRTSDT, ~TRTEDT,  
  lubridate::ymd("2020-01-01"), lubridate::ymd("2020-02-24")  
)  
  
derive_var_trtdurd(data)
```

derive_var_trtedtm *Derive Datetime of Last Exposure to Treatment*

Description

[Deprecated]

This function is *deprecated*, please use `derive_vars_merged_dtm()` instead.

Derives datetime of last exposure to treatment (TRTEDTM)

Usage

```
derive_var_trtedtm(
  dataset,
  dataset_ex,
  filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & str_detect(EXTRT, "PLACEBO"))) &
    nchar(EXENDTC) >= 10,
  subject_keys = vars(STUDYID, USUBJID)
)
```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter are expected.
dataset_ex	ex dataset The variables EXENDTC, EXSEQ, and those specified by the <code>filter_ex</code> parameter are expected.
filter_ex	Filter condition for the ex dataset Only observations of the ex dataset which fulfill the specified condition are considered for the treatment start date. Default: <code>EXDOSE > 0 (EXDOSE == 0 & str_detect(EXTRT, 'PLACEBO')) & nchar(EXENDTC) >= 10</code> Permitted Values: logical expression
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by <code>vars()</code> is expected.

Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first observation (with respect to the order specified for the `order` parameter) is included in the output dataset.

Value

The input dataset with TRTEDTM variable added

Author(s)

Stefan Bundfuss

 derive_var_trtsdtm *Derive Datetime of First Exposure to Treatment*

Description**[Deprecated]**

This function is *deprecated*, please use `derive_vars_merged_dtm()` instead.

Derives datetime of first exposure to treatment (TRTSDTM)

Usage

```
derive_var_trtsdtm(
  dataset,
  dataset_ex,
  filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & str_detect(EXTRT, "PLACEBO"))) &
    nchar(EXSTDTC) >= 10,
  subject_keys = vars(STUDYID, USUBJID)
)
```

Arguments

dataset	Input dataset The variables specified by the <code>by_vars</code> parameter are expected.
dataset_ex	ex dataset The variables EXSTDTC, EXSEQ, and those specified by the <code>filter_ex</code> parameter are expected.
filter_ex	Filter condition for the ex dataset Only observations of the ex dataset which fulfill the specified condition are considered for the treatment start date. Default: <code>EXDOSE > 0 (EXDOSE == 0 & str_detect(EXTRT, 'PLACEBO')) & nchar(EXSTDTC) >= 10</code> Permitted Values: logical expression
subject_keys	Variables to uniquely identify a subject A list of quosures where the expressions are symbols as returned by <code>vars()</code> is expected.

Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first observation (with respect to the order specified for the `order` parameter) is included in the output dataset.

Value

The input dataset with TRTSDTM variable added

Author(s)

Stefan Bundfuss

derive_var_worst_flag *Adds a Variable Flagging the Maximal / Minimal Value Within a Group of Observations*

Description

Adds a Variable Flagging the Maximal / Minimal Value Within a Group of Observations

Usage

```
derive_var_worst_flag(
  dataset,
  by_vars,
  order,
  new_var,
  param_var,
  analysis_var,
  worst_high,
  worst_low,
  filter = deprecated(),
  check_type = "warning"
)
```

Arguments

dataset	Input dataset. Variables specified by by_vars, order, param_var, and analysis_var are expected.
by_vars	Grouping variables Permitted Values: list of variables
order	Sort order. Used to determine maximal / minimal observation if they are not unique, see Details section for more information.
new_var	Variable to add to the dataset. It is set "Y" for the maximal / minimal observation of each group, see Details section for more information.
param_var	Variable with the parameter values for which the maximal / minimal value is calculated.
analysis_var	Variable with the measurement values for which the maximal / minimal value is calculated.

worst_high	Character with param_var values specifying the parameters referring to "high". Use character(0) if not required.
worst_low	Character with param_var values specifying the parameters referring to "low". Use character(0) if not required.
filter	Deprecated, please use restrict_derivation() instead (see examples).
check_type	Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order. Default: "warning" Permitted Values: "none", "warning", "error"

Details

For each group with respect to the variables specified by the by_vars parameter, the maximal / minimal observation of analysis_var is labelled in the new_var column as "Y", if its param_var is in worst_high / worst_low. Otherwise, it is assigned NA. If there is more than one such maximal / minimal observation, the first one with respect to the order specified by the order parameter is flagged. The direction of "worst" depends on the definition of worst for a specified parameters in the arguments worst_high / worst_low, i.e. for some parameters the highest value is the worst and for others the worst is the lowest value.

Value

The input dataset with the new flag variable added.

Author(s)

Ondrej Slama

See Also

[derive_var_extreme_flag\(\)](#)

Examples

```
input <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~PARAMCD, ~AVISIT, ~ADT, ~AVAL,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-27"), 15.0,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-25"), 14.0,
  "TEST01", "PAT01", "PARAM01", "BASELINE", as.Date("2021-04-23"), 15.0,
  "TEST01", "PAT01", "PARAM01", "WEEK 1", as.Date("2021-04-27"), 10.0,
  "TEST01", "PAT01", "PARAM01", "WEEK 2", as.Date("2021-04-30"), 12.0,
  "TEST01", "PAT02", "PARAM01", "SCREENING", as.Date("2021-04-27"), 15.0,
  "TEST01", "PAT02", "PARAM01", "BASELINE", as.Date("2021-04-25"), 14.0,
  "TEST01", "PAT02", "PARAM01", "BASELINE", as.Date("2021-04-23"), 15.0,
  "TEST01", "PAT02", "PARAM01", "WEEK 1", as.Date("2021-04-27"), 10.0,
  "TEST01", "PAT02", "PARAM01", "WEEK 2", as.Date("2021-04-30"), 12.0,
```

```

"TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-27"), 15.0,
"TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-25"), 14.0,
"TEST01", "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-23"), 15.0,
"TEST01", "PAT01", "PARAM02", "BASELINE", as.Date("2021-04-27"), 10.0,
"TEST01", "PAT01", "PARAM02", "WEEK 2", as.Date("2021-04-30"), 12.0,
"TEST01", "PAT02", "PARAM02", "SCREENING", as.Date("2021-04-27"), 15.0,
"TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-25"), 14.0,
"TEST01", "PAT02", "PARAM02", "WEEK 1", as.Date("2021-04-23"), 15.0,
"TEST01", "PAT02", "PARAM02", "WEEK 1", as.Date("2021-04-27"), 10.0,
"TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-30"), 12.0,
"TEST01", "PAT02", "PARAM03", "SCREENING", as.Date("2021-04-27"), 15.0,
"TEST01", "PAT02", "PARAM03", "BASELINE", as.Date("2021-04-25"), 14.0,
"TEST01", "PAT02", "PARAM03", "WEEK 1", as.Date("2021-04-23"), 15.0,
"TEST01", "PAT02", "PARAM03", "WEEK 1", as.Date("2021-04-27"), 10.0,
"TEST01", "PAT02", "PARAM03", "BASELINE", as.Date("2021-04-30"), 12.0
)

derive_var_worst_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD, AVISIT),
  order = vars(desc(ADT)),
  new_var = WORSTFL,
  param_var = PARAMCD,
  analysis_var = AVAL,
  worst_high = c("PARAM01", "PARAM03"),
  worst_low = "PARAM02"
)
## Not run:
# example with ADVS
restrict_derivation(
  advs,
  derivation = derive_var_worst_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD, AVISIT),
    order = vars(ADT, ATPTN),
    new_var = WORSTFL,
    param_var = PARAMCD,
    analysis_var = AVAL,
    worst_high = c("SYSBP", "DIABP"),
    worst_low = "RESP"
  ),
  filter = !is.na(AVISIT) & !is.na(AVAL)
)

## End(Not run)

```

Description

These pre-defined dose frequencies are sourced from **CDISC**. The number of rows to generate using `create_single_dose_dataset()` arguments `start_date` and `end_date` is derived from `DOSE_COUNT`, `DOSE_WINDOW`, and `CONVERSION_FACTOR` with appropriate functions from `lubridate`.

Usage

```
dose_freq_lookup
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 86 rows and 5 columns.

Details

`NCI_CODE` and `CDISC_VALUE` are included from the **CDISC** source for traceability.

`DOSE_COUNT` represents the number of doses received in one single unit of `DOSE_WINDOW`. For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `DOSE_WINDOW=="MONTH"` and `DOSE_COUNT==10`. Similarly, for `CDISC_VALUE=="EVERY 2 WEEKS"`, `DOSE_WINDOW=="WEEK"` and `DOSE_COUNT==0.5` (to yield one dose every two weeks).

`CONVERSION_FACTOR` is used to convert `DOSE_WINDOW` units "WEEK", "MONTH", and "YEAR" to the unit "DAY".

For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `CONVERSION_FACTOR` is 0.0329. One day of a month is assumed to be 1 / 30.4375 of a month (one day is assumed to be 1/365.25 of a year). Given only `start_date` and `end_date` in the aggregate dataset, `CONVERSION_FACTOR` is used to calculate specific dates for `start_date` and `end_date` in the resulting single dose dataset for the doses that occur. In such cases, doses are assumed to occur at evenly spaced increments over the interval.

To see the entire table in the console, run `print(dose_freq_lookup)`.

See Also

[create_single_dose_dataset\(\)](#)

dquote

Wrap a String in Double Quotes

Description

Wrap a string in double quotes, e.g., for displaying character values in messages.

Usage

```
dquote(x)
```

Arguments

x A character vector

Value

If the input is NULL, the text "NULL" is returned. Otherwise, the input in double quotes is returned.

Author(s)

Stefan Bundfuss

Examples

```
admiral:::dquote("foo")
admiral:::dquote(NULL)
```

dthcaus_source *Create a dthcaus_source Object*

Description

Create a dthcaus_source Object

Usage

```
dthcaus_source(
  dataset_name,
  filter,
  date,
  mode = "first",
  dthcaus,
  traceability_vars = NULL
)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the death cause.
filter	An expression used for filtering dataset.
date	A character vector to be used for sorting dataset.
mode	One of "first" or "last". Either the "first" or "last" observation is preserved from the dataset which is ordered by date.
dthcaus	A variable name or a string literal — if a variable name, e.g., AEDECOD, it is the variable in the source dataset to be used to assign values to DTHCAUS; if a string literal, e.g. "Adverse Event", it is the fixed value to be assigned to DTHCAUS.

traceability_vars

A named list returned by `vars()` listing the traceability variables, e.g. `vars(DTHDOM = "DS", DTHSEQ = DSSEQ)`. The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

Value

An object of class "dthcaus_source".

Author(s)

Shimeng Huang

See Also

[derive_var_dthcaus\(\)](#)

event_source

Create an event_source Object

Description

event_source objects are used to define events as input for the `derive_param_tte()` function.

Usage

```
event_source(dataset_name, filter = NULL, date, set_values_to = NULL)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of <code>derive_param_tte()</code> .
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points.
date	A variable providing the date of the event or censoring. A date, a datetime, or a character variable containing ISO 8601 dates can be specified. An unquoted symbol is expected. Refer to <code>derive_vars_dt()</code> to impute and derive a date from a date character vector to a date object.
set_values_to	A named list returned by <code>vars()</code> defining the variables to be set for the event or censoring, e.g. <code>vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT")</code> . The values must be a symbol, a character string, a numeric value, or NA.

Value

An object of class `event_source`, inheriting from class `tte_source`

Author(s)

Stefan Bundfuss

See Also

[derive_param_tte\(\)](#), [censor_source\(\)](#)

Examples

```
# Death event
event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = vars(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)
```

expect_dfs_equal

Expectation: Are Two Datasets Equal?

Description

Uses `diffdf::diffdf()` to compares 2 datasets for any differences

Usage

```
expect_dfs_equal(base, compare, keys, ...)
```

Arguments

<code>base</code>	Input dataset
<code>compare</code>	Comparison dataset
<code>keys</code>	character vector of variables that define a unique row in the base and compare datasets
<code>...</code>	Additional arguments passed onto <code>diffdf::diffdf()</code>

Value

An error if base and compare do not match or NULL invisibly if they do

Author(s)

Thomas Neitmann

Examples

```
## Not run:
testthat::test_that("a missing row is detected", {
  data(dm)
  expect_dfs_equal(dm, dm[-1L, ], keys = "USUBJID")
})

## End(Not run)
```

extend_source_datasets

Add By Groups to All Datasets if Necessary

Description

The function ensures that the by variables are contained in all source datasets.

Usage

```
extend_source_datasets(source_datasets, by_vars)
```

Arguments

source_datasets

Source datasets

A named list of datasets is expected. Each dataset must contain either all by variables or none of the by variables.

by_vars

By variables

Details

1. The by groups are determined as the union of the by groups occurring in the source datasets.
2. For all source datasets which do not contain the by variables the source dataset is replaced by the cartesian product of the source dataset and the by groups.

Value

The list of extended source datasets

Author(s)

Stefan Bundfuss

Examples

```

library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adsl <- tibble::tribble(
  ~USUBJID, ~TRTSDT, ~EOSDT,
  "01", ymd("2020-12-06"), ymd("2021-03-06"),
  "02", ymd("2021-01-16"), ymd("2021-02-03")
) %>%
  mutate(STUDYID = "AB42")

ae <- tibble::tribble(
  ~USUBJID, ~AESTDTC, ~AESEQ, ~AEDECOD,
  "01", "2021-01-03T10:56", 1, "Flu",
  "01", "2021-03-04", 2, "Cough",
  "01", "2021", 3, "Flu"
) %>%
  mutate(STUDYID = "AB42")

extend_source_datasets(
  source_datasets = list(adsl = adsl, ae = ae),
  by_vars = vars(AEDECOD)
)

```

extract_duplicate_records

Extract Duplicate Records

Description

Extract Duplicate Records

Usage

```
extract_duplicate_records(dataset, by_vars)
```

Arguments

dataset	A data frame
by_vars	A list of variables created using vars() identifying groups of records in which to look for duplicates

Value

A data frame of duplicate records within dataset

Author(s)

Thomas Neitmann

Examples

```
data(admiral_adsl)

# Duplicate the first record
adsl <- rbind(admiral_adsl[1L, ], admiral_adsl)

extract_duplicate_records(adsl, vars(USUBJID))
```

extract_unit

Extract Unit From Parameter Description

Description

Extract the unit of a parameter from a description like "Param (unit)".

Extract the unit of a parameter from a description like "Param (unit)".

Usage

```
extract_unit(x)
```

```
extract_unit(x)
```

Arguments

x A parameter description

Value

A string

Examples

```
extract_unit("Height (cm)")

extract_unit("Diastolic Blood Pressure (mmHg)")
extract_unit("Height (cm)")

extract_unit("Diastolic Blood Pressure (mmHg)")
```

ex_single	<i>Single Dose Exposure Dataset</i>
-----------	-------------------------------------

Description

A derived dataset with single dose per date.

Usage

```
ex_single
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 22439 rows and 17 columns.

Source

Derived from the `ex` dataset using `{admiral}` and `{dplyr}` (https://github.com/pharmaverse/admiral/blob/main/inst/example_scripts/derive_single_dose.R)

filter_date_sources	<i>Select the First or Last Date from Several Sources</i>
---------------------	---

Description

Select for each subject the first or last observation with respect to a date from a list of sources.

Usage

```
filter_date_sources(
  sources,
  source_datasets,
  by_vars,
  create_datetime = FALSE,
  subject_keys,
  mode
)
```

Arguments

<code>sources</code>	Sources A list of <code>tte_source()</code> objects is expected.
<code>source_datasets</code>	Source datasets A named list of datasets is expected. The <code>dataset_name</code> field of <code>tte_source()</code> refers to the dataset provided in the list.

by_vars	By variables If the parameter is specified, for each by group the observations are selected separately.
create_datetime	Create datetime variable? If set to TRUE, variables ADTM is created. Otherwise, variables ADT is created.
subject_keys	Variables to uniquely identify a subject A list of symbols created using vars() is expected.
mode	Selection mode (first or last) If "first" is specified, for each subject the first observation with respect to the date is included in the output dataset. If "last" is specified, the last observation is included in the output dataset. Permitted Values: "first", "last"

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the filter element are selected. Then for each patient the first or last observation (with respect to date) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied. If the source variable is a character variable, it is converted to a date. If the date is incomplete, it is imputed as the first possible date.
3. The CNSR is added and set to the value of the censor element.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the ADT variable) from the single dataset is selected.

Value

A dataset with one observation per subject as described in the "Details" section.

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

ads1 <- tibble::tribble(
  ~USUBJID, ~TRTSDT, ~EOSDT,
  "01", ymd("2020-12-06"), ymd("2021-03-06"),
  "02", ymd("2021-01-16"), ymd("2021-02-03")
) %>%
  mutate(STUDYID = "AB42")
```

```

ae <- tibble::tribble(
  ~USUBJID, ~AESTDTC, ~AESEQ, ~AEDECOD,
  "01", "2021-01-03T10:56", 1, "Flu",
  "01", "2021-03-04", 2, "Cough",
  "01", "2021", 3, "Flu"
) %>%
  mutate(STUDYID = "AB42")

ttae <- event_source(
  dataset_name = "ae",
  date = AESTDTC,
  set_values_to = vars(
    EVNTDESC = "AE",
    SRCDOM = "AE",
    SRCVAR = "AESTDTC",
    SRCSEQ = AESEQ
  )
)

filter_date_sources(
  sources = list(ttae),
  source_datasets = list(adsl = adsl, ae = ae),
  by_vars = vars(AEDECOD),
  create_datetime = FALSE,
  subject_keys = vars(STUDYID, USUBJID),
  mode = "first"
)

```

filter_extreme

Filter the First or Last Observation for Each By Group

Description

Filters the first or last observation for each by group.

Usage

```
filter_extreme(dataset, by_vars = NULL, order, mode, check_type = "warning")
```

Arguments

dataset	Input dataset The variables specified by the order and the by_vars parameter are expected.
by_vars	Grouping variables <i>Default:</i> NULL <i>Permitted Values:</i> list of variables created by vars()

order	Sort order Within each by group the observations are ordered by the specified order. <i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL))
mode	Selection mode (first or last) If "first" is specified, the first observation of each by group is included in the output dataset. If "last" is specified, the last observation of each by group is included in the output dataset. <i>Permitted Values:</i> "first", "last"
check_type	Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order. <i>Default:</i> "warning" <i>Permitted Values:</i> "none", "warning", "error"

Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is included in the output dataset.

Value

A dataset containing the first or last observation of each by group

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflict = FALSE)
library(admiral.test)
data("admiral_ex")

# Select first dose for each patient
admiral_ex %>%
  filter_extreme(
    by_vars = vars(USUBJID),
    order = vars(EXSEQ),
    mode = "first"
  ) %>%
  select(USUBJID, EXSEQ)

# Select highest dose for each patient on the active drug
admiral_ex %>%
  filter(EXTRT != "PLACEBO") %>%
  filter_extreme(
```

```
    by_vars = vars(USUBJID),
    order = vars(EXDOSE),
    mode = "last",
    check_type = "none"
  ) %>%
  select(USUBJID, EXTRT, EXDOSE)
```

filter_if

Optional Filter

Description

Filters the input dataset if the provided expression is not NULL

Usage

```
filter_if(dataset, filter)
```

Arguments

dataset	Input dataset
filter	A filter condition. Must be a quosure.

Value

A data.frame containing all rows in dataset matching filter or just dataset if filter is NULL

Author(s)

Thomas Neitmann

Examples

```
library(admiral.test)
data(admiral_vs)

admiral::filter_if(admiral_vs, rlang::quo(NULL))
admiral::filter_if(admiral_vs, rlang::quo(VSTESTCD == "WEIGHT"))
```

filter_relative	<i>Filter the Observations Before or After a Condition is Fulfilled</i>
-----------------	---

Description

Filters the observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to select for each subject all observations before the first disease progression.

Usage

```
filter_relative(
  dataset,
  by_vars,
  order,
  condition,
  mode,
  selection,
  inclusive,
  keep_no_ref_groups = TRUE,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset
by_vars	The variables specified by the order and the by_vars parameter are expected. Grouping variables <i>Permitted Values:</i> list of variables created by vars()
order	Sort order Within each by group the observations are ordered by the specified order. <i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL))
condition	Condition for Reference Observation The specified condition determines the reference observation. The output dataset contains all observations before or after (selection parameter) the reference observation.
mode	Selection mode (first or last) If "first" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the <i>first</i> time is included in the output dataset. If "last" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the <i>last</i> time is included in the output dataset. <i>Permitted Values:</i> "first", "last"

selection	Select observations before or after the reference observation? <i>Permitted Values:</i> "before", "after"
inclusive	Include the reference observation? <i>Permitted Values:</i> TRUE, FALSE
keep_no_ref_groups	Should by groups without reference observation be kept? <i>Default:</i> TRUE <i>Permitted Values:</i> TRUE, FALSE
check_type	Check uniqueness? If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order. <i>Default:</i> "none" <i>Permitted Values:</i> "none", "warning", "error"

Details

For each by group (`by_vars` parameter) the observations before or after (`selection` parameter) the observations where the condition (`condition` parameter) if fulfilled the first or last time (`order` parameter and `mode` parameter) is included in the output dataset.

Value

A dataset containing for each by group the observations before or after the observation where the condition was fulfilled the first or last time

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflict = FALSE)

response <- tibble::tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,      "PR",
  "1",      2,      "CR",
  "1",      3,      "CR",
  "1",      4,      "SD",
  "1",      5,      "NE",
  "2",      1,      "SD",
  "2",      2,      "PD",
  "2",      3,      "PD",
  "3",      1,      "SD",
  "4",      1,      "SD",
  "4",      2,      "PR",
  "4",      3,      "PD",
  "4",      4,      "SD",
```

```

    "4",      5,      "PR"
  )

# Select observations up to first PD for each patient
response %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Select observations after last CR, PR, or SD for each patient
response %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC %in% c("CR", "PR", "SD"),
    mode = "last",
    selection = "after",
    inclusive = FALSE
  )

# Select observations from first response to first PD
response %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC %in% c("CR", "PR"),
    mode = "first",
    selection = "after",
    inclusive = TRUE,
    keep_no_ref_groups = FALSE
  ) %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

```

Description

The function returns a character representation of a `sdg_select()` object. It can be used for error messages for example.

Usage

```
## S3 method for class 'sdg_select'  
format(x, ...)
```

Arguments

x	A <code>sdg_select()</code> object
...	Not used

Value

A character representation of the `sdg_select()` object

Author(s)

Stefan Bundfuss

See Also

[sdg_select\(\)](#)

Examples

```
format(  
  sdg_select(  
    name = "5-aminosalicylates for ulcerative colitis"  
  )  
)
```

format.smq_select	<i>Returns a Character Representation of a smq_select() Object</i>
-------------------	--

Description

The function returns a character representation of a `smq_select()` object. It can be used for error messages for example.

Usage

```
## S3 method for class 'smq_select'  
format(x, ...)
```

Arguments

x	A smq_select() object
...	Not used

Value

A character representation of the smq_select() object

Author(s)

Stefan Bundfuss

See Also

[smq_select\(\)](#)

Examples

```
format(smq_select(id = 42, scope = "NARROW"))
```

format_eoxxstt_default

Default Format for Disposition Status

Description

Define a function to map the disposition status. To be used as an input for derive_var_disposition_status().

Usage

```
format_eoxxstt_default(status)
```

Arguments

status	the disposition variable used for the mapping (e.g. DSDECOD).
--------	---

Details

Usually this function can not be used with %>%.

Value

A character vector derived based on the values given in status: "COMPLETED" if status is "COMPLETED", "DISCONTINUED" if status is not "COMPLETED" nor NA, "ONGOING" otherwise.

Author(s)

Samia Kabi

See Also[derive_var_disposition_status\(\)](#)**Examples**

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ds")

admiral_dm %>%
  derive_var_disposition_status(
    dataset_ds = admiral_ds,
    new_var = EOSSTT,
    status_var = DSDECOD,
    format_new_var = format_eoxxstt_default,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, EOSSTT)
```

format_reason_default *Default Format for the Disposition Reason*

Description

Define a function to map the disposition reason, to be used as a parameter in `derive_vars_disposition_reason()`.

Usage

```
format_reason_default(reason, reason_spe = NULL)
```

Arguments

reason	the disposition variable used for the mapping (e.g. DSDECOD).
reason_spe	the disposition variable used for the mapping of the details if required (e.g. DSTERM).

Details

`format_reason_default(DSDECOD)` returns DSDECOD when DSDECOD is not 'COMPLETED' nor NA.
`format_reason_default(DSDECOD, DSTERM)` returns DSTERM when DSDECOD is equal to 'OTHER'.
Usually this function can not be used with `%>%`.

Value

A character vector

Author(s)

Samia Kabi

See Also

[derive_vars_disposition_reason\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("admiral_dm")
data("admiral_ds")

# Derive DCSREAS using format_reason_default
admiral_dm %>%
  derive_vars_disposition_reason(
    dataset_ds = admiral_ds,
    new_var = DCSREAS,
    reason_var = DSDECOD,
    format_new_vars = format_reason_default,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, DCSREAS)
```

get_duplicates_dataset

Get Duplicate Records that Lead to a Prior Error

Description

Get Duplicate Records that Lead to a Prior Error

Usage

```
get_duplicates_dataset()
```

Details

Many admiral function check that the input dataset contains only one record per `by_vars` group and throw an error otherwise. The `get_duplicates_dataset()` function allows one to retrieve the duplicate records that lead to an error.

Note that the function always returns the dataset of duplicates from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_duplicates_dataset()` will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A data.frame or NULL

Author(s)

Thomas Neitmann

Examples

```
data(admiral_ads1)

# Duplicate the first record
ads1 <- rbind(admiral_ads1[1L, ], admiral_ads1)

signal_duplicate_records(ads1, vars(USUBJID), cnd_type = "warning")

get_duplicates_dataset()
```

get_many_to_one_dataset

Get Many to One Values that Led to a Prior Error

Description

Get Many to One Values that Led to a Prior Error

Usage

```
get_many_to_one_dataset()
```

Details

If `assert_one_to_one()` detects an issue, the many to one values are stored in a dataset. This dataset can be retrieved by `get_many_to_one_dataset()`.

Note that the function always returns the many to one values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_many_to_one_dataset()` will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A data.frame or NULL

Author(s)

Stefan Bundfuss

Examples

```
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, vars(SITEID), vars(STUDYID))
)

get_many_to_one_dataset()
```

get_one_to_many_dataset

Get One to Many Values that Led to a Prior Error

Description

Get One to Many Values that Led to a Prior Error

Usage

```
get_one_to_many_dataset()
```

Details

If `assert_one_to_one()` detects an issue, the one to many values are stored in a dataset. This dataset can be retrieved by `get_one_to_many_dataset()`.

Note that the function always returns the one to many values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_one_to_many_dataset()` will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A data.frame or NULL

Author(s)

Stefan Bundfuss

Examples

```
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, vars(STUDYID), vars(SITEID))
)

get_one_to_many_dataset()
```

get_terms_from_db *Get Terms from the Queries Database*

Description

The function checks if all requirements to access the database are fulfilled (version and access function are available, see `assert_db_requirements()`), reads the terms from the database, and checks if the dataset with the terms is in the expected format (see `assert_terms()`).

Usage

```
get_terms_from_db(
  version,
  fun,
  queries,
  definition,
  expect_query_name = FALSE,
  expect_query_id = FALSE,
  i,
  temp_env,
  type
)
```

Arguments

version	Version The version must be non null. Otherwise, an error is issued. The value is passed to the access function (<code>fun</code>).
fun	Access function The access function must be non null. Otherwise, an error is issued. The function is called to retrieve the terms.
queries	Queries List of all queries passed to <code>create_query_data()</code> . It is used for error messages.
definition	Definition of the query The definition is passed to the access function. It defines which terms are returned.
expect_query_name	Is QUERY_NAME expected in the output dataset?
expect_query_id	Is QUERY_ID expected in the output dataset?
i	Index of definition in queries The value is used for error messages.
temp_env	Temporary environment The value is passed to the access function.

type Type of query
Permitted Values: "smq", "sdg"

Value

Output dataset of the access function

Author(s)

Stefan Bundfuss

impute_dtc	<i>Impute Partial Date(-time) Portion of a '--DTC' Variable</i>
------------	---

Description

Imputation partial date/time portion of a '--DTC' variable. based on user input.

Usage

```
impute_dtc(
  dtc,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dtc The '--DTC' date to impute
 A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss.
 If the year part is not recorded (missing date), no imputation is performed.

date_imputation
 The value to impute the day/month when a datepart is missing.
 If NULL: no date imputation is performed and partial dates are returned as missing.
 Otherwise, a character value is expected, either as a

- format with month and day specified as "mm-dd": e.g. "06-15" for the 15th of June,
- or as a keyword: "FIRST", "MID", "LAST" to impute to the first/mid/last day/month.

Default is NULL.

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "FIRST", "LAST" to impute to the start/end of a day.

Default is "00:00:00".

min_dates

Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  date_imputation = "first"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

max_dates

Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

preserve

Preserve day if month is missing and day is present

For example "2019--07" would return "2019-06-07" if preserve = TRUE (and date_imputation = "MID").

Permitted Values: TRUE, FALSE

Default: FALSE

Details

Usually this computation function can not be used with %>%.

Value

A character vector

Author(s)

Samia Kabi

Examples

```
library(lubridate)

dates <- c(
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18T15",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019",
  "2019---07",
  ""
)

# No date imputation (date_imputation defaulted to NULL)
# Missing time part imputed with 00:00:00 portion by default
impute_dtc(dtc = dates)

# No date imputation (date_imputation defaulted to NULL)
# Missing time part imputed with 23:59:59 portion
impute_dtc(
  dtc = dates,
  time_imputation = "23:59:59"
)

# Same as above
impute_dtc(
  dtc = dates,
  time_imputation = "LAST"
)

# Impute to first day/month if date is partial
# Missing time part imputed with 00:00:00 portion by default
impute_dtc(
  dtc = dates,
  date_imputation = "01-01"
)

# same as above
impute_dtc(
  dtc = dates,
  date_imputation = "FIRST"
)

# Impute to last day/month if date is partial
# Missing time part imputed with 23:59:59 portion
impute_dtc(
  dtc = dates,
```

```
    date_imputation = "LAST",
    time_imputation = "LAST"
  )

  # Impute to mid day/month if date is partial
  # Missing time part imputed with 00:00:00 portion by default
  impute_dtc(
    dtc = dates,
    date_imputation = "MID"
  )

  # Impute a date and ensure that the imputed date is not before a list of
  # minimum dates
  impute_dtc(
    "2020-12",
    min_dates = list(
      ymd_hms("2020-12-06T12:12:12"),
      ymd_hms("2020-11-11T11:11:11")
    ),
    date_imputation = "first"
  )
}
```

is_auto

Checks if the argument equals the auto keyword

Description

Checks if the argument equals the auto keyword

Usage

```
is_auto(arg)
```

Arguments

arg argument to check

Value

TRUE if the argument equals the auto keyword, i.e., it is a quosure of a symbol named auto.

Author(s)

Stefan Bundfuss

Examples

```
example_fun <- function(arg) {  
  arg <- rlang::enquo(arg)  
  if (admiral:::is_auto(arg)) {  
    "auto keyword was specified"  
  } else {  
    arg  
  }  
}  
  
example_fun("Hello World!")  
  
example_fun(auto)
```

list_all_templates *List All Available ADaM Templates*

Description

List All Available ADaM Templates

Usage

```
list_all_templates()
```

Value

A character vector of all available templates

Author(s)

Shimeng Huang, Thomas Neitmann

Examples

```
list_all_templates()
```

lstalvdt_source *Create an lstalvdt_source object*

Description

[Deprecated]

Deprecated, please use `date_source()` instead.

Usage

```
lstalvdt_source(  
  dataset_name,  
  filter = NULL,  
  date,  
  date_imputation = NULL,  
  traceability_vars = NULL  
)
```

Arguments

- | | |
|--------------------------------|--|
| <code>dataset_name</code> | The name of the dataset, i.e. a string, used to search for the last known alive date. |
| <code>filter</code> | An unquoted condition for filtering dataset. |
| <code>date</code> | A variable providing a date where the patient was known to be alive. A date, a datetime, or a character variable containing ISO 8601 dates can be specified. An unquoted symbol is expected. |
| <code>date_imputation</code> | A string defining the date imputation for date. See <code>date_imputation</code> parameter of <code>impute_dtc()</code> for valid values. |
| <code>traceability_vars</code> | A named list returned by <code>vars()</code> defining the traceability variables, e.g. <code>vars(LALVDOM = "AE", LALVSEQ = AESEQ, LALVVAR = "AESTDTC")</code> . The values must be a symbol, a character string, or NA. |

Value

An object of class `lstalvdt_source`.

Author(s)

Stefan Bundfuss

negate_vars

Negate List of Variables

Description

The function adds a minus sign as prefix to each variable.

Usage

```
negate_vars(vars = NULL)
```

Arguments

vars List of variables created by vars()

Details

This is useful if a list of variables should be removed from a dataset, e.g., `select(!negate_vars(by_vars))` removes all by variables.

Value

A list of quosures

Author(s)

Stefan Bundfuss

Examples

```
negate_vars(vars(USUBJID, STUDYID))
```

params

Create a Set of Parameters

Description

Create a set of variable parameters/function arguments to be used in [call_derivation\(\)](#).

Usage

```
params(...)
```

Arguments

... One or more named arguments

Value

An object of class `params`

Author(s)

Thomas Neitmann, Tracey Wang

See Also

`call_derivation()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_ae)
data(admiral_adsl)

adae <- admiral_ae[sample(1:nrow(admiral_ae), 1000), ] %>%
  select(USUBJID, AESTDTC, AEENDTC) %>%
  derive_vars_merged(
    dataset_add = admiral_adsl,
    new_vars = vars(TRTSDT, TRTEDT),
    by_vars = vars(USUBJID)
  )

## In order to derive both `ASTDT` and `AENDT` in `ADAE`, one can use `derive_vars_dt()`
adae %>%
  derive_vars_dt(
    new_vars_prefix = "AST",
    dtc = AESTDTC,
    date_imputation = "first",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  )

## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go.
## The function arguments which are different from a variable to another (e.g. `new_vars_prefix`,
## `dtc`, and `date_imputation`) are specified as a list of `params()` in the `variable_params`
## argument of `call_derivation()`. All other arguments which are common to all variables
## (e.g. `min_dates` and `max_dates`) are specified outside of `variable_params` (i.e. in `...`).
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
```

```

variable_params = list(
  params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
  params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
),
min_dates = vars(TRTSDT),
max_dates = vars(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the call using `derive_vars_dt()`
## to derive variables `ASTDT` and `AENDT` separately at the beginning.

```

```
print.derivation_slice
```

Print derivation_slice Objects

Description

Print derivation_slice Objects

Usage

```
## S3 method for class 'derivation_slice'
print(x, ...)
```

Arguments

x	A derivation_slice object
...	Not used

Value

No return value, called for side effects

See Also

[derivation_slice\(\)](#)

Examples

```
print(death_event)
```

print.tte_source	<i>Print tte_source Objects</i>
------------------	---------------------------------

Description

Print tte_source Objects

Usage

```
## S3 method for class 'tte_source'  
print(x, ...)
```

Arguments

x	A tte_source object
...	Not used

Value

No return value, called for side effects

See Also

[tte_source\(\)](#), [censor_source\(\)](#), [event_source\(\)](#)

Examples

```
print(death_event)
```

queries	<i>Queries Dataset</i>
---------	------------------------

Description

An example of standard query dataset to be used in deriving variables in ADAE and ADCM

Usage

```
queries
```

Format

An object of class tbl_df (inherits from tbl, data.frame) with 15 rows and 8 columns.

query	<i>Create an query object</i>
-------	-------------------------------

Description

A query object defines a query, e.g., a Standard MedDRA Query (SMQ), a Standardised Drug Grouping (SDG), or a customized query (CQ). It is used as input to `create_query_data()`.

Usage

```
query(prefix, name = auto, id = NULL, add_scope_num = FALSE, definition = NULL)
```

Arguments

- | | |
|---------------|--|
| prefix | The value is used to populate VAR_PREFIX in the output dataset of <code>create_query_data()</code> , e.g., "SMQ03" |
| name | The value is used to populate QUERY_NAME in the output dataset of <code>create_query_data()</code> . If the auto keyword is specified, the variable is set to the name of the query in the SMQ/SDG database.
<i>Permitted Values:</i> A character scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an <code>smq_select()</code> or <code>sdg_select()</code> object. |
| id | The value is used to populate QUERY_ID in the output dataset of <code>create_query_data()</code> . If the auto keyword is specified, the variable is set to the id of the query in the SMQ/SDG database.
<i>Permitted Values:</i> A integer scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an <code>smq_select()</code> or <code>sdg_select()</code> object. |
| add_scope_num | Determines if QUERY_SCOPE_NUM in the output dataset of <code>create_query_data()</code> is populated
If the parameter is set to TRUE, the definition must be an <code>smq_select()</code> object.
<i>Default:</i> FALSE
<i>Permitted Values:</i> TRUE, FALSE |
| definition | Definition of terms belonging to the query
There are four different ways to define the terms: <ul style="list-style-type: none"> • An <code>smq_select()</code> object is specified to select a query from the SMQ database. • An <code>sdg_select()</code> object is specified to select a query from the SDG database. • A data frame with columns TERM_LEVEL and TERM_NAME or TERM_ID can be specified to define the terms of a customized query. The TERM_LEVEL should be set to the name of the variable which should be used to select the terms, e.g., "AEDECOD" or "AELLTCD". TERM_LEVEL does not need to be constant within a query. For example a query can be based on AEDECOD and AELLT. |

If `TERM_LEVEL` refers to a character variable, `TERM_NAME` should be set to the value the variable. If it refers to a numeric variable, `TERM_ID` should be set to the value of the variable. If only character variables or only numeric variables are used, `TERM_ID` or `TERM_NAME` respectively can be omitted.

- A list of data frames and `smq_select()` objects can be specified to define a customized query based on custom terms and SMQs. The data frames must have the same structure as described for the previous item.

Permitted Values: an `smq_select()` object, an `sdg_select()` object, a data frame, or a list of data frames and `smq_select()` objects.

Value

An object of class `query`.

Author(s)

Stefan Bundfuss

See Also

[create_query_data\(\)](#), [smq_select\(\)](#), [sdg_select\(\)](#), [Queries Dataset Documentation](#)

Examples

```
# create a query for an SMQ
library(tibble)
library(magrittr, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)

query(
  prefix = "SMQ02",
  id = auto,
  definition = smq_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW"
  )
)

# create a query for an SDG
query(
  prefix = "SDG01",
  id = auto,
  definition = sdg_select(
    name = "5-aminosalicylates for ulcerative colitis"
  )
)

# creating a query for a customized query
cqterms <- tribble(
  ~TERM_NAME, ~TERM_ID,
```

```

"APPLICATION SITE ERYTHEMA", 10003041L,
"APPLICATION SITE PRURITUS", 10003053L
) %>%
mutate(TERM_LEVEL = "AEDECOD")

query(
  prefix = "CQ01",
  name = "Application Site Issues",
  definition = cqterms
)

# creating a customized query based on SMQs and additional terms
query(
  prefix = "CQ03",
  name = "Special issues of interest",
  definition = list(
    cqterms,
    smq_select(
      name = "Pregnancy and neonatal topics (SMQ)",
      scope = "NARROW"
    ),
    smq_select(
      id = 8050L,
      scope = "BROAD"
    )
  )
)
)

```

restrict_derivation *Execute a Derivation on a Subset of the Input Dataset*

Description

Execute a derivation on a subset of the input dataset.

Usage

```
restrict_derivation(dataset, derivation, args = NULL, filter)
```

Arguments

dataset	Input dataset
derivation	Derivation
args	Arguments of the derivation A params() object is expected.
filter	Filter condition

Author(s)

Stefan Bundfuss

See Also[params\(\)](#) [slice_derivation\(\)](#)**Examples**

```

library(magrittr)
adlb <- tibble::tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~ABLFL,
  "1",      -1,    113, NA_character_,
  "1",      0,    113, "Y",
  "1",      3,    117, NA_character_,
  "2",      0,    95, "Y",
  "3",      0,    111, "Y",
  "3",      1,   101, NA_character_,
  "3",      2,   123, NA_character_
)

# Derive BASE for post-baseline records only (derive_var_base() can not be used in this case
# as it requires the baseline observation to be in the input dataset)
restrict_derivation(
  adlb,
  derivation = derive_vars_merged,
  args = params(
    by_vars = vars(USUBJID),
    dataset_add = adlb,
    filter_add = ABLFL == "Y",
    new_vars = vars(BASE = AVAL)
  ),
  filter = AVISITN > 0
)

# Derive BASE for baseline and post-baseline records only
restrict_derivation(
  adlb,
  derivation = derive_var_base,
  args = params(
    by_vars = vars(USUBJID)
  ),
  filter = AVISITN >= 0
)%>%
# Derive CHG for post-baseline records only
restrict_derivation(
  derivation = derive_var_chg,
  filter = AVISITN > 0
)

```

sdg_select *Create an sdg_select object*

Description

Create an sdg_select object

Usage

```
sdg_select(name = NULL, id = NULL)
```

Arguments

name	Name of the query used to select the definition of the query from the company database.
id	Identifier of the query used to select the definition of the query from the company database.

Details

Exactly one name or id must be specified.

Value

An object of class sdg_select.

Author(s)

Stefan Bundfuss

See Also

[create_query_data\(\)](#), [query\(\)](#)

signal_duplicate_records
Signal Duplicate Records

Description

Signal Duplicate Records

Usage

```

signal_duplicate_records(
  dataset,
  by_vars,
  msg = paste("Dataset contains duplicate records with respect to",
    enumerate(vars2chr(by_vars))),
  cnd_type = "error"
)

```

Arguments

dataset	A data frame
by_vars	A list of variables created using vars() identifying groups of records in which to look for duplicates
msg	The condition message
cnd_type	Type of condition to signal when detecting duplicate records. One of "message", "warning" or "error". Default is "error".

Value

No return value, called for side effects

Author(s)

Thomas Neitmann

Examples

```

data(admiral_adsl)

# Duplicate the first record
adsl <- rbind(admiral_adsl[1L, ], admiral_adsl)

signal_duplicate_records(adsl, vars(USUBJID), cnd_type = "message")

```

slice_derivation	<i>Execute a Derivation with Different Arguments for Subsets of the Input Dataset</i>
------------------	---

Description

The input dataset is split into slices (subsets) and for each slice the derivation is called separately. Some or all arguments of the derivation may vary depending on the slice.

Usage

```

slice_derivation(dataset, derivation, args = NULL, ...)

```

Arguments

dataset	Input dataset
derivation	Derivation
args	Arguments of the derivation A param() object is expected.
...	A derivation_slice() object is expected Each slice defines a subset of the input dataset and some of the parameters for the derivation. The derivation is called on the subset with the parameters specified by the args parameter and the args field of the derivation_slice() object. If a parameter is specified for both, the value in derivation_slice() overwrites the one in args.

Details

For each slice the derivation is called on the subset defined by the filter field of the derivation_slice() object and with the parameters specified by the args parameter and the args field of the derivation_slice() object. If a parameter is specified for both, the value in derivation_slice() overwrites the one in args.

- Observations that match with more than one slice are only considered for the first matching slice.
- Observations with no match to any of the slices are included in the output dataset but the derivation is not called for them.

Value

The input dataset with the variables derived by the derivation added

Author(s)

Stefan Bundfuss

See Also

[params\(\)](#) [restrict_derivation\(\)](#)

Examples

```
library(stringr)
advs <- tibble::tribble(
  ~USUBJID, ~VSDTC,      ~VSTPT,
  "1",      "2020-04-16", NA_character_,
  "1",      "2020-04-16", "BEFORE TREATMENT"
)

# For the second slice filter is set to TRUE. Thus derive_vars_dtm is called
# with time_imputation = "last" for all observations which do not match for the
# first slice.
```

```

slice_derivation(
  advs,
  derivation = derive_vars_dtm,
  args = params(
    dtc = VSDTC,
    new_vars_prefix = "A"
  ),
  derivation_slice(
    filter = str_detect(VSTPT, "PRE|BEFORE"),
    args = params(time_imputation = "first")
  ),
  derivation_slice(
    filter = TRUE,
    args = params(time_imputation = "last")
  )
)

```

smq_select

Create an smq_select object

Description

Create an smq_select object

Usage

```
smq_select(name = NULL, id = NULL, scope = NULL)
```

Arguments

name	Name of the query used to select the definition of the query from the company database.
id	Identifier of the query used to select the definition of the query from the company database.
scope	Scope of the query used to select the definition of the query from the company database. <i>Permitted Values: "BROAD", "NARROW"</i>

Details

Exactly one of name or id must be specified.

Value

An object of class smq_select.

Author(s)

Stefan Bundfuss

See Also

[create_query_data\(\)](#), [query\(\)](#)

suppress_warning	<i>Suppress Specific Warnings</i>
------------------	-----------------------------------

Description

Suppress certain warnings issued by an expression.

Usage

```
suppress_warning(expr, regexpr)
```

Arguments

expr	Expression to be executed
regexpr	Regular expression matching warnings to suppress

Details

All warnings which are issued by the expression and match the regular expression are suppressed.

Value

Return value of the expression

Author(s)

- Thomas Neitmann
- Stefan Bundfuss

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(admiral_adsl)
data(admiral_vs)

# Remove label
attr(admiral_vs$USUBJID, "label") <- NULL

left_join(admiral_adsl, admiral_vs, by = "USUBJID")

suppress_warning(
  left_join(admiral_adsl, admiral_vs, by = "USUBJID"),
  "^Column `USUBJID` has different attributes on LHS and RHS of join$"
)
```

tte_source	<i>Create a tte_source Object</i>
------------	-----------------------------------

Description

The tte_source object is used to define events and possible censorings.

Usage

```
tte_source(dataset_name, filter = NULL, date, censor = 0, set_values_to = NULL)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte().
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points.
date	A variable providing the date of the event or censoring. A date, a datetime, or a character variable containing ISO 8601 dates can be specified. An unquoted symbol is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
censor	Censoring value CDISC strongly recommends using 0 for events and positive integers for censoring.
set_values_to	A named list returned by vars() defining the variables to be set for the event or censoring, e.g. vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, or NA.

Value

An object of class tte_source

Author(s)

Stefan Bundfuss

See Also

[derive_param_tte\(\)](#), [censor_source\(\)](#), [event_source\(\)](#)

use_ad_template	<i>Open an ADaM Template Script</i>
-----------------	-------------------------------------

Description

Open an ADaM Template Script

Usage

```
use_ad_template(  
  adam_name = "adsl",  
  save_path = paste0("./", adam_name, ".R"),  
  overwrite = FALSE,  
  open = interactive()  
)
```

Arguments

adam_name	An ADaM dataset name. You can use any of the available dataset name ADAE, ADCM, ADEG, ADEX, ADLB, ADPP, ADSL, ADVS, and the dataset name is case-insensitive. The default dataset name is ADSL.
save_path	Path to save the script.
overwrite	Whether to overwrite an existing file named save_path.
open	Whether to open the script right away.

Details

Running without any arguments such as use_ad_template() auto-generates adsl.R in the current path. Use list_all_templates() to discover which templates are available.

Value

No return values, called for side effects

Author(s)

Shimeng Huang, Thomas Neitmann

Examples

```
if (interactive()) {  
  use_ad_template("adsl")  
}
```

validate_query	<i>Validate an object is indeed a query object</i>
----------------	--

Description

Validate an object is indeed a query object

Usage

```
validate_query(obj)
```

Arguments

obj An object to be validated.

Value

The original object.

Author(s)

Stefan Bundfuss

See Also

[query\(\)](#)

validate_sdg_select	<i>Validate an object is indeed a sdg_select object</i>
---------------------	---

Description

Validate an object is indeed a sdg_select object

Usage

```
validate_sdg_select(obj)
```

Arguments

obj An object to be validated.

Value

The original object.

Author(s)

Stefan Bundfuss

See Also

[sdg_select\(\)](#)

validate_smq_select *Validate an object is indeed a smq_select object*

Description

Validate an object is indeed a smq_select object

Usage

```
validate_smq_select(obj)
```

Arguments

obj An object to be validated.

Value

The original object.

Author(s)

Stefan Bundfuss

See Also

[smq_select\(\)](#)

Value

a warning if the 2 lists have different names or length

Author(s)

Samia Kabi

Examples

```
# no warning
warn_if_inconsistent_list(
  base = vars(DTHDOM = "DM", DTHSEQ = DMSEQ),
  compare = vars(DTHDOM = "DM", DTHSEQ = DMSEQ),
  list_name = "Test"
)
# warning
warn_if_inconsistent_list(
  base = vars(DTHDOM = "DM", DTHSEQ = DMSEQ, DTHVAR = "text"),
  compare = vars(DTHDOM = "DM", DTHSEQ = DMSEQ),
  list_name = "Test"
)
```

warn_if_invalid_dtc *Warn If a Vector Contains Unknown Datetime Format*

Description

Warn if the vector contains unknown datetime format such as "2003-12-15T-:15:18", "2003-12-15T13:-:19", "-12-15", "—T07:15"

Usage

```
warn_if_invalid_dtc(dtc, is_valid = is_valid_dtc(dtc))
```

Arguments

`dtc` a character vector containing the dates
`is_valid` a logical vector indicating whether elements in `dtc` are valid

Value

No return value, called for side effects

Author(s)

Samia Kabi

Examples

```
## No warning as `dte` is a valid date format
warn_if_invalid_dtc(dtc = "2021-04-06")

## Issues a warning
warn_if_invalid_dtc(dtc = "2021-04-06T-:30:30")
```

warn_if_vars_exist	<i>Warn If a Variable Already Exists</i>
--------------------	--

Description

Warn if a variable already exists inside a dataset

Usage

```
warn_if_vars_exist(dataset, vars)
```

Arguments

dataset	A data.frame
vars	character vector of columns to check for in dataset

Value

No return value, called for side effects

Author(s)

Thomas Neitmann

Examples

```
library(admiral.test)
data(admiral_dm)

## No warning as `AAGE` doesn't exist in `dm`
warn_if_vars_exist(admiral_dm, "AAGE")

## Issues a warning
warn_if_vars_exist(admiral_dm, "ARM")
```

yn_to_numeric	<i>Map "Y" and "N" to Numeric Values</i>
---------------	--

Description

Map "Y" and "N" to numeric values.

Usage

```
yn_to_numeric(arg)
```

Arguments

arg Character vector

Value

1 if arg equals "Y", 0 if arg equals "N", NA_real_ otherwise

Author(s)

Stefan Bundfuss

Examples

```
yn_to_numeric(c("Y", "N", NA_character_))
```

Index

- * **ADaM**
 - derive_var_atirel, 145
 - derive_vars_dy, 118
- * **ATIREL**
 - derive_var_atirel, 145
- * **BMI**
 - compute_bmi, 33
- * **BSA**
 - compute_bsa, 34
- * **Relationship**
 - derive_var_atirel, 145
- * **Var**
 - derive_var_atirel, 145
- * **adae**
 - create_query_data, 49
 - create_single_dose_dataset, 53
 - derive_last_dose, 66
 - derive_vars_query, 133
- * **adam**
 - compute_bmi, 33
 - compute_bsa, 34
 - compute_duration, 36
 - derive_var_analysis_ratio, 141
 - derive_var_extreme_flag, 163
 - derive_var_last_dose_amt, 167
 - derive_var_last_dose_date, 170
 - derive_var_last_dose_grp, 172
 - derive_var_merged_cat, 175
 - derive_var_merged_character, 177
 - derive_var_merged_exist_flag, 180
 - derive_var_obs_number, 182
 - derive_var_shift, 188
 - derive_var_worst_flag, 193
 - derive_vars_dt, 106
 - derive_vars_dtm, 110
 - derive_vars_dtm_to_dt, 114
 - derive_vars_dtm_to_tm, 115
 - derive_vars_duration, 116
 - derive_vars_last_dose, 120
 - derive_vars_merged, 122
 - derive_vars_merged_dt, 125
 - derive_vars_merged_dtm, 129
 - derive_vars_transposed, 135
 - filter_extreme, 205
 - filter_relative, 208
- * **adcm**
 - create_query_data, 49
 - derive_vars_atc, 102
 - derive_vars_query, 133
- * **adeg**
 - compute_qtc, 39
 - compute_rr, 40
 - derive_param_qtc, 87
 - derive_param_rr, 89
- * **adex**
 - create_single_dose_dataset, 53
 - derive_param_doseint, 73
 - derive_param_exposure, 79
- * **adlb**
 - derive_param_wbc_abs, 95
 - derive_var_analysis_ratio, 141
 - derive_var_shift, 188
- * **adsl**
 - derive_var_disposition_dt, 151
 - derive_var_disposition_status, 152
 - derive_var_dthcaus, 154
 - derive_var_extreme_dt, 157
 - derive_var_extreme_dtm, 160
 - derive_var_lstalvdt, 174
 - derive_var_trtdurd, 189
 - derive_var_trtedtm, 191
 - derive_var_trtsdtm, 192
 - derive_vars_disposition_reason, 103
 - format_eoxxstt_default, 212
 - format_reason_default, 213
- * **advs**
 - compute_map, 38

- derive_param_bmi, 69
- derive_param_bsa, 71
- derive_param_map, 84
- * **assertion**
 - assert_character_scalar, 9
 - assert_character_vector, 10
 - assert_data_frame, 11
 - assert_filter_cond, 13
 - assert_function, 14
 - assert_has_variables, 15
 - assert_integer_scalar, 16
 - assert_list_element, 17
 - assert_list_of, 18
 - assert_logical_scalar, 19
 - assert_numeric_vector, 20
 - assert_one_to_one, 21
 - assert_order_vars, 21
 - assert_param_does_not_exist, 22
 - assert_s3_class, 23
 - assert_symbol, 24
 - assert_terms, 25
 - assert_unit, 26
 - assert_valid_queries, 27
 - assert_vars, 28
 - assert_varval_list, 29
- * **bds**
 - derive_derived_param, 60
 - derive_extreme_records, 63
 - derive_param_exist_flag, 75
 - derive_param_exposure, 79
 - derive_param_first_event, 81
 - derive_param_tte, 91
 - derive_param_wbc_abs, 95
 - derive_summary_records, 98
 - derive_var_ady, 137
 - derive_var_aendy, 138
 - derive_var_analysis_ratio, 141
 - derive_var_anrind, 143
 - derive_var_astdy, 144
 - derive_var_base, 146
 - derive_var_basetype, 148
 - derive_var_chg, 150
 - derive_var_ontrtfl, 183
 - derive_var_pchg, 187
 - derive_var_shift, 188
- * **check**
 - is_auto, 221
- * **computation**
 - compute_bmi, 33
 - compute_bsa, 34
 - compute_dtf, 35
 - compute_duration, 36
 - compute_map, 38
 - compute_qtc, 39
 - compute_rr, 40
 - compute_tmf, 41
 - convert_date_to_dtm, 43
 - convert_dtc_to_dt, 45
 - convert_dtc_to_dtm, 47
 - format_eoxxstt_default, 212
 - format_reason_default, 213
 - impute_dtc, 218
- * **datasets**
 - admiral_adae, 6
 - admiral_adcm, 6
 - admiral_adeq, 7
 - admiral_adex, 7
 - admiral_adpp, 8
 - admiral_ads1, 8
 - admiral_adv9, 9
 - ex_single, 203
 - queries, 227
- * **derivation**
 - derive_derived_param, 60
 - derive_extreme_records, 63
 - derive_last_dose, 66
 - derive_param_bmi, 69
 - derive_param_bsa, 71
 - derive_param_doseint, 73
 - derive_param_exist_flag, 75
 - derive_param_exposure, 79
 - derive_param_first_event, 81
 - derive_param_map, 84
 - derive_param_qtc, 87
 - derive_param_rr, 89
 - derive_param_tte, 91
 - derive_param_wbc_abs, 95
 - derive_summary_records, 98
 - derive_var_ady, 137
 - derive_var_aendy, 138
 - derive_var_analysis_ratio, 141
 - derive_var_anrind, 143
 - derive_var_astdy, 144
 - derive_var_base, 146
 - derive_var_basetype, 148
 - derive_var_chg, 150

- derive_var_dthcaus, 154
- derive_var_extreme_dt, 157
- derive_var_extreme_dtm, 160
- derive_var_extreme_flag, 163
- derive_var_last_dose_amt, 167
- derive_var_last_dose_date, 170
- derive_var_last_dose_grp, 172
- derive_var_lstalvdt, 174
- derive_var_merged_cat, 175
- derive_var_merged_character, 177
- derive_var_merged_exist_flag, 180
- derive_var_obs_number, 182
- derive_var_ontrtfl, 183
- derive_var_pchg, 187
- derive_var_shift, 188
- derive_var_trtdurd, 189
- derive_var_trtedtm, 191
- derive_var_trtsdtm, 192
- derive_var_worst_flag, 193
- derive_vars_atc, 102
- derive_vars_dt, 106
- derive_vars_dtm, 110
- derive_vars_duration, 116
- derive_vars_dy, 118
- derive_vars_last_dose, 120
- derive_vars_merged, 122
- derive_vars_merged_dt, 125
- derive_vars_merged_dtm, 129
- derive_vars_query, 133
- derive_vars_transposed, 135
- * **dev_utility**
 - call_user_fun, 31
 - dataset_vignette, 55
 - dquote, 196
 - expect_dfs_equal, 199
 - extend_source_datasets, 200
 - extract_duplicate_records, 201
 - filter_date_sources, 203
 - format.sdg_select, 210
 - format.smq_select, 211
 - signal_duplicate_records, 232
 - tte_source, 237
- * **high_order_function**
 - call_derivation, 30
 - restrict_derivation, 230
 - slice_derivation, 233
- * **metadata**
 - dose_freq_lookup, 195
- * **occds**
 - derive_var_aendy, 138
 - derive_var_astdy, 144
- * **source_specifications**
 - censor_source, 32
 - date_source, 56
 - derivation_slice, 60
 - dthcaus_source, 197
 - event_source, 198
 - lstalvdt_source, 223
 - params, 224
 - query, 228
 - sdg_select, 232
 - smq_select, 235
- * **test_helper**
 - expect_dfs_equal, 199
- * **timing**
 - compute_dtf, 35
 - compute_duration, 36
 - compute_tmf, 41
 - convert_date_to_dtm, 43
 - convert_dtc_to_dt, 45
 - convert_dtc_to_dtm, 47
 - derive_var_ady, 137
 - derive_var_aendy, 138
 - derive_var_astdy, 144
 - derive_var_disposition_dt, 151
 - derive_var_trtdurd, 189
 - derive_var_trtedtm, 191
 - derive_var_trtsdtm, 192
 - derive_vars_dt, 106
 - derive_vars_dtm, 110
 - derive_vars_dtm_to_dt, 114
 - derive_vars_dtm_to_tm, 115
 - derive_vars_duration, 116
 - derive_vars_dy, 118
 - derive_vars_merged_dt, 125
 - derive_vars_merged_dtm, 129
 - impute_dtc, 218
- * **tte_source**
 - death_event, 58
- * **user_utility**
 - call_derivation, 30
 - convert_blanks_to_na, 42
 - create_query_data, 49
 - create_single_dose_dataset, 53
 - default_qtc_paramcd, 59
 - derive_vars_last_dose, 120

- extract_unit, 202
- filter_extreme, 205
- filter_if, 207
- filter_relative, 208
- format_eoxsstt_default, 212
- format_reason_default, 213
- get_duplicates_dataset, 214
- get_many_to_one_dataset, 215
- get_one_to_many_dataset, 216
- list_all_templates, 222
- negate_vars, 224
- restrict_derivation, 230
- slice_derivation, 233
- use_ad_template, 238
- vars2chr, 241
- yn_to_numeric, 244
- * warning**
 - suppress_warning, 236
 - warn_if_inconsistent_list, 241
 - warn_if_invalid_dtc, 242
 - warn_if_vars_exist, 243
- admiral_adae, 6
- admiral_adcm, 6
- admiral_adeq, 7
- admiral_adex, 7
- admiral_adpp, 8
- admiral_adsl, 8
- admiral_advs, 9
- ae_event (death_event), 58
- ae_gr1_event (death_event), 58
- ae_gr2_event (death_event), 58
- ae_gr35_event (death_event), 58
- ae_gr3_event (death_event), 58
- ae_gr4_event (death_event), 58
- ae_gr5_event (death_event), 58
- ae_ser_event (death_event), 58
- ae_sev_event (death_event), 58
- ae_wd_event (death_event), 58
- assert_character_scalar, 9
- assert_character_vector, 10
- assert_data_frame, 11
- assert_db_requirements, 12
- assert_filter_cond, 13
- assert_function, 14
- assert_has_variables, 15
- assert_integer_scalar, 16
- assert_list_element, 17
- assert_list_of, 18
- assert_logical_scalar, 19
- assert_numeric_vector, 20
- assert_one_to_one, 21
- assert_order_vars, 21
- assert_param_does_not_exist, 22
- assert_s3_class, 23
- assert_symbol, 24
- assert_terms, 25
- assert_unit, 26
- assert_valid_queries, 27
- assert_valid_queries(), 133, 134
- assert_vars, 28
- assert_varval_list, 29
- call_derivation, 30
- call_derivation(), 224, 225
- call_user_fun, 31
- cancel_source, 32
- cancel_source(), 59, 199, 227, 237
- compute_bmi, 33
- compute_bsa, 34
- compute_dtf, 35
- compute_duration, 36
- compute_duration(), 118
- compute_map, 38
- compute_qtc, 39
- compute_qtc(), 88
- compute_rr, 40
- compute_tmf, 41
- convert_blanks_to_na, 42
- convert_date_to_dtm, 43
- convert_dtc_to_dt, 45
- convert_dtc_to_dtm, 47
- create_query_data, 49
- create_query_data(), 25, 134, 229, 232, 236
- create_single_dose_dataset, 53
- create_single_dose_dataset(), 169, 171, 173, 196
- cut(), 173
- dataset_vignette, 55
- date_source, 56
- date_source(), 158, 161
- death_event, 58
- default_qtc_paramcd, 59
- derivation_slice, 60
- derivation_slice(), 226
- derive_derived_param, 60

- derive_extreme_records, 63
- derive_last_dose, 66
- derive_param_bmi, 69
- derive_param_bsa, 71
- derive_param_doseint, 73
- derive_param_exist_flag, 75
- derive_param_exposure, 79
- derive_param_first_event, 81
- derive_param_map, 84
- derive_param_qtc, 87
- derive_param_rr, 89
- derive_param_tte, 91
- derive_param_tte(), 33, 59, 199, 237
- derive_param_wbc_abs, 95
- derive_summary_records, 98
- derive_var_ady, 137
- derive_var_aendy, 138
- derive_var_age_years, 140
- derive_var_agegr_ema
 - (derive_var_agegr_fda), 139
- derive_var_agegr_fda, 139
- derive_var_analysis_ratio, 141
- derive_var_anrind, 143
- derive_var_astdy, 144
- derive_var_atirel, 145
- derive_var_base, 146
- derive_var_basetype, 148
- derive_var_chg, 150
- derive_var_chg(), 187
- derive_var_disposition_dt, 151
- derive_var_disposition_status, 152
- derive_var_disposition_status(), 213
- derive_var_dthcaus, 154
- derive_var_dthcaus(), 198
- derive_var_extreme_dt, 157
- derive_var_extreme_dt(), 57, 161
- derive_var_extreme_dtm, 160
- derive_var_extreme_dtm(), 57, 158
- derive_var_extreme_flag, 163
- derive_var_extreme_flag(), 194
- derive_var_last_dose_amt, 167
- derive_var_last_dose_date, 170
- derive_var_last_dose_grp, 172
- derive_var_lstalvdt, 174
- derive_var_merged_cat, 175
- derive_var_merged_character, 177
- derive_var_merged_exist_flag, 180
- derive_var_obs_number, 182
- derive_var_ontrtfl, 183
- derive_var_pchg, 187
- derive_var_pchg(), 150
- derive_var_shift, 188
- derive_var_trtdurd, 189
- derive_var_trtedtm, 191
- derive_var_trtsdtm, 192
- derive_var_worst_flag, 193
- derive_var_worst_flag(), 165
- derive_vars_aage, 100
- derive_vars_atc, 102
- derive_vars_disposition_reason, 103
- derive_vars_disposition_reason(), 214
- derive_vars_dt, 106
- derive_vars_dtm, 110
- derive_vars_dtm_to_dt, 114
- derive_vars_dtm_to_tm, 115
- derive_vars_duration, 116
- derive_vars_duration(), 102, 190
- derive_vars_dy, 118
- derive_vars_last_dose, 120
- derive_vars_last_dose(), 169, 171, 173
- derive_vars_merged, 122
- derive_vars_merged(), 158, 161
- derive_vars_merged_dt, 125
- derive_vars_merged_dt(), 158, 161
- derive_vars_merged_dtm, 129
- derive_vars_merged_dtm(), 158, 161
- derive_vars_query, 133
- derive_vars_query(), 51
- derive_vars_suppqual, 134
- derive_vars_transposed, 135
- diffdf::diffdf(), 199
- dose_freq_lookup, 195
- dquote, 196
- dthcaus_source, 197
- dthcaus_source(), 155
- event_source, 198
- event_source(), 33, 59, 227, 237
- ex_single, 203
- expect_dfs_equal, 199
- extend_source_datasets, 200
- extract_duplicate_records, 201
- extract_unit, 202
- filter_date_sources, 203
- filter_extreme, 205
- filter_if, 207

filter_relative, 208
format.sdg_select, 210
format.smq_select, 211
format_eoxxstt_default, 212
format_reason_default, 213
format_reason_default(), 105

get_duplicates_dataset, 214
get_many_to_one_dataset, 215
get_one_to_many_dataset, 216
get_terms_from_db, 217

here, 54

impute_dtc, 67, 121, 218
impute_dtc(), 47
is_auto, 221

lastalive_censor (death_event), 58
list_all_templates, 222
lstalvdt_source, 223

negate_vars, 224

params, 224
params(), 30, 60, 231, 234
print.derivation_slice, 226
print.tte_source, 227

queries, 227
query, 228
query(), 25, 51, 232, 236, 239

restrict_derivation, 230
restrict_derivation(), 234

sdg_select, 232
sdg_select(), 51, 211, 229, 240
signal_duplicate_records, 232
slice_derivation, 233
slice_derivation(), 60, 231
smq_select, 235
smq_select(), 51, 212, 229, 240
suppress_warning, 236

tte_source, 237
tte_source(), 59, 227

use_ad_template, 238

validate_query, 239
validate_sdg_select, 239
validate_smq_select, 240
vars(), 67, 98, 120, 121, 168, 171, 173, 198, 241
vars2chr, 241

warn_if_inconsistent_list, 241
warn_if_invalid_dtc, 242
warn_if_vars_exist, 243

yn_to_numeric, 244