Package 'admiral'

February 17, 2022

Type Package

Title ADaM in R Asset Library

Version 0.6.3

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>).

License Apache License (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Depends R (>= 3.5)

Imports admiral.test, assertthat, dplyr, lubridate, magrittr, purrr, rlang, stringr, hms, tidyr, tidyselect

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

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Neitmann [aut, cre],
Alice Ehmann [aut],
Stefan Bundfuss [aut],
Samia Kabi [aut],
Vignesh Thanikachalam [aut],
Ondrej Slama [aut],
Shimeng Huang [aut],
Robin Koeger [ctb],
Gordon Miller [ctb],
Teckla Akinyi [aut],

Eric Simms [aut],
Michael Thorpe [aut],
Pavan Kumar [ctb],
Hamza Rahal [ctb],
Annie Yang [aut],
Ross Farrugia [ctb],
Ben Straub [aut],
Andrew Smith [aut],
Konstantina Koukourikou [ctb],
Tom Ratford [ctb],
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline LLC [cph, fnd]

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

Repository CRAN

Date/Publication 2022-02-17 19:52:03 UTC

R topics documented:

`+,iso_dtm,Period-method` *Add Methods*

Description

Add Methods

Usage

```
## S4 method for signature 'iso_dtm,Period'
e1 + e2

## S4 method for signature 'Period,iso_dtm'
e1 + e2

## S4 method for signature 'iso_dtm,Duration'
e1 + e2

## S4 method for signature 'Duration,iso_dtm'
e1 + e2
```

Arguments

`e1`	First object
`e2`	Second object

`adae` *Adverse Event Analysis Dataset*

Description

An example adverse event analysis dataset

Usage

```
adae
```

Format

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

Source

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

---

adcm *Concomitant Medication Analysis Dataset*

Description

An example concomitant medication analysis dataset

Usage

```
adcm
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7510 rows and 90 columns.

Source

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

---

adex *Exposure Analysis Dataset*

Description

An example exposure analysis dataset

Usage

```
adex
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 6315 rows and 88 columns.

Source

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

adsl *Subject Level Analysis Dataset*

Description

An example subject level analysis dataset

Usage

```
adsl
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 306 rows and 49 columns.

Source

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

advs *Vital Signs Analysis Dataset*

Description

An example vital signs analysis dataset

Usage

```
advs
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 57763 rows and 103 columns.

Source

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

`arg`	A function argument to be checked
`values`	A `character` vector of valid values for `arg`
`case_sensitive`	Should the argument be handled case-sensitive? If set to `FALSE`, the argument is converted to lower case for checking the permitted values and returning the argument.
`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 `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 is 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 is `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(dm)

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

example_fun(dm)

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

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

---

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(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(dm, AGE == 64)

try(example_fun(dm, USUBJID))
```

---

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(dm)

assert_has_variables(dm, "STUDYID")
## Not run:
assert_has_variables(dm, "AVAL")

## End(Not run)
```

`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., censor == 0 to check the censor 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 censor 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"
  )
)

try(assert_list_element(
  list = list(death, lstalv),
  element = "censor",
  condition = censor == 0,
  message_text = "For events the censor values must be zero."
))

try(assert_list_element(
  list = events,
  element = "dataset_name",
  condition = dataset_name %in% c("adrs", "adae"),
  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)
```

Arguments

`arg`	A function argument to be checked

Value

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

Author(s)

Thomas Neitmann

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(adsl)
try(
  assert_one_to_one(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(exprs(USUBJID, PARAMCD)))

try(example_fun(c("USUBJID", "PARAMCD", "VISIT")))

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(advs)
assert_param_does_not_exist(advs, param = "HR")
try(assert_param_does_not_exist(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 *Is an Argument a Symbol?*

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(dm)

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

example_fun(dm, USUBJID)

try(example_fun(dm))

try(example_fun(dm, "USUBJID"))

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

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(advs)
assert_unit(advs, param = "WEIGHT", required_unit = "kg", get_unit_expr = VSSTRESU)
## Not run:
assert_unit(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 *Is an Argument a List of Variables?*

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(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 it 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` *Call a Single Derivation Multiple Times*

Description

Call a single derivation multiple times with some parameters 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 arguments that are different across iterations. Each set of arguments must be created using `params()`.
`...`	Any number of *named* arguments that are fixed across iterations. If a parameter is specified both inside `variable_params` and `...` then the value in `variable_params` overwrites the one in `...`

Value

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

Author(s)

Thomas Neitmann, Stefan Bundfuss

See Also

`params()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data(ae)
data(adsl)

adae <- ae[sample(1:nrow(ae), 1000), ] %>%
  left_join(adsl, by = "USUBJID") %>%
  select(USUBJID, AESTDTC, AEENDTC, TRTSDT, TRTEDT)

## 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)
  )
```

`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.
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

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(height * weight / 3600) DuBois-DuBois: 0.20247 * (height/100) ^ 0.725 * weight ^ 0.425 Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378 Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456 Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight))) Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444 Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425 Permitted Values: character value

Value

The BSA (Body Surface Area) in m^2.

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.

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 *Derive Duration*

Description

Derives duration between two dates, e.g., duration of adverse events, 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	The start date A date or date-time object is expected.
end_date	The end date A date or date-time object is expected.
in_unit	Input unit See floor_in and add_one parameter for details. Default: 'days' Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'
out_unit	Output unit The duration is derived in the specified unit Default: 'days' Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'
floor_in	Round down input dates? 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. Default: 'TRUE" Permitted Values: TRUE, FALSE
add_one	Add one input unit? If the duration is non-negative, one input unit is added. I.e., the duration can not be zero. Default: TRUE Permitted Values: TRUE, FALSE
trunc_out	Return integer part The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned. Default: FALSE Permitted Values: TRUE, FALSE

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.

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

diabp	Diastolic blood pressure A numeric vector is expected.
sysbp	Systolic blood pressure A numeric vector is expected.
hr	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.

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:

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

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.

Value

RR interval in msec:

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

Author(s)

Stefan Bundfuss

Examples

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

compute_tmf *Derive the Time Imputation Flag*

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

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 = NULL,
  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. 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.

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

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. 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.

`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`

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 = 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-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` 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. 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.

`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`

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")
```

`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 theChoose 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("dm")

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

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")
```

derive_agegr_fda *Derive Age Groups*

Description

[Deprecated]

Deprecated, please use `derive_var_agegr_fda()` and `derive_var_agegr_ema()` instead.

Usage

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

derive_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.

Value

`dataset` with new column `new_var` of class factor.

Author(s)

Ondrej Slama

derive_baseline *Derive Baseline*

Description

[Deprecated]

This function is *deprecated*. Please use `derive_var_base()` instead.

Usage

```
derive_baseline(dataset, by_vars, source_var, new_var)
```

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`

Value

The input dataset with variable `new_var` added

Author(s)

Thomas Neitmann

See Also

`derive_var_base()`

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	Input dataset The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by parameters.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. *Permitted Values:* list of variables
parameters	Required parameter codes It is expected that all parameter codes (PARAMCD) which are required to derive the new parameter are specified for this parameter or the constant_parameters parameter. *Permitted Values:* A character vector of PARAMCD values
analysis_value	Definition of the analysis value An expression defining the analysis value (AVAL) of the new parameter is expected. The analysis values of the parameters specified by parameters can be accessed using AVAL.<parameter code>, e.g., AVAL.SYSBP. *Permitted Values:* An unquoted expression
set_values_to	Variables to be set 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. *Permitted Values:* List of variable-value pairs
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. *Permitted Values:* a condition
constant_by_vars	By variables for constant parameters

The constant parameters are merged to the other parameters using the specified variables. This is useful if some parameters were measured only once. For example if BMI should be derived and height is measured only once while weight is measured at each visit. Height could be specified for the constant parameters.

Permitted Values: list of variables

`constant_parameters`

Required constant parameter codes

It is expected that all parameter codes (`PARAMCD`) which are required to derive the new parameter are specified for this parameter or the `parameters` parameter.

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

Author(s)

Stefan Bundfuss

Examples

```
# 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"
  )
)

# derive BMI where height is measured only once
advs <- 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(
  advs,
  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_disposition_dt` *Derive a Disposition Date*

Description

[Deprecated]

Deprecated, please use `derive_var_disposition_dt()` instead.

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

Usage

```
derive_disposition_dt(
  dataset,
  dataset_ds,
  new_var,
  dtc,
  filter_ds,
```

```
  date_imputation = NULL,
  subject_keys = vars(STUDYID, USUBJID)
)
```

Arguments

dataset	Input dataset
dataset_ds	Datasets containing the disposition information (e.g.: ds) It must contain: • 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	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 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. Default is NULL
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 date (new_var) added

Author(s)

Samia Kabi

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("dm")
data("ds")

dm %>%
  derive_disposition_dt(
    dataset_ds = ds,
    new_var = FRVDT,
    dtc = DSSTDTC,
    filter_ds = DSCAT == "OTHER EVENT" & DSDECOD == "FINAL RETRIEVAL VISIT"
  ) %>%
  select(STUDYID, USUBJID, FRVDT)
```

---

derive_disposition_reason

Derive a Disposition Reason at a Specific Timepoint

Description

[Deprecated]

Deprecated, please use `derive_vars_disposition_reason()` instead.

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

Usage

```
derive_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`). It must contain: • `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

- 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. format_reason <- function(reason) and new_var will be derived as new_var = format_reason(reason_var) Typically, the content of the function would return reason_var or NA depending on the value (e.g. if_else (reason != "COMPLETED" & !is.na(reason),reason,NA_character_)). DCSREAS = format_reason(DSDECOD) returns DCSREAS = DSDECOD when DSDECOD 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. format_reason2 <- function(reason, reason_spe) and new_var will be derived as new_var = format_reason(reason_var), new_var_spe will be derived as new_var_spe = format_reason(reason_var,reason_var_spe), Typically, the content of the function would return reason_var_spe or NA depending on the reason_var value (e.g. if_else (reason != "COMPLETED" & !is.na(reason),reason_spe,NA_character_)). DCSREASP = format_reason(DSDECOD,DSTER returns DCSREASP = DSTERM when DSDECOD is not 'COMPLETED' nor NA.

Default: format_reason_default defined as: format_reason_default <- function(reason, reason_spe = NULL) out <- if (is.null(reason_spe)) reason else reason_spe if_else (reason != "COMPLETED" & !is.na(reason), out, NA_character_)

format_reason_default(DSDECOD) returns DSDECOD when DSDECOD is not 'COMPLETED' nor NA. format_reason_default(DSDECOD, DSTERM) returns DSTERM when DSDECOD is not 'COMPLETED' nor NA.

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("dm")
data("ds")

# Derive DCSREAS using the default format
dm %>%
  derive_disposition_reason(
    dataset_ds = 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) {
  out <- if (is.null(y)) x else y
  case_when(
    !(x %in% c("COMPLETED", "SCREEN FAILURE")) & !is.na(x) ~ out,
    TRUE ~ NA_character_
```

```
  )
}
dm %>%
  derive_disposition_reason(
    dataset_ds = 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_disposition_status` *Derive a Disposition Status at a Specific Timepoint*

Description

[Deprecated]

Deprecated, please use `derive_var_disposition_status()` instead.

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

Usage

```
derive_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: • `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:

```
format_eoxxstt_default <- function(x) {
  case_when(
    x == "COMPLETED" ~ "COMPLETED",
    x != "COMPLETED" & !is.na(x) ~ "DISCONTINUED",
    TRUE ~ "ONGOING"
  )
}
```

where x is the status_var.

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("dm")
data("ds")

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

dm %>%
  derive_disposition_status(
    dataset_ds = 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"
  )
}

dm %>%
  derive_disposition_status(
    dataset_ds = ds,
    new_var = EOSSTT,
    status_var = DSDECOD,
    format_new_var = format_eoxxstt1,
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, EOSSTT)
```

`derive_extreme_flag` *Add a Variable Flagging the First or Last Observation Within Each By Group*

Description

[Deprecated]

Deprecated, please use `derive_var_extreme_flag()` instead.

Add a variable flagging the first or last observation within each by group

Usage

```
derive_extreme_flag(
  dataset,
  by_vars,
  order,
  new_var,
  mode,
  filter = NULL,
  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	Filter for flag data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. Permitted Values: a condition
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.

Value

The input dataset with the new flag variable added

Author(s)

Stefan Bundfuss

Examples

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

# Flag last value for each patient, test, and visit, baseline observations are ignored
vs %>%
  derive_extreme_flag(
    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
derive_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(ADT),
  new_var = ABLFL,
```

```
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = High
derive_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(AVAL, ADT),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = Lo
derive_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(desc(AVAL), ADT),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Average observation
derive_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(ADT, desc(AVAL)),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE" & DTYPE == "AVERAGE"
)
```

`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 dplyr::vars).
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 by_vars 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 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. 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(ae)
data(ex_single)

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
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_obs_number` *Adds a Variable Numbering the Observations Within Each By Group*

Description

[Deprecated]

Deprecated, please use `derive_var_obs_number()` instead.

Adds a variable numbering the observations within each by group

Usage

```
derive_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("vs")

vs %>%
  select(USUBJID, VSTESTCD, VISITNUM, VSTPTNUM) %>%
  filter(VSTESTCD %in% c("HEIGHT", "WEIGHT")) %>%
  derive_obs_number(
    by_vars = vars(USUBJID, VSTESTCD),
    order = vars(VISITNUM, VSTPTNUM)
  )
```

---

derive_params_exposure

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

---

Description

[Deprecated]

This function is *deprecated*. Please use derive_param_exposure() instead.

Usage

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

Arguments

dataset	Input dataset • The variables specified by the by_vars,analysis_var parameters and PARAMCD are expected, • Either ASTDTM and AENDTM or ASTDT and AENDT are also expected.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. *Permitted Values:* list of variables
input_code	Required parameter code The observations where PARAMCD equals the specified value are considered to compute the summary record. *Permitted Values:* A character of PARAMCD value
analysis_var	Analysis variable.
summary_fun	Function that takes as an input the analysis_var and performs the calculation. This can include built-in functions as well as user defined functions, for example mean or function(x) mean(x,na.rm = TRUE).
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. *Permitted Values:* 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

Value

The input dataset with a new record added for each group (with respect to the variables specified for the by_vars parameter).

Author(s)

Samia Kabi

See Also

derive_param_exposure()

---

derive_param_bmi *Adds a Parameter for BMI*

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 by_vars parameter, PARAMCD, and AVAL are expected. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by weight_code and height_code.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. *Permitted Values:* 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 vars(PARAMCD = "MAP") defines the parameter code for the new parameter. *Permitted Values:* List of variable-value pairs

weight_code	WEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the WEIGHT. It is expected that WEIGHT is measured in kg Permitted Values: character value
height_code	HEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the HEIGHT. It is expected that HEIGHT is measured in cm Permitted Values: 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. *Permitted Values:* 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

Author(s)

Pavan Kumar

Examples

```
advs <- tibble::tribble(
  ~USUBJID,      ~PARAMCD, ~PARAM,        ~AVAL, ~AVALU, ~AVISIT,
  "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_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` *Adds a Parameter for BSA (Body Surface Area) Using the Specified Method*

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`	Input dataset The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `HEIGHT` and `WEIGHT`.
`by_vars`	Grouping variables For each group defined by `by_vars` an observation is added to the output dataset. *Permitted Values:* list of variables
`method`	Derivation method to use. Note that `HEIGHT` is expected in cm and `WEIGHT` is expected in kg: Mosteller: `sqrt(height * weight / 3600)` DuBois-DuBois: `0.20247 * (height/100) ^ 0.725 * weight ^ 0.425` Haycock: `0.024265 * height ^ 0.3964 * weight ^ 0.5378`

Gehan-George: `0.0235 * height ^ 0.42246 * weight ^ 0.51456`

Boyd: `0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 -(0.0188 * log10(1000 * weight)))`

Fujimoto: `0.008883 * height ^ 0.663 * weight ^ 0.444`

Takahira: `0.007241 * height ^ 0.725 * weight ^ 0.425`

Permitted Values: character value

`set_values_to` Variables to be set

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.

Permitted Values: List of variable-value pairs

`height_code` HEIGHT parameter code

The observations where `PARAMCD` equals the specified value are considered as the HEIGHT assessments. It is expected that HEIGHT is measured in cm.

Permitted Values: character value

`weight_code` WEIGHT parameter code

The observations where `PARAMCD` equals the specified value are considered as the WEIGHT assessments. It is expected that WEIGHT is measured in kg.

Permitted Values: 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.

Permitted Values: a condition

Value

The input dataset with the new parameter added

Author(s)

Eric Simms

Examples

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

```
  "01-701-1028", "WEIGHT", "Weight (kg)",  85,   "kg",   "MONTH 2",
)

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

derive_param_bsa(
  advs,
  by_vars = vars(USUBJID, VISIT),
  method = "Fujimoto",
  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 `by_vars` parameter, `PARAMCD`, and `AVAL` are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `tadm_code` and `padm_code`.
`by_vars`	Grouping variables Permitted Values: 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 `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted Values: List of variable-value pairs

`tadm_code` Total Doses Administered parameter code

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.

Permitted Values: character value

`tpadm_code` Total Doses Planned parameter code

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.

Permitted Values: character value

`zero_doses` Flag indicating logic for handling 0 planned or administered doses for a `by_vars` group

Default: `Inf`

Permitted Values: `Inf`, `100`

No record is returned if either the planned (`tpadm_code`) or administered (`tadm_code`) `AVAL` are `NA`. No record is returned is a record does not exist for both `tadm_code` and `tpadm_code` for the specified `by_var`.

If `zero_doses = Inf`:

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.

If `zero_doses = 100` :

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` 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.

Permitted Values: a condition

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

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 = "TDOSINT2"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
  zero_doses = "100")
```

---

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

dataset	Input dataset • The variables specified by the by_vars,analysis_var parameters and PARAMCD are expected, • Either ASTDTM and AENDTM or ASTDT and AENDT are also expected.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. *Permitted Values:* list of variables
input_code	Required parameter code The observations where PARAMCD equals the specified value are considered to compute the summary record. *Permitted Values:* A character of PARAMCD value
analysis_var	Analysis variable.
summary_fun	Function that takes as an input the analysis_var and performs the calculation. This can include built-in functions as well as user defined functions, for example mean or function(x) mean(x,na.rm = TRUE).
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. *Permitted Values:* 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). 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_map` *Adds a Parameter for Mean Arterial Pressure*

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`	Input dataset The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `sysbp_code`, `diabp_code` and `hr_code`.

`by_vars` Grouping variables

For each group defined by `by_vars` an observation is added to the output dataset.

Permitted Values: 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 `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted Values: List of variable-value pairs

`sysbp_code` Systolic blood pressure parameter code

The observations where `PARAMCD` equals the specified value are considered as the systolic blood pressure assessments.

Permitted Values: character value

`diabp_code` Diastolic blood pressure parameter code

The observations where `PARAMCD` equals the specified value are considered as the diastolic blood pressure assessments.

Permitted Values: character value

`hr_code` Heart rate parameter code

The observations where `PARAMCD` equals the specified value are considered as the heart rate assessments.

Permitted Values: 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.

Permitted Values: a condition

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

Author(s)

Stefan Bundfuss

Examples

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

advs <- tibble::tribble(
  ~USUBJID,      ~PARAMCD, ~PARAM,                              ~AVAL, ~AVALU,      ~VISIT,
  "01-701-1015", "PULSE",  "Pulse (beats/min)"           ,  59,  "beats/min", "BASELINE",
  "01-701-1015", "PULSE",  "Pulse (beats/min)"             ,  61,  "beats/min", "WEEK 2",
  "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", "PULSE",  "Pulse (beats/min)"           ,  62,  "beats/min", "BASELINE",
  "01-701-1028", "PULSE",  "Pulse (beats/min)"             ,  77,  "beats/min", "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 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 = AVALU
  ) %>%
  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*

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 by_vars and the unit_var parameter, PARAMCD, and AVAL are expected. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by qt_code and rr_code.
by_vars	Grouping variables 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 vars(PARAMCD = "MAP") defines the parameter code for the new parameter. *Permitted Values:* List of variable-value pairs
qt_code	QT parameter code The observations where PARAMCD equals the specified value are considered as the QT interval assessments. It is expected that QT is measured in msec. Permitted Values: character value
rr_code	RR parameter code The observations where PARAMCD equals the specified value are considered as the RR interval assessments. It is expected that RR is measured in msec. Permitted Values: 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. *Permitted Values:* a condition

Value

The input dataset with the new parameter added

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` *Adds a Parameter for Derived RR*

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`	Input dataset The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `hr_code`.
`by_vars`	Grouping variables For each group defined by `by_vars` an observation is added to the output dataset. *Permitted Values:* 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 `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter. *Permitted Values:* List of variable-value pairs

hr_code	HR parameter code The observations where PARAMCD equals the specified value are considered as the heart rate assessments. Permitted Values: 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. *Permitted Values:* a condition

Details

The analysis value of the new parameter is derived as

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

Value

The input dataset with the new parameter added

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,
  start_date_imputation_flag = NULL,
  start_time_imputation_flag = NULL,
  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 specified for `start_imputation_flag`.

`start_date_imputation_flag`

Date imputation flag for start date

If the start date is imputed, the corresponding date imputation flag must be specified. The variable `STARTDTF` is set to the specified variable.

`start_time_imputation_flag`

Time imputation flag for start date

If the start time is imputed, the corresponding time imputation flag must be specified. The variable `STARTTMF` is set to the specified variable.

`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, warn.conflicts = FALSE)
data("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_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 vars() 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 by_vars as this will help when an aggregating, lagging, or ranking function is involved. For example, • filter = (AVAL > mean(AVAL,na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars. • filter = (dplyr::n() > 2) will filter n count of by_vars greater than 2.
analysis_var	Analysis variable.
summary_fun	Function that takes as an input the analysis_var and performs the calculation. This can include built-in functions as well as user defined functions, for example mean or function(x) mean(x,na.rm = TRUE).
set_values_to	A list of variable name-value pairs. Use this argument if you need to change the values of any newly derived records. Set a list of variables to some specified value for the new observation(s) • LHS refer 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

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")
)
```

`derive_vars_aage` *Derive Analysis Age*

Description

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

Usage

```
derive_vars_aage(
  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. Default: BRTHDT
end_date	The end date A date or date-time object is expected. 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 *Derive ATC Class Variables*

Description

Add Anatomical Therapeutic Chemical class variables from FACM to ADCM

Usage

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

Arguments

dataset	Input dataset The variables specified by the by_vars parameter are required
dataset_facm	FACM dataset The variables specified by the by_vars parameter, FAGRPID, FATESTCD and FASTRESC are required
by_vars	Keys used to merge dataset_facm with dataset *Permitted Values:* 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: • `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

  - 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. format_reason <- function(reason) and new_var will be derived as new_var = format_reason(reason_var) Typically, the content of the function would return reason_var or NA depending on the value (e.g. if_else (reason != "COMPLETED" & !is.na(reason),reason,NA_character_)). DCSREAS = format_reason(DSDECOD) returns DCSREAS = DSDECOD when DSDECOD 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. format_reason2 <- function(reason, reason_spe) and new_var will be derived as new_var = format_reason(reason_var), new_var_spe will be derived as new_var_spe = format_reason(reason_var,reason_var_spe), Typically, the content of the function would return reason_var_spe or NA depending on the reason_var value (e.g. if_else (reason != "COMPLETED" & !is.na(reason),reason_spe,NA_character_)). DCSREASP = format_reason(DSDECOD,DSTER returns DCSREASP = DSTERM when DSDECOD is not 'COMPLETED' nor NA.

  Default: format_reason_default defined as: format_reason_default <- function(reason, reason_spe = NULL) out <- if (is.null(reason_spe)) reason else reason_spe if_else (reason != "COMPLETED" & !is.na(reason), out, NA_character_)

  format_reason_default(DSDECOD) returns DSDECOD when DSDECOD is not 'COMPLETED' nor NA. format_reason_default(DSDECOD, DSTERM) returns DSTERM when DSDECOD is not 'COMPLETED' nor NA.

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("dm")
data("ds")

# Derive DCSREAS using the default format
dm %>%
  derive_vars_disposition_reason(
    dataset_ds = 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) {
  out <- if (is.null(y)) x else y
  case_when(
    !(x %in% c("COMPLETED", "SCREEN FAILURE")) & !is.na(x) ~ out,
    TRUE ~ NA_character_
  )
}
dm %>%
  derive_vars_disposition_reason(
    dataset_ds = 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 = TRUE,
  min_dates = NULL,
  max_dates = NULL,
  preserve = 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 is expected: e.g. `new_vars_prefix = "AST"`.

`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.

flag_imputation

Whether the date imputation flag should also be derived.

A logical value

Default: TRUE

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. 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.

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

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 = FALSE
)

# 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_dtm(
  mhdt,
  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 is expected: e.g. `new_vars_prefix = "AST"`.

`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.

One of "auto", "date" or "both"

Default: "auto"

`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. 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.

`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`

`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

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, 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 columns 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	The start date A date or date-time object is expected.
end_date	The end date A date or date-time object is expected.
in_unit	Input unit See floor_in and add_one parameter for details. Default: 'days' Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'
out_unit	Output unit The duration is derived in the specified unit Default: 'days' Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds'
floor_in	Round down input dates? 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. Default: 'TRUE" Permitted Values: TRUE, FALSE
add_one	Add one input unit? If the duration is non-negative, one input unit is added. I.e., the duration can not be zero. Default: TRUE Permitted Values: TRUE, FALSE
trunc_out	Return integer part The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned. Default: FALSE Permitted Values: TRUE, FALSE

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.

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

dataset	Input dataset The columns specified by the reference_date and the source_vars parameter are expected.
reference_date	The start date column, e.g., date of first treatment A date or date-time object column is expected.

source_vars — A list of datetime or date variables created using `vars()` 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. `vars(TRTSDTM,ASTDTM,AENDT)` or `vars(TRTSDT,ASTDTM,AENDT,DEATHDY = DTHDT)`. 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:O9", "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 by_vars and analysis_date parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars 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 dplyr::vars).
dose_id	Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars().
dose_date	The EX dose date variable.
analysis_date	The analysis date variable.
single_dose_condition	The condition for checking if dataset_ex is single dose. An error is issued if the condition is not true. Defaults to (EXDOSFRQ == "ONCE").
new_vars	Variables to keep from dataset_ex, with the option to rename. Can either be variables created by dplyr::vars (e.g. vars(VISIT)), or named list returned by vars() (e.g. vars(LSTEXVIS = VISIT)). If set to NULL, then all variables from dataset_ex are kept without renaming. Defaults to NULL.

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(ae)
data(ex_single)

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
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_query *Derive Query Variables*

Description

Derive Query Variables

Usage

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

Arguments

dataset	Input dataset.

dataset_queries

A data.frame 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().

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

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

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.

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.

Details

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

Value

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

Author(s)

Vignesh Thanikachalam

Examples

```
## The following example includes selected variables from AE and SUPPAE
## datasets for a rash whose locations are the face, neck, and chest.
ae <- tibble::tribble(
  ~STUDYID,   ~DOMAIN, ~USUBJID,   ~AESEQ, ~AETERM,  ~AELOC,
  "1234-005", "AE",    "XYZ-1001",      1, "RASH",  "MULTIPLE",
  "1234-005", "AE",    "XYZ-1002",      1, "NAUSEA", "",
)
suppae <- tibble::tribble(
  ~STUDYID,   ~RDOMAIN, ~USUBJID,    ~IDVAR,  ~IDVARVAL, ~QNAM,     ~QLABEL,     ~QVAL,
  "1234-005", "AE",     "XYZ-1001", "AESEQ", "1",        "AELOC1", "Location 1", "FACE",
```

```
  "1234-005", "AE",     "XYZ-1001", "AESEQ", "1",        "AELOC2", "Location 2", "NECK",
  "1234-005", "AE",     "XYZ-1001", "AESEQ", "1",       "AELOC3", "Location 3", "CHEST",
)
derive_vars_suppqual(ae, suppae)

## The following example included subjects with multiple/other specific race.
dm <- tibble::tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~RACE,
  "ABC",    "DM",    "001",    "OTHER",
  "ABC",    "DM",    "002",    "MULTIPLE",
  "ABC",    "DM",    "003",    NA,
  "ABC",    "DM",    "004",    "ASIAN"
)
suppdm <- tibble::tribble(
  ~STUDYID, ~RDOMAIN, ~USUBJID, ~IDVAR, ~IDVARVAL, ~QNAM,     ~QLABEL,       ~QVAL,
  "ABC",   "DM",      "001",     "",     "",       "RACEOTH", "Race, Other", "BRAZILIAN",
  "ABC",   "DM",      "002",     "",     "",       "RACE1"  , "Race 1",      "AMERICAN",
  "ABC",   "DM",      "002",     "",     "",       "RACE2"  , "Race 2",      "OTHER",
  "ABC",   "DM",      "002",     "",     "",       "RACEOTH", "Race, Other", "ABORIGINE"
)
derive_vars_suppqual(dm, suppdm)
```

`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

[Questioning]

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 `reference_date` and the `date` parameter are expected.
`reference_date`	The start date column, e.g., date of first treatment A date or date-time object column is expected. 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. 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

Examples

```
data <- tibble::tribble(
  ~TRTSDT, ~ADT,
  lubridate::ymd("2020-01-01"), lubridate::ymd("2020-02-24")
)

derive_var_ady(data)
```

---

derive_var_aendy *Derive Analysis End Relative Day*

Description

[Questioning]

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 reference_date and the date parameter are expected.
reference_date	The start date column, e.g., date of first treatment A date or date-time object column is expected. 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. 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

Examples

```
data <- tibble::tribble(
  ~TRTSDT, ~AENDT,
  lubridate::ymd("2020-01-01"), lubridate::ymd("2020-02-24")
)

derive_var_aendy(data)
```

`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(dm)

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(dm)

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 paramater 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_anrind *Derive Reference Range Indicator*

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(vs)

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

```
vs %>%
  mutate(
    PARAMCD = VSTESTCD,
    AVAL = VSSTRESN
  ) %>%
  filter(PARAMCD %in% c("PULSE", "DIABP")) %>%
  left_join(ref_ranges, by = "PARAMCD") %>%
  derive_var_anrind() %>%
  select(USUBJID, PARAMCD, AVAL, ANRLO:ANRIND)
```

`derive_var_astdy` *Derive Analysis Start Relative Day*

Description

[Questioning]

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 `reference_date` and the `date` parameter are expected.
`reference_date`	The start date column, e.g., date of first treatment A date or date-time object column is expected. 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. 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

Examples

```
data <- tibble::tribble(
  ~TRTSDT, ~ASTDT,
  lubridate::ymd("2020-01-01"), lubridate::ymd("2020-02-24")
)

derive_var_astdy(data)
```

`derive_var_atirel` *Derive Time Relative to Reference*

Description

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

Examples

```
library(dplyr, warn.conflicts = FALSE)
adcm <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~TRTSDTM, ~ASTDTM, ~AENDTM, ~ASTTMF,
 "TEST01", "PAT01", "2012-02-25 23:00:00", "2012-02-28 19:00:00", "2012-02-25 23:00:00", "",
  "TEST01", "PAT01", "", "2012-02-28 19:00:00", "", "",
 "TEST01", "PAT01", "2017-02-25 23:00:00", "2013-02-25 19:00:00", "2014-02-25 19:00:00", "",
 "TEST01", "PAT01", "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-03-25 23:00:00", "m",
 "TEST01", "PAT01", "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-04-29 14:00:00", ""
) %>% dplyr::mutate(
  TRTSDTM = lubridate::as_datetime(TRTSDTM),
  ASTDTM = lubridate::as_datetime(ASTDTM),
  AENDTM = lubridate::as_datetime(AENDTM)
)

derive_var_atirel(
  dataset = adcm,
  flag_var = ASTTMF,
  new_var = ATIREL
)
```

---

derive_var_base *Derive Baseline Variables*

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",  9.7,   NA,       "Day 7",    "N",
  "TEST01", "PAT01",  "PARAM01", 15.01,  NA,       "Day 14",   "N",
  "TEST01", "PAT01",  "PARAM02",  8.35,  NA,       "Baseline", "Y",
  "TEST01", "PAT01",  "PARAM02", NA,     NA,       "Day 7",    "N",
  "TEST01", "PAT01",  "PARAM02",  8.35,  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_basec` *Derive BASEC Variable*

Description

[Deprecated]

This function is *deprecated*. Please use `derive_var_base()` instead.

Usage

```
derive_var_basec(dataset, by_vars)
```

Arguments

`dataset`	The input dataset
`by_vars`	Grouping variables uniquely identifying a set of records for which to calculate `new_var`

Value

The input dataset with variable `BASEC` added

Author(s)

Thomas Neitmann

See Also

`derive_var_base()`

`derive_var_basetype` *Derive BASETYPE Variable*

Description

Adds the BASETYPE variable to a dataset and duplicates records based upon the provided conditions

Usage

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

Arguments

dataset	Input dataset The columns specified in the expressions inside basetypes are required.
basetypes	A *named* list of expressions created using the exprs function The names corresponds to the values of the newly created BASETYPE variables and the expressions are used to subset the input dataset.

Details

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,
  "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 = 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"
  )
)
print(bds_with_basetype)

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

`derive_var_chg` *Derive Change from Baseline*

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

[Questioning]

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: • `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
: 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 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.

  Default is NULL

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

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 date (new_var) added

Author(s)

Samia Kabi

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("dm")
data("ds")

dm %>%
  derive_var_disposition_dt(
    dataset_ds = ds,
    new_var = FRVDT,
    dtc = DSSTDTC,
    filter_ds = DSCAT == "OTHER EVENT" & DSDECOD == "FINAL RETRIEVAL VISIT"
  ) %>%
  select(STUDYID, USUBJID, FRVDT)
```

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: • 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: `format_eoxxstt_default <- function(x) { case_when( x == "COMPLETED" ~ "COMPLETED", x != "COMPLETED" & !is.na(x) ~ "DISCONTINUED", TRUE ~ "ONGOING" ) }` where x is the status_var.

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("dm")
data("ds")

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

dm %>%
  derive_var_disposition_status(
    dataset_ds = 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"
  )
}

dm %>%
  derive_var_disposition_status(
    dataset_ds = 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"
)
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"
)

# Derive `DTHCAUS` 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
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_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 = NULL,
  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	Filter for flag data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. Permitted Values: a condition
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.

Value

The input dataset with the new flag variable added

Author(s)

Stefan Bundfuss

Examples

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

# Flag last value for each patient, test, and visit, baseline observations are ignored
vs %>%
  derive_var_extreme_flag(
    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
derive_var_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(ADT),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = High
derive_var_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(AVAL, ADT),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Worst observation - Direction = Lo
```

```
derive_var_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(desc(AVAL), ADT),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE"
)

# Average observation
derive_var_extreme_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD),
  order = vars(ADT, desc(AVAL)),
  new_var = ABLFL,
  mode = "last",
  filter = AVISIT == "BASELINE" & DTYPE == "AVERAGE"
)
```

`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 `by_vars` and `analysis_date` parameters are expected.

dataset_ex	Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars 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 dplyr::vars).
dose_id	Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars().
dose_date	The EX dose date variable.
analysis_date	The analysis date variable.
single_dose_condition	The condition for checking if dataset_ex is single dose. An error is issued if the condition is not true. Defaults to (EXDOSFRQ == "ONCE").
new_var	The new variable added to dataset.
dose_var	The EX source dose amount variable. Defaults to EXDOSE.
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 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.

Value

Input dataset with additional column new_var.

Author(s)

Annie Yang

See Also

derive_vars_last_dose()

Examples

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

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
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 by_vars and analysis_date parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars 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 dplyr::vars).
dose_id	Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars().
dose_date	The EX dose date variable.
analysis_date	The analysis date variable.
single_dose_condition	The condition for checking if dataset_ex is single dose. An error is issued if the condition is not true. Defaults to (EXDOSFRQ == "ONCE").
new_var	The new date or datetime variable added to dataset.
output_datetime	Display new_var 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.

Value

Input dataset with additional column new_var.

Author(s)

Ben Straub

See Also

derive_vars_last_dose()

Examples

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

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 by_vars and analysis_date parameters are expected.
dataset_ex	Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars 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 dplyr::vars).
dose_id	Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars().
dose_date	The EX dose date variable.
analysis_date	The analysis date variable.
single_dose_condition	The condition for checking if dataset_ex is single dose. An error is issued if the condition is not true. Defaults to (EXDOSFRQ == "ONCE").
new_var	The output variable defined by the user.
grp_brks	User supplied breaks to apply to groups. Refer to breaks parameter in cut() for details.
grp_lbls	User supplied labels to apply to groups. Refer to labels parameter in cut() for details.
include_lowest	logical, indicating if a value equal to the lowest (or highest, for right = FALSE) 'breaks' value should be included. Refer to include.lowest parameter in cut() for details.
right	Logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. Refer to right parameter in cut() for details.
dose_var	The source dose amount variable. Defaults to EXDOSE.
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

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.

Value

Input dataset with additional column new_var.

Author(s)

Ben Straub

See Also

derive_vars_last_dose(), cut()

Examples

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

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

[Questioning]

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 subject_keys are required.
...	Source(s) of known alive dates. One or more lstalvdt_source() 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.

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 last observation (with respect to date) is selected.
2. The LSTALVDT 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 specified by the date_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 last observation (with respect to the LSTALVDT variable) from the single dataset is selected and the new variable is merged to the input dataset.

Value

The input dataset with the LSTALVDT variable added.

Author(s)

Stefan Bundfuss, Thomas Neitmann

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("dm")
data("ae")
data("lb")
data("adsl")

ae_start <- lstalvdt_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first"
)
ae_end <- lstalvdt_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first"
)
lb_date <- lstalvdt_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10
)
adsl_date <- lstalvdt_source(dataset_name = "adsl", date = TRTEDT)

dm %>%
  derive_var_lstalvdt(
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(adsl = adsl, ae = ae, lb = lb)
  ) %>%
  select(USUBJID, LSTALVDT)

# derive last alive date and traceability variables
ae_start <- lstalvdt_source(
  dataset_name = "ae",
  date = AESTDTC,
  date_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- lstalvdt_source(
  dataset_name = "ae",
  date = AEENDTC,
  date_imputation = "first",
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
```

```
  )
)
lb_date <- lstalvdt_source(
  dataset_name = "lb",
  date = LBDTC,
  filter = nchar(LBDTC) >= 10,
  traceability_vars = vars(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- lstalvdt_source(
  dataset_name = "adsl",
  date = TRTEDT,
  traceability_vars = vars(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDTM"
  )
)

dm %>%
  derive_var_lstalvdt(
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(adsl = adsl, ae = ae, lb = lb)
  ) %>%
  select(USUBJID, LSTALVDT, LALVDOM, LALVSEQ, LALVVAR)
```

---

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("vs")

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
end_date	The end date of assessment/event (e.g. AENDT) A date or date-time object column is expected. 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.
ref_end_date	The upper bound of the on-treatment period A date or date-time object column is expected. 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_dateand 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_dateis 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_dateand was ongoing or ended after theref_end_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(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,                 ~TPT,
  "P01", ymd("2020-01-02T12:00"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"), NA,
  "P02", ymd("2020-01-01"),     ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"), "PRE",
  "P03", ymd("2019-12-31"),     ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"), 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` *Derive Percent Change from Baseline*

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_trtdurd` *Derive Total Treatment Duration (Days)*

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. Default: `TRTSDT`
`end_date`	The end date A date or date-time object is expected. 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

[Questioning]

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 by_vars parameter are expected.
dataset_ex	ex dataset The variables EXENDTC, EXSEQ, and those specified by the filter_ex 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: EXDOSE > 0 | (EXDOSE == 0 & str_detect(EXTRT, 'PLACEBO') 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

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

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("ex")
data("dm")

dm %>%
  derive_var_trtedtm(dataset_ex = ex) %>%
  select(USUBJID, TRTEDTM)
```

---

derive_var_trtsdtm *Derive Datetime of First Exposure to Treatment*

Description

[Questioning]

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 by_vars parameter are expected.
dataset_ex	ex dataset The variables EXSTDTC, EXSEQ, and those specified by the filter_ex 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: EXDOSE > 0 | (EXDOSE == 0 & str_detect(EXTRT, 'PLACEBO') 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

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

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(admiral.test)
data("ex")
data("dm")

dm %>%
  derive_var_trtsdtm(dataset_ex = ex) %>%
  select(USUBJID, TRTSDTM)
```

---

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 = NULL,
  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	Filter for flag data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. Permitted Values: a condition
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.

Value

The input dataset with the new flag variable added.

Author(s)

Ondrej Slama

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
derive_var_worst_flag(
  advs,
  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)
```

`derive_worst_flag` *Adds a Variable Flagging the Maximal / Minimal Value Within a Group of Observations*

Description

[Deprecated]

Deprecated, please use `derive_var_worst_flag()` instead.

Usage

```
derive_worst_flag(
  dataset,
  by_vars,
  order,
  new_var,
  param_var,
  analysis_var,
  worst_high,
  worst_low,
  filter = NULL,
```

```
  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	Filter for flag data Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered. Permitted Values: a condition
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.

Value

The input dataset with the new flag variable added.

Author(s)

Ondrej Slama

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_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
derive_worst_flag(
  advs,
  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)
```

---

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,
  dataset = deprecated()
)
```

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.
dataset	Deprecated, please use dataset_name instead.

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 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.
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 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

`base`	Input dataset
`compare`	Comparison dataset
`keys`	`character` vector of variables that define a unique row in the `base` and `compare` datasets
`...`	Additional arguments passed onto `diffdf::diffdf()`

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 occuring 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

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(adsl)

# Duplicate the first record
adsl <- rbind(adsl[1L, ], 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)".

Usage

```
extract_unit(x)
```

Arguments

x	A parameter description

Value

A string

Examples

```
extract_unit("Height (cm)")

extract_unit("Diastolic Blood Pressure (mmHg)")
```

ex_single *Single Dose Exposure Dataset*

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 *Select the First or Last Date from Several Sources*

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

sources	Sources A list of `tte_source()` objects is 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, 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

---

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 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
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. Permitted Values: "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. 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 the first or last observation of each by group

Author(s)

Stefan Bundfuss

Examples

```
library(dplyr, warn.conflict = FALSE)
library(admiral.test)
data("ex")

# Select first dose for each patient
ex %>%
  filter_extreme(
    by_vars = vars(USUBJID),
    order = vars(EXSEQ),
    mode = "first"
  ) %>%
```

```
  select(USUBJID, EXSEQ)

# Select highest dose for each patient
ex %>%
  filter_extreme(
    by_vars = vars(USUBJID),
    order = vars(EXDOSE),
    mode = "last",
    check_type = "none"
  ) %>%
  select(USUBJID, 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(vs)

filter_if(vs, rlang::quo(NULL))
filter_if(vs, rlang::quo(VSTESTCD == "Weight"))
```

format_eoxxstt_default *Default Format for Disposition Status*

Description

Define a function to map the disposition status.

Usage

```
format_eoxxstt_default(x)
```

Arguments

x	the disposition variable used for the mapping (e.g. DSDECOD).

Value

A character vector derived based on the values given in x: "COMPLETED" if x is "COMPLETED", "DISCONTINUED" if x is not "COMPLETED" nor NA, "ONGOING" otherwise.

Author(s)

Samia Kabi

format_reason_default *Default Format for the Disposition Reason*

Description

Define a function to map the 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 not 'COMPLETED' nor NA.

For example:

```
DCSREAS = format_reason_default(DSDECOD)
DCSREASP = format_reason_default(DSDECOD, DSTERM)
```

Value

A character vector

Author(s)

Samia Kabi

---

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(adsl)

# Duplicate the first record
adsl <- rbind(adsl[1L, ], adsl)

signal_duplicate_records(adsl, 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(adsl)

try(
  assert_one_to_one(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(adsl)

try(
  assert_one_to_one(adsl, vars(STUDYID), vars(SITEID))
)

get_one_to_many_dataset()
```

impute_dtc *Impute Partial Date(-time) Portion of a '--DTC' Variable*

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. 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.

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

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"
)
```

`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

Create an `lstalvdt_source` object

Usage

```
lstalvdt_source(
  dataset_name,
  filter = NULL,
  date,
  date_imputation = NULL,
  preserve = FALSE,
  traceability_vars = NULL,
  dataset = deprecated()
)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the last known alive date.
filter	An unquoted condition for filtering dataset.
date	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.
date_imputation	A string defining the date imputation for date. See date_imputation parameter of impute_dtc() for valid values.
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
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, or NA.
dataset	Deprecated, please use dataset_name instead.

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 to be used in an iteration of `call_derivation()`

Usage

```
params(...)
```

Arguments

`...`	One or more named arguments

Value

An object of class `params`

Author(s)

Thomas Neitmann

Examples

```
params(
  fns = list(VSSTRESN ~ mean(., na.rm = TRUE)),
  set_values_to = vars(DTYPE = "AVERAGE")
)
```

print.tte_source *Print* tte_source *Objects*

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 *Queries Dataset*

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.

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(adsl)

# Duplicate the first record
adsl <- rbind(adsl[1L, ], adsl)

signal_duplicate_records(adsl, vars(USUBJID), cnd_type = "message")
```

suppress_warning *Suppress Specific Warnings*

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(adsl)
data(vs)

# Remove label
attr(vs$USUBJID, "label") <- NULL

left_join(adsl, vs, by = "USUBJID")

suppress_warning(
  left_join(adsl, vs, by = "USUBJID"),
  "^Column `USUBJID` has different attributes on LHS and RHS of join$"
)
```

tte_source *Create a* tte_source *Object*

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.
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 *Open an ADaM Template Script*

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.
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.

Value

No return values, called for side effects

Author(s)

Shimeng Huang, Thomas Neitmann

Examples

```
if (interactive()) {
  use_ad_template("adsl")
}
```

vars2chr *Turn a List of Quosures into a Character Vector*

Description

Turn a List of Quosures into a Character Vector

Usage

```
vars2chr(quosures)
```

Arguments

quosures	A list of quosures created using vars()

Value

A character vector

Author(s)

Thomas Neitmann

Examples

```
vars2chr(vars(USUBJID, AVAL))
```

warn_if_inconsistent_list *Warn If Two Lists are Inconsistent*

Description

Checks if two list inputs have the same names and same number of elements and issues a warning otherwise.

Usage

```
warn_if_inconsistent_list(base, compare, list_name, i = 2)
```

Arguments

base	A named list
compare	A named list
list_name	A string the name of the list
i	the index id to compare the 2 lists

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 `dtc` 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` *Warn If a Variable Already Exists*

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(dm)

## No warning as `AAGE` doesn't exist in `dm`
warn_if_vars_exist(dm, "AAGE")

## Issues a warning
warn_if_vars_exist(dm, "ARM")
```

Index