Package ‘admiral’

October 18, 2023

Type Package

Title ADaM in R Asset Library

Version 0.12.3

Description A toolbox for programming Clinical Data Interchange Standards 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>).

Language en-US

License Apache License (>= 2)

BugReports https://github.com/pharmaverse/admiral/issues

URL https://pharmaverse.github.io/admiral/,
https://github.com/pharmaverse/admiral

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Depends R (>= 3.5)

Imports admiraldev (>= 0.4.0), dplyr (>= 0.8.4), hms (>= 0.5.3), lifecycle (>= 0.1.0), lubridate (>= 1.7.4), magrittr (>= 1.5), purrr (>= 0.3.3), rlang (>= 0.4.4), stringr (>= 1.4.0), tidy
(>= 1.0.2), tidyselect (>= 1.1.0)

Suggests pharmaversesdtm, DT, diffdf, knitr, rmarkdown, methods, readxl, testthat (>= 3.0.0), tibble

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no
Author Ben Straub [aut, cre],
    Stefan Bundfuss [aut],
    Jeffrey Dickinson [aut],
    Ross Farrugia [aut],
    Adam Forys [aut],
    Daphne Grasselly [aut],
    Dinakar Kulkarni [aut],
    Edoardo Mancini [aut],
    Sadchla Mascary [aut],
    Gordon Miller [aut],
    Sophie Shapcott [aut],
    Eric Simms [aut],
    Stefan Thoma [aut],
    Kangjie Zhang [aut],
    Zelos Zhu [aut],
    G Gayatri [ctb],
    Thomas Neitmann [ctb],
    F. Hoffmann-La Roche AG [cph, fnd],
    GlaxoSmithKline LLC [cph, fnd]

Maintainer Ben Straub <ben.x.straub@gsk.com>

Repository CRAN

Date/Publication 2023-10-18 20:50:11 UTC

R topics documented:

  admiral_adlb ................................................................. 5
  admiral_adsI ................................................................. 6
  assert_db_requirements ..................................................... 7
  assert_terms ................................................................. 8
  assert_valid_queries ........................................................ 9
  atoxgr_criteria_ctcv4 ...................................................... 10
  atoxgr_criteria_ctcv5 ...................................................... 11
  atoxgr_criteria_daids ...................................................... 12
  basket_select ............................................................... 13
  call_derivation ............................................................. 14
  call_user_fun ............................................................... 16
  censor_source ............................................................... 17
  chr2vars ..................................................................... 18
  compute_age_years .......................................................... 19
  compute_bmi ................................................................. 20
  compute_bsa ................................................................. 21
  compute_dtf ................................................................. 22
  compute_duration ........................................................... 23
  compute_egfr ................................................................. 25
  compute_framingham ........................................................ 28
  compute_map ................................................................. 30
  compute_qtc ................................................................. 32
topics documented:

- compute_qual_imputation ................................. 33
- compute_qual_imputation_dec                      34
- compute_rr .................................................. 35
- compute_scale ............................................. 36
- compute_tmf ................................................ 37
- consolidate_metadata ..................................... 38
- convert_blanks_to_na ..................................... 40
- convert_date_to_dtm ...................................... 41
- convert_dtc_to_dt ......................................... 44
- convert_dtc_to_dtm ........................................ 46
- convert_na_to_blanks ..................................... 48
- count_vals .................................................. 49
- create_period_dataset ..................................... 50
- create_query_data ......................................... 53
- create_single_dose_dataset ................................. 56
- date_source ................................................ 61
- death_event ............................................... 63
- default_qtc_paramcd ...................................... 64
- derivation_slice .......................................... 65
- derive_basetype_records .................................. 65
- derive_expected_records .................................. 67
- derive_extreme_event ...................................... 69
- derive_extreme_records .................................... 76
- derive_locf_records ....................................... 81
- derive_param_bmi ......................................... 84
- derive_param_bsa ......................................... 87
- derive_param_computed .................................... 90
- derive_param_doseint ..................................... 96
- derive_param_exist_flag .................................. 98
- derive_param_exposure ..................................... 102
- derive_param_extreme_event ............................... 104
- derive_param_extreme_record ......................... 107
- derive_param_framingham .................................. 109
- derive_param_map ......................................... 114
- derive_param_qtc .......................................... 117
- derive_param_rr ........................................... 119
- derive_param_tte .......................................... 121
- derive_param_wbc_abs ..................................... 127
- derive_summary_records ................................... 129
- derive_vars_aage .......................................... 133
- derive_vars_atc .......................................... 134
- derive_vars_dt ............................................. 136
- derive_vars_dtm ............................................ 140
- derive_vars_dtm_to_dt ...................................... 145
- derive_vars_dtm_to_tm .................................... 146
- derive_vars_duration ...................................... 147
- derive_vars_dy ............................................. 151
- derive_varsJoined ........................................ 152
topics documented:

derive_vars_last_dose ......................................................... 158
derive_vars_merged ........................................................... 160
derive_vars_merged_lookup .................................................. 165
derive_vars_period ............................................................. 169
derive_vars_query ............................................................. 171
derive_vars_transposed ....................................................... 173
derive_var_age_years ........................................................... 175
derive_var_analysis_ratio ...................................................... 176
derive_var_anrind ............................................................... 178
derive_var_atoxgr ............................................................... 179
derive_var_atoxgr_dir ........................................................... 181
derive_var_base ................................................................. 183
derive_var_basetype ............................................................ 185
derive_var_chg ................................................................. 186
derive_var_confirmation_flag ................................................. 187
derive_var_dthcaus ............................................................. 190
derive_var_extreme_dt .......................................................... 193
derive_var_extreme_dtm ......................................................... 198
derive_var_extreme_flag ....................................................... 202
derive_var_jointed_exist_flag ................................................. 207
derive_var_last_dose_amt ....................................................... 213
derive_var_last_dose_date ..................................................... 215
derive_var_last_dose_grp ....................................................... 216
derive_var_merged_cat .......................................................... 218
derive_var_merged_character .................................................. 220
derive_var_merged_exist_flag ................................................ 222
derive_var_merged_summary ................................................... 225
derive_var_obs_number .......................................................... 228
derive_var_ontrtfl ............................................................... 230
derive_var_pchg ................................................................. 233
derive_var_relative_flag ....................................................... 234
derive_var_shift ................................................................. 238
derive_var_trtdurd .............................................................. 239
derive_var_trtemfl .............................................................. 241
desc ................................................................. 244
dose_freq_lookup .............................................................. 244
dthcaus_source ................................................................. 245
event ................................................................. 246
event_jointed ................................................................. 248
event_source ................................................................. 250
example_qs ................................................................. 251
exprs ................................................................. 251
extend_source_datasets ........................................................ 252
extract_duplicate_records ................................................... 253
extract_unit ................................................................. 254
ex_single ................................................................. 254
filter_date_sources ............................................................ 255
filter_exist ................................................................. 257
Lab Analysis Dataset

An example of lab analysis dataset

Usage

admiral_adlb
Format

An object of class tbl_df (inherits from tbl, data.frame) with 3779 rows and 111 columns.

Source

Derived from the adlb template, then further filtered due to dataset size by the following USUB-JIDs: 01-701-1015, 01-701-1023, 01-701-1028, 01-701-1033, 01-701-1034, 01-701-1047, 01-701-1097, 01-705-1186, 01-705-1292, 01-705-1310, 01-708-1286

See Also

Other datasets: admiral_adsl, ex_single, example_qs, queries_mh, queries

Description

An example subject level analysis dataset

Usage

admiral_adsl

Format

An object of class tbl_df (inherits from tbl, data.frame) with 306 rows and 50 columns.

Source


See Also

Other datasets: admiral_adlb, ex_single, example_qs, queries_mh, queries
assert_db_requirements

Check required parameters for a basket

Description

If a basket (SMQ, SDG, ...) are requested, the version and a function to access the database must be provided. The function checks these requirements.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version provided by user</td>
</tr>
<tr>
<td>version_arg_name</td>
<td>Name of the argument providing the version</td>
</tr>
<tr>
<td>fun</td>
<td>Function provided by user</td>
</tr>
<tr>
<td>fun_arg_name</td>
<td>Name of the argument providing the function</td>
</tr>
<tr>
<td>queries</td>
<td>Queries provide by user</td>
</tr>
<tr>
<td>i</td>
<td>Index of query being checked</td>
</tr>
</tbody>
</table>

Value

An error is issued if version or fun is null.

See Also

Other Advanced Functions: assert_terms(), assert_valid_queries(), extend_source_datasets(), filter_date_sources(), format.basket_select(), list_tte_source_objects(), params(), validate_basket_select(), validate_query()
assert_terms

Asserts Requirements for Terms for Queries

Description

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

Usage

assert_terms(terms, expect_grpname = FALSE, expect_grpid = FALSE, source_text)

Arguments

terms Terms provided by user
expect_grpname Is the GRPNAME column expected?
expect_grpid Is the GRPID column expected?
source_text Text describing the source of the terms, e.g., “the data frame provided for the definition element”.

Value

An error is issued if

- terms is not a data frame,
- terms has zero observations,
- the SRCVAR variable is not in terms,
- neither the TERMNAME nor the TERMID variable is in terms,
- expect_grpname == TRUE and the GRPNAME variable is not in terms,
- expect_grpid == TRUE and the GRPID variable is not in terms.

See Also

create_query_data(), query()

Other Advanced Functions: assert_db_requirements(), assert_valid_queries(), extend_source_datasets(), filter_date_sources(), format.basket_select(), list_tte_source_objects(), params(), validate_basket_select(), validate_query()
assert_valid_queries

Examples

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

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

- PREFIX, e.g., SMQ01, CQ12
- GRPNAME, non NA, must be unique per each PREFIX
- GRPID, could be NA, must be unique per each PREFIX
- SCOPE, 'BROAD', 'NARROW', or NA
- SCOPEN, 1, 2, or NA
- SRCVAR, e.g., "AEDECOD", "AELLT", "AELLTC8", ...
- TERMNAME, character, could be NA only at those observations where TERMID is non-NA
- TERMID, integer, could be NA only at those observations where TERMNAME is non-NA

Value

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

See Also

Other Advanced Functions: assert_db_requirements(), assert_terms(), extend_source_datasets(), filter_date_sources(), format.basket_select(), list_tte_source_objects(), params(), validate_basket_select(), validate_query()
Examples

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

atxgr_criteria_ctcv4  Metadata Holding Grading Criteria for NCI-CTCAEv4

Description

Metadata Holding Grading Criteria for NCI-CTCAEv4

Usage

atxgr_criteria_ctcv4

Format

An object of class tbl_df (inherits from tbl, data.frame) with 40 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec Excel file which ships with \{admiral\} and can be accessed using system.file("adlb_grading/adlb_grading_spec.xlsx", package = "admiral") in sheet = "NCICTCAEv4". The dataset contained in there has the following columns:

- SOC: variable to hold the SOC of the lab test criteria.
- TERM: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- Grade 1: Criteria defining lab value as Grade 1.
- Grade 2: Criteria defining lab value as Grade 2.
- Grade 3: Criteria defining lab value as Grade 3.
- Grade 4: Criteria defining lab value as Grade 4.
- Grade 5: Criteria defining lab value as Grade 5.
- Definition: Holds the definition of the lab test abnormality.
- GRADE_CRITERIA_CODE: variable to hold code that creates grade based on defined criteria.
- SI_UNIT_CHECK: variable to hold unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- VAR_CHECK: List of variables required to implement lab grade criteria. Use to check against input data.
- DIRECTION: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables SOC, TERM, Grade 1, Grade 2, Grade 3, Grade 4, Grade 5, Definition are from the source document on NCI-CTC website defining the grading criteria. Common Terminology Criteria for Adverse Events (CTCAE)v4.0 From these variables only 'TERM' is used in the \{admiral\} code, the rest are for information and traceability only.
See Also

Other metadata: `atoxgr_criteria.ctcv5`, `atoxgr_criteria.daids`, `dose_freq_lookup`

---

### Metadata Holding Grading Criteria for NCI-CTCAEv5

**Description**

Metadata Holding Grading Criteria for NCI-CTCAEv5

**Usage**

`atoxgr_criteria.ctcv5`

**Format**

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 37 rows and 13 columns.

**Details**

This metadata has its origin in the ADLB Grading Spec Excel file which ships with `{admiral}` and can be accessed using `system.file("adlb_grading/adlb_grading_spec.xlsx", package = "admiral")` in sheet = "NCICTCAEv5". The dataset contained in there has the following columns:

- **SOC**: variable to hold the SOC of the lab test criteria.
- **TERM**: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- **Grade 1**: Criteria defining lab value as Grade 1.
- **Grade 2**: Criteria defining lab value as Grade 2.
- **Grade 3**: Criteria defining lab value as Grade 3.
- **Grade 4**: Criteria defining lab value as Grade 4.
- **Grade 5**: Criteria defining lab value as Grade 5.
- **Definition**: Holds the definition of the lab test abnormality.
- **GRADE CRITERIA CODE**: variable to hold code that creates grade based on defined criteria.
- **SI_UNIT_CHECK**: variable to hold unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- **VAR_CHECK**: List of variables required to implement lab grade criteria. Use to check against input data.
- **DIRECTION**: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- **COMMENT**: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`, `Grade 3`, `Grade 4`, `Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v5.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.
**atoxgr_criteria_daids**

**Metadata Holding Grading Criteria for DAIDs**

**Description**

Metadata Holding Grading Criteria for DAIDs

**Usage**

`atoxgr_criteria_daids`

**Format**

An object of class `tbl_df` (inherits from `tbl.data.frame`) with 63 rows and 15 columns.

**Details**

This metadata has its origin in the ADLB Grading Spec Excel file which ships with `{admiral}` and can be accessed using `system.file("adlb_grading/adlb_grading_spec.xlsx", package = "admiral")` in sheet = "DAIDS". The dataset contained in there has the following columns:

- **SOC**: variable to hold the SOC of the lab test criteria.
- **TERM**: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- **SUBGROUP**: Description of sub-group of subjects were grading will be applied (i.e. >= 18 years)
- **Grade 1**: Criteria defining lab value as Grade 1.
- **Grade 2**: Criteria defining lab value as Grade 2.
- **Grade 3**: Criteria defining lab value as Grade 3.
- **Grade 4**: Criteria defining lab value as Grade 4.
- **Grade 5**: Criteria defining lab value as Grade 5.
- **Definition**: Holds the definition of the lab test abnormality.
- **FILTER**: admiral code to apply the filter based on SUBGROUP column.
- **GRADE_CRITERIA_CODE**: variable to hold code that creates grade based on defined criteria.
- **SI_UNIT_CHECK**: variable to hold unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- **VAR_CHECK**: List of variables required to implement lab grade criteria. Use to check against input data.
- **DIRECTION**: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.

See Also

Other metadata: `atoxgr_criteria_cctv4, atoxgr_criteria_daids, dose_freq_lookup`
• **COMMENT**: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables SOC, TERM, SUBGROUP, Grade 1, Grade 2, Grade 3, Grade 4, Grade 5. Definition are from the source document on DAIDS website defining the grading criteria. [Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events] From these variables only 'TERM' is used in the {admiral} code, the rest are for information and traceability only.

**See Also**

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv5`, `dose_freq_lookup`

---

**basket_select**

*Create a basket_select object*

**Description**

Create a `basket_select` object

**Usage**

```
basket_select(name = NULL, id = NULL, scope = NULL, type)
```

**Arguments**

- `name` Name of the query used to select the definition of the query from the company database.
- `id` Identifier of the query used to select the definition of the query from the company database.
- `scope` Scope of the query used to select the definition of the query from the company database.
  
  *Permitted Values*: "BROAD", "NARROW", NA_character_
- `type` The type argument expects a character scalar. It is passed to the company specific `get_terms()` function such that the function can determine which sort of basket is requested

**Details**

Exactly one of `name` or `id` must be specified.

**Value**

An object of class `basket_select`.

**See Also**

`create_query_data()`, `query()`
Call a Single Derivation Multiple Times

Description

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

Usage

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

Arguments

dataset The input dataset
derivation The derivation function to call
A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. The function must provide the dataset argument and all arguments specified in the params() objects passed to the variable_params and ... argument.

Please note that it is not possible to specify {dplyr} functions like mutate() or summarize().

variable_params A list of function arguments that are different across iterations. Each set of function arguments must be created using params().

... Any number of named function arguments that stay the same across iterations. If a function argument 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.

See Also

params()

Higher Order Functions: derivation_slice(), restrict_derivation(), slice_derivation()

Examples

library(dplyr, warn.conflicts = FALSE)
ads1 <- tribble(~STUDYID, ~USUBJID, ~TRTSDT, ~TRTEDT, ~PILOT01", ~01-1307", NA, NA,
call_derivation

"PILOT01", "06-1384", "2012-09-15", "2012-09-24",
"PILOT01", "15-1085", "2013-02-16", "2013-08-18",
"PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
mutate(
  across(TRTSDT:TRTEDT, as.Date)
)

ae <- tribble(~STUDYID, ~DOMAIN, ~USUBJID, ~AESTDTC, ~AEENDTC,
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01", "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01", "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01", "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01", "AE", "16-1298", "2013-04-22", "2013-07-06"
)

adae <- ae %>%
derive_vars_merged(
  dataset_add = adsl,
  new_vars = exprs(TRTSDT, TRTEDT),
  by_vars = exprs(USUBJID)
)

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

## The above call using `call_derivation()` is equivalent to the following
adae %>%
derive_vars.dt(
  new_vars_prefix = "AST",

```r
dtc = AESTDTC,
date_imputation = "first",
min_dates = exprs(TRTSDT),
max_dates = exprs(TRTEDT)
)
%
derive_vars_dt(
    new_vars_prefix = "AEN",
dtc = AEENDTC,
date_imputation = "last",
min_dates = exprs(TRTSDT),
max_dates = exprs(TRTEDT)
)
```

---

**call_user_fun**

*Calls a Function Provided by the User*

**Description**

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

**Usage**

```r
call_user_fun(call)
```

**Arguments**

- `call` Call to be executed

**Value**

The return value of the function call

**See Also**

Utilities used within Derivation functions: `extract_unit()`, `get_not_mapped()`, `signal_duplicate_records()`

**Examples**

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

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

Create a censor_source Object

Description

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

Note: This is a wrapper function for the more generic tte_source().

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 or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object.

censor  Censoring value
CDISC strongly recommends using 0 for events and positive integers for censoring.

set_values_to  A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.

Value

An object of class censor_source, inheriting from class tte_source

See Also

derive_param_tte(), event_source()

Source Objects: basket_select(), date_source(), death_event, dthcaus_source(), event_joined(), event_source(), event(), query(), records_source(), tte_source()
Examples

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

chr2vars(chr)

Arguments

chr A character vector

Value

A list of expressions as returned by exprs()

See Also

Other utils_quo: negate_vars()

Examples

chr2vars(c("USUBJID", "AVAL"))
**compute_age_years**

**Compute Age in Years**

**Description**

Converts a set of age values from the specified time unit to years.

**Usage**

`compute_age_years(age, age_unit)`

**Arguments**

- **age**
  - The ages to convert.
  - A numeric vector is expected.

- **age_unit**
  - Age unit.
  - Either a string containing the time unit of all ages in `age` or a character vector containing the time units of each age in `age` is expected. Note that permitted values are cases insensitive (e.g. "YEARS" is treated the same as "years" and "Years").
  - Permitted Values: "years", "months", "weeks", "days", "hours", "minutes", "seconds", NA_character_.

**Details**

Returns a numeric vector of ages in years as doubles. Note that passing NA_character_ as a unit will result in an NA value for the outputted age. Also note, underlying computations assume an equal number of days in each year (365.25).

**Value**

The ages contained in `age` converted to years.

**See Also**

Date/Time Computation Functions that returns a vector: `compute_dtf()`, `compute_duration()`, `compute_tmf()`, `convert_date_to_dtm()`, `convert_dtc_to_dtm()`, `convert_dtc_to_dt()`, `impute_dtc_dtm()`, `impute_dtc_dt()`

**Examples**

```r
compute_age_years(
  age = c(240, 360, 480),
  age_unit = "MONTHS"
)
```

```r
compute_age_years(
  age = c(10, 520, 3650, 1000),
)```
compute_bmi

age_unit = c("YEARS", "WEEKS", "DAYS", NA_character_)

---

**compute_bmi**  
Compute Body Mass Index (BMI)

**Description**
Computes BMI from height and weight

**Usage**
```r
compute_bmi(height, weight)
```

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

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

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

**See Also**
- `derive_param_bmi()`
- BDS-Findings Functions that returns a vector: `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_map()`, `compute_qtc()`, `compute_qual_imputation_dec()`, `compute_qual_imputation()`, `compute_rr()`, `compute_scale()`

**Examples**
```r
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

Details

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

Value

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

See Also

derive_param_bsa()
BDS-Findings Functions that returns a vector: compute_bmi(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr(), compute_scale()
Examples

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

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

----------

compute_dtf  Derive the Date Imputation Flag

Description

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

Usage

compute_dtf(dtc, dt)

Arguments

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

Details

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

Value

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

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_duration(),
compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm(),
impute_dtc_dt()
**compute_duration**

**Examples**

```r
compute_dtf(dtc = "2019-07", dt = as.Date("2019-07-18"))
compute_dtf(dtc = "2019", dt = as.Date("2019-07-18"))
compute_dtf(dtc = "06-01T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022-06--T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022---01T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022----T00:00", dt = as.Date("2022-06-01"))
```

**compute_duration**  
*Compute Duration*

**Description**

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

**Usage**

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

**Arguments**

- `start_date`: The start date  
  A date or date-time object is expected.  
  Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.  
  Refer to `convert_dtc_to_dt()` to obtain a vector of imputed dates.

- `end_date`: The end date  
  A date or date-time object is expected.  
  Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.  
  Refer to `convert_dtc_to_dt()` to obtain a vector of imputed dates.

- `in_unit`: Input unit  
  See `floor_in` and `add_one` parameter for details.  
  Default: 'days'  
  Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'min', 'seconds', 'sec'
The duration is derived in the specified unit
Default: 'days'
Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'min', 'seconds', 'sec'

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 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

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

Lubridate duration type.
See below for details.
Default: "duration"
Permitted Values: "duration", "interval"

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

Value
The duration between the two date in the specified unit

Duration Type
The lubridate package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The interval between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The duration between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the lubridate time span reference page.
compute_egfr

Compute Estimated Glomerular Filtration Rate (eGFR) for Kidney Function

See Also
derive_vars_duration()

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm(), impute_dtc_dt()

Examples

library(lubridate)

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

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

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

# Derive duration in hours
compute_duration(
  start_date = ymd_hms("2020-12-06T9:00:00"),
  end_date = ymd_hms("2020-12-06T13:30:00"),
  out_unit = "hours",
  floor_in = FALSE,
  add_one = FALSE
)
Description

Compute Kidney Function Tests:

- Estimated Creatinine Clearance (CRCL) by Cockcroft-Gault equation
- Estimated Glomerular Filtration Rate (eGFR) by CKD-EPI or MDRD equations

Usage

compute_egfr(creat, creatu = "SI", age, weight, sex, race = NULL, method, wt)

Arguments

creat Creatinine
  A numeric vector is expected.
creatu Creatinine Units
  A character vector is expected.
    Default: "SI"
    Expected Values: "SI", "CV", "umol/L", "mg/dL"

age Age (years)
  A numeric vector is expected.

weight Weight (kg)
  A numeric vector is expected if method = "CRCL"

sex Gender
  A character vector is expected.
  Expected Values: "M", "F"

race Race
  A character vector is expected if method = "MDRD"
  Expected Values: "BLACK OR AFRICAN AMERICAN" and others

method Method
  A character vector is expected.
  Expected Values: "CRCL", "CKD-EPI", "MDRD"

wt Deprecated, please use weight instead.

Details

Calculates an estimate of Glomerular Filtration Rate (eGFR)

**CRCL Creatinine Clearance (Cockcroft-Gault)**

For Creatinine in umol/L:

\[
\frac{(140 - \text{age}) \times \text{weight(kg)} \times \text{constant}}{\text{Serum Creatinine(µmol/L)}}
\]

*Constant = 1.04 for females, 1.23 for males*
compute_egfr

For Creatinine in mg/dL:

\[
\frac{(140 - age) \times weight(kg) \times (0.85 \text{ if female})}{72 \times \text{Serum Creatinine(mg/dL)}}
\]

units = mL/min

CKD-EPI Chronic Kidney Disease Epidemiology Collaboration formula

\[
eGFR = 142 \times \min(SCr/\kappa,1)^\alpha \times \max(SCr/\kappa,1)^{-1.209} \times 0.9938^{Age} \times 1.012[\text{if female}]
\]

SCr = standardized serum creatinine in mg/dL (Note SCr(mg/dL) = Creat(umol/L) / 88.42)

\(\kappa\) = 0.7 (females) or 0.9 (males)

\(\alpha\) = -0.241 (female) or -0.302 (male) units = mL/min/1.73 m²

MDRD Modification of Diet in Renal Disease formula

\[
eGFR = 175 \times (SCr)^{-1.154} \times (age)^{-0.203} \times 0.742[\text{if female}] \times 1.212[\text{if Black}]
\]

SCr = standardized serum creatinine in mg/dL (Note SCr(mg/dL) = Creat(umol/L) / 88.42)

units = mL/min/1.73 m²

Value

A numeric vector of egfr values

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr(), compute_scale()

Examples

```r
compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, weight = 85, sex = "M", method = "CRCL"
)

compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, sex = "M", race = "ASIAN", method = "MDRD"
)

compute_egfr(
  creat = 70, creatu = "umol/L", age = 52, sex = "F", race = "BLACK OR AFRICAN AMERICAN", method = "MDRD"
)
```
compute_framingham

Description

Computes Framingham Heart Study Cardiovascular Disease 10-Year Risk Score (FCVD101) based on systolic blood pressure, total serum cholesterol (mg/dL), HDL serum cholesterol (mg/dL), sex, smoking status, diabetic status, and treated for hypertension flag.

Usage

compute_framingham(sysbp, chol, cholhdl, age, sex, smokefl, diabetfl, trthypfl)

Arguments

sysbp  Systolic blood pressure
       A numeric vector is expected.

chol   Total serum cholesterol (mg/dL)
       A numeric vector is expected.
**compute_framingham**

`cholhdl` HDL serum cholesterol (mg/dL)  
A numeric vector is expected.

`age` Age (years)  
A numeric vector is expected.

`sex` Gender  
A character vector is expected. Expected Values: 'M' 'F'

`smokefl` Smoking Status  
A character vector is expected. Expected Values: 'Y' 'N'

`diabetfl` Diabetic Status  
A character vector is expected. Expected Values: 'Y' 'N'

`trthypfl` Treated for hypertension status  
A character vector is expected. Expected Values: 'Y' 'N'

**Details**

The predicted probability of having cardiovascular disease (CVD) within 10-years according to Framingham formula. See AHA Journal article General Cardiovascular Risk Profile for Use in Primary Care for reference.

**For Women:**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>2.32888</td>
</tr>
<tr>
<td>Total Chol</td>
<td>1.20904</td>
</tr>
<tr>
<td>HDL Chol</td>
<td>-0.70833</td>
</tr>
<tr>
<td>Sys BP</td>
<td>2.76157</td>
</tr>
<tr>
<td>Sys BP + Hypertension Meds</td>
<td>2.82263</td>
</tr>
<tr>
<td>Smoker</td>
<td>0.52873</td>
</tr>
<tr>
<td>Non-Smoker</td>
<td>0</td>
</tr>
<tr>
<td>Diabetic</td>
<td>0.69154</td>
</tr>
<tr>
<td>Not Diabetic</td>
<td>0</td>
</tr>
<tr>
<td>Average Risk</td>
<td>26.1931</td>
</tr>
<tr>
<td>Risk Period</td>
<td>0.95012</td>
</tr>
</tbody>
</table>

**For Men:**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>3.06117</td>
</tr>
<tr>
<td>Total Chol</td>
<td>1.12370</td>
</tr>
<tr>
<td>HDL Chol</td>
<td>-0.93263</td>
</tr>
<tr>
<td>Sys BP</td>
<td>1.93303</td>
</tr>
<tr>
<td>Sys BP + Hypertension Meds</td>
<td>2.99881</td>
</tr>
<tr>
<td>Smoker</td>
<td>0.65451</td>
</tr>
<tr>
<td>Non-Smoker</td>
<td>0</td>
</tr>
<tr>
<td>Diabetic</td>
<td>0.57367</td>
</tr>
<tr>
<td>Not Diabetic</td>
<td>0</td>
</tr>
</tbody>
</table>
The equation for calculating risk:

\[
RiskFactors = (\log(Age) \times AgeFactor) + (\log(TotalChol) \times TotalCholFactor) + (\log(CholHDL) \times CholHDLFactor) + (\log(SysBP) \times SysBPFactor) + Smoker + DiabetesPresent - AvgRisk
\]

\[
Risk = 100 \times (1 - RiskPeriodFactor \times e^{xp(RiskFactors)})
\]

Value

A numeric vector of Framingham values

See Also

derive_param_framingham()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr(), compute_scale()

Examples

compute_framingham(
    sysbp = 133, chol = 216.16, cholhdl = 54.91, age = 53,
    sex = "M", smokefl = "N", diabetfl = "N", trthypfl = "N"
)

compute_framingham(
    sysbp = 161, chol = 186.39, cholhdl = 64.19, age = 52,
    sex = "F", smokefl = "Y", diabetfl = "N", trthypfl = "Y"
)

---

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)
compute_map

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

\[
\frac{2\text{DIABP} + \text{SYSBP}}{3}
\]

if it is based on diastolic and systolic blood pressure and

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

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

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

Value

A numeric vector of MAP values

See Also

derive_param_map()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_qtc(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr(), compute_scale()

Examples

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

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

Compute Corrected QT

Description

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

Usage

compute_qtc(qt, rr, method)

Arguments

qt     QT interval
       A numeric vector is expected. It is expected that QT is measured in msec.

rr     RR interval
       A numeric vector is expected. It is expected that RR is measured in msec.

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

Details

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

**Bazett:**

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

**Fridericia:**

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

**Sagie:**

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

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

Value

QT interval in msec

See Also

derive_param_qtc()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr(), compute_scale()
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_qual_imputation

Function to Impute Values When Qualifier Exists in Character Result

Description

Derive an imputed value

Usage

compute_qual_imputation(character_value, imputation_type = 1, factor = 0)

Arguments

character_value
Character version of value to be imputed

imputation_type
(default value=1) Valid Values: 1: Strip <, >, = and convert to numeric. 2: imputation_type=1 and if the character value contains a < or >, the number of decimals associated with the character value is found and then a factor of 1/10^(number of decimals + 1) will be added/subtracted from the numeric value. If no decimals exists, a factor of 1/10 will be added/subtracted from the value.

factor
Numeric value (default=0), when using imputation_type = 1, this value can be added or subtracted when the qualifier is removed.

Value

The imputed value

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_rr(), compute_scale()

Examples

compute_qual_imputation("<40")
compute_qual_imputation_dec

*Compute Factor for Value Imputations When Character Value Contains < or >*

**Description**

Function to compute factor for value imputation when character value contains < or >. The factor is calculated using the number of decimals. If there are no decimals, the factor is 1, otherwise the factor = $1/10^{\text{decimal place}}$. For example, the factor for 100 = 1, the factor for 5.4 = $1/10^1$, the factor for 5.44 = $1/10^2$. This results in no additional false precision added to the value. This is an intermediate function.

**Usage**

```r
compute_qual_imputation_dec(character_value_decimal)
```

**Arguments**

- `character_value_decimal`
  - Character value to determine decimal precision

**Details**

Derive an imputed value

**Value**

Decimal precision value to add or subtract

**See Also**

BDS-Findings Functions that returns a vector: `compute_bmi()`, `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_map()`, `compute_qtc()`, `compute_qual_imputation()`, `compute_rr()`, `compute_scale()`

**Examples**

```r
compute_qual_imputation_dec("<40.1")
```
**compute_rr**

*Compute RR Interval From Heart Rate*

**Description**

Computes RR interval from heart rate.

**Usage**

```r
compute_rr(hr)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hr</td>
<td>Heart rate</td>
</tr>
</tbody>
</table>

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

**Details**

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

**Value**

RR interval in msec:

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

**See Also**

`derive_param_rr()`

BDS-Findings Functions that returns a vector: `compute_bmi()`, `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_map()`, `compute_qtc()`, `compute_qual_imputation_dec()`, `compute_qual_imputation()`, `compute_scale()`

**Examples**

```r
compute_rr(hr = 70.14)
```
compute_scale  Compute Scale Parameters

**Description**

Computes the average of a set of source values and transforms the result from the source range to the target range. For example, for calculating the average of a set of questionnaire response scores and re-coding the average response to obtain a subscale score.

**Usage**

```r
compute_scale(
  source,
  source_range = NULL,
  target_range = NULL,
  flip_direction = FALSE,
  min_n = 1
)
```

**Arguments**

- `source`: A vector of values to be scaled
  - A numeric vector is expected.
- `source_range`: The permitted source range
  - A numeric vector containing two elements is expected, representing the lower and upper bounds of the permitted source range. Alternatively, if no argument is specified for `source_range` and `target_range`, no transformation will be performed.
- `target_range`: The target range
  - A numeric vector containing two elements is expected, representing the lower and upper bounds of the target range. Alternatively, if no argument is specified for `source_range` and `target_range`, no transformation will be performed.
- `flip_direction`: Flip direction of the scale?
  - The transformed values will be reversed within the target range, e.g. within the range 0 to 100, 25 would be reversed to 75.
  - This argument will be ignored if `source_range` and `target_range` aren’t specified.
  - Default: FALSE
  - Permitted Values: TRUE, FALSE
- `min_n`: Minimum number of values for computation
  - The minimum number of non-missing values in source for the computation to be carried out. If the number of non-missing values is below `min_n`, the result will be set to missing, i.e. NA.
  - A positive integer is expected.
  - Default: 1
compute_tmf

Details

Returns a numeric value. If source contains less than min_n values, the result is set to NA. If source_range and target_range aren’t specified, the mean will be computed without any transformation being performed.

Value

The average of source transformed to the target range or NA if source doesn’t contain min_n values.

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_qual_imputation(), compute_rr()

Examples

```r
compute_scale(
  source = c(1, 4, 3, 5),
  source_range = c(1, 5),
  target_range = c(0, 100),
  flip_direction = TRUE,
  min_n = 3
)
```

---

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

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

Arguments

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

ADaM IG states that given SDTM ('--DTC') variable, if only hours and minutes are ever collected, and seconds are imputed in ('--DTM') as 00, then it is not necessary to set ('--TMF') to 'S'. A user can set this to TRUE so the 'S' Flag is dropped from ('--TMF').
A logical value
Default: FALSE

Details

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

Value

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

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), convert_date_to_dtm(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm(), impute_dtc_dt()

Examples

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"))

consolidate_metadata

Consolidate Multiple Meta Datasets Into a Single One

Description

The purpose of the function is to consolidate multiple meta datasets into a single one. For example, from global and project specific parameter mappings a single lookup table can be created.

Usage

consolidate_metadata(
  datasets,
  key_vars,
  source_var = SOURCE,
  check_vars = "warning",
  check_keys = "error"
)
Arguments

datasets  List of datasets to consolidate

  *Permitted Values*: A named list of datasets

key_vars  Key variables

  The specified variables must be a unique of all input datasets.

  *Permitted Values*: A list of variables created by exprs()

source_var  Source variable

  The specified variable is added to the output dataset. It is set the name of the
dataset the observation is originating from.

  *Permitted Values*: A symbol

check_vars  Check variables?

  If "message", "warning", or "error" is specified, a message is issued if the
variable names differ across the input datasets (datasets).

  *Permitted Values*: "none", "message", "warning", "error"

check_keys  Check keys?

  If "warning" or "error" is specified, a message is issued if the key variables
(key_vars) are not a unique key in all of the input datasets (datasets).

  *Permitted Values*: "none", "warning", "error"

Details

All observations of the input datasets are put together into a single dataset. If a by group (defined
by key_vars) exists in more than one of the input datasets, the observation from the last dataset is
selected.

Value

A dataset which contains one row for each by group occurring in any of the input datasets.

See Also

Creating auxiliary datasets: `create_period_dataset()`, `create_query_data()`, `create_single_dose_dataset()`

Examples

```r
library(tibble)
glob_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI, 
  "PULSE", 60, 100,
  "SYSBP", 90, 130,
  "DIABP", 60, 80
)
proj_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI, 
  "PULSE", 60, 100,
  "SYSBP", 100, 140,
  "DIABP", 70, 90
)
```
stud_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI, 
  "BMI", 18, 25
)

consolidate_metadata(
  datasets = list(
    global = glob_ranges,
    project = proj_ranges,
    study = stud_ranges
  ),
  key_vars = exprs(PARAMCD)
)

---

**convert_blanks_to_na**  
*Convert Blank Strings Into NAs*

**Description**

Turn SAS blank strings into proper R NAs.

**Usage**

convert_blanks_to_na(x)

```r
## 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.
convert_date_to_dtm

Convert a Date into a Datetime Object

Convert a date (datetime, date, or date character) into a Date vector (usually `"DTM"`).

Note: This is a wrapper function for the function convert_dtc_to_dtm().

Usage

```r
convert_date_to_dtm(
  dt,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  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`.  

Value

An object of the same class as the input

See Also

Utilities for Formatting Observations: convert_na_to_blanks(), yn_to_numeric()

Examples

```r
library(tibble)

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

df <- tribble(
  ~USUBJID, ~RFICDTC,
  "1001", "2000-01-01",
  "1002", "2001-01-01",
  "1003", ""
)
print(df)
convert_blanks_to_na(df)
```
highest_imputation

Highest imputation level

The `highest_imputation` argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, `NA_character_` is returned. For example, for `highest_imputation = "D"` "2020" results in `NA_character_` because the month is missing.

If "n" is specified, no imputation is performed, i.e., if any component is missing, `NA_character_` is returned.

If "Y" is specified, `date_imputation` should be "first" or "last" and `min_dates` or `max_dates` should be specified respectively. Otherwise, `NA_character_` is returned if the year component is missing.

*Default: "h"

*Permitted Values: "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

date_imputation

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

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 (The year can not be specified; for imputing the year "first" or "last" together with `min_dates` or `max_dates` argument can be used (see examples)),
- or as a keyword: "first", "mid", "last" to impute to the first/mid/last day/month.

The argument is ignored if `highest_imputation` is less then "D".

*Default: "first".

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.

The argument is ignored if `highest_imputation = "n"`.

*Default: "first".

min_dates

Minimum dates

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

```r
impute_dtc_dtm(
    "2020-11",
```
convert_date_to_dtm

min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
),
    highest_imputation = "M"
)

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).

For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

max_dates

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

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

preserve

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

Details

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

Value

A datetime object

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm(), impute_dtc_dt()

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 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,  
  highest_imputation = "n",  
  date_imputation = "first",  
  min_dates = NULL,  
  max_dates = NULL,  
  preserve = FALSE
)
```

Arguments
- **dtc**: The DTC date to convert.
- **highest_imputation**: Highest imputation level
  The `highest_imputation` argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.
  - If a component at a higher level than the highest imputation level is missing, `NA_character_` is returned. For example, for `highest_imputation = "D"` "2020" results in `NA_character_` because the month is missing.
  - If "n" is specified no imputation is performed, i.e., if any component is missing, `NA_character_` is returned.
  - If "Y" is specified, `date_imputation` should be "first" or "last" and `min_dates` or `max_dates` should be specified respectively. Otherwise, `NA_character_` is returned if the year component is missing.
  - Default: "n"

- **Permitted Values**: "Y" (year, highest level), "M" (month), "D" (day), "n" (none, lowest level)

- **date_imputation**: The value to impute the day/month when a datepart is missing.
  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 (The year can not be specified; for imputing the year "first" or "last" together with `min_dates` or `max_dates` argument can be used (see examples)),
  - or as a keyword: "first", "mid", "last" to impute to the first/mid/last day/month.
convert_dtc_to_dt

The argument is ignored if highest_imputation is less than "D".

Default: "first"

**min_dates** Minimum dates

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

```r
impute_dtc_dtm(
  "2020-11",
  min_dates = list(  
    ymd_hms("2020-12-06T12:12:12"), 
    ymd_hms("2020-11-11T11:11:11")  
  ),
  highest_imputation = "M"
)
```

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

**max_dates** Maximum dates

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

**preserve** Preserve day if month is missing and day is present

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

Permitted Values: TRUE, FALSE

Default: FALSE

**Details**

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

**Value**

a date object

**See Also**

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dtm(), impute_dtc_dtm(), impute_dtc_dt()
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,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments
- **dtc**: The `"DTC"` date to convert.
- **highest_imputation**: Highest imputation level
  
  The `highest_imputation` argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.
  
  If a component at a higher level than the highest imputation level is missing, `NA_character_` is returned. For example, for `highest_imputation = "D"` "2020" results in `NA_character_` because the month is missing.
  
  If "n" is specified, no imputation is performed, i.e., if any component is missing, `NA_character_` is returned.
  
  If "Y" is specified, `date_imputation` should be "first" or "last" and `min_dates` or `max_dates` should be specified respectively. Otherwise, `NA_character_` is returned if the year component is missing.
  
  *Default*: "h"

  *Permitted Values*: "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

- **date_imputation**: The value to impute the day/month when a datepart is missing.
  
  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 (The year can not be specified; for imputing the year "first" or "last" together with min_dates or max_dates argument can be used (see examples)).
• or as a keyword: "first", "mid", "last" to impute to the first/mid/last day/month.

The argument is ignored if highest_imputation is less then "D".

Default: "first".

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.

The argument is ignored if highest_imputation = "n".

Default: "first".

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

```r
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  highest_imputation = "M"
)
```
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).

For date variables (not datetime) in the list the time is imputed to "00:00:00".
Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

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

For date variables (not datatime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.
convert_na_to_blanks

preserve

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

Details

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

Value

A datetime object

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(),
compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm(),
impute_dtc_dt()

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")

convert_na_to_blanks Convert NAs Into Blank Strings

Description

Turn NAs to blank strings.

Usage

convert_na_to_blanks(x)

## Default S3 method:
convert_na_to_blanks(x)

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

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

## S3 method for class 'data.frame'
convert_na_to_blanks(x)
count_vals

Arguments

x  Any R object

Details

The default methods simply returns its input unchanged. The character method turns every instance of NA_character_ or NA into "" 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 all attributes such as labels are preserved.

Value

An object of the same class as the input

See Also

Utilities for Formatting Observations: convert_blanks_to_na(), yn_to_numeric()

Examples

library(tibble)

convert_na_to_blanks(c("a", "b", NA, "d", NA))

df <- tribble(
  ~USUBJID, ~RFICDT
)

print(df)

convert_na_to_blanks(df)

---

count_vals  Count Number of Observations Where a Variable Equals a Value

Description

Count number of observations where a variable equals a value.

Usage

count_vals(var, val)

Arguments

var  A vector

val  A value
create_period_dataset

Create a Reference Dataset for Subperiods, Periods, or Phases

Description

The function creates a reference dataset for subperiods, periods, or phases from the ADSL dataset. The reference dataset can be used to derive subperiod, period, or phase variables like ASPER, ASPRSDT, ASPREDT, APERIOD, APERSDT, APEREDT, TRTA, APHASEN, PHSDTM, PHEDTM, ... in OCCDS and BDS datasets.

Usage

```r
create_period_dataset(
  dataset,
  new_vars,
  subject_keys = get_admiral_option("subject_keys")
)
```

See Also

Utilities for Filtering Observations:
- `filter_exist()`
- `filter_extreme()`
- `filter_joined()`
- `filter_not_exist()`
- `filter_relative()`
- `max_cond()`
- `min_cond()`

Examples

```r
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1", 1, "PR",
  "1", 2, "CR",
  "1", 3, "NE",
  "1", 4, "CR",
  "1", 5, "NE",
  "2", 1, "CR",
  "2", 2, "PR",
  "2", 3, "CR",
  "3", 1, "CR",
  "4", 1, "CR",
  "4", 2, "NE",
  "4", 3, "NE",
  "4", 4, "CR",
  "4", 5, "PR"
)

# add variable providing the number of NEs for each subject
group_by(data, USUBJID) %>%
  mutate(nr_nes = count_vals(var = AVALC, val = "NE"))
```
Arguments

dataset  ADSL dataset
        The variables specified by new_vars and subject_keys are expected. For each element of new_vars at least one variable of the form of the right hand side value must be available in the dataset.

new_vars  New variables
        A named list of variables like \( \text{exprs}(\text{PHSDT} = \text{PHwSDT}, \text{PHEDT} = \text{PHwEDT}, \text{APHASE} = \text{APHASEw}) \) is expected. The left hand side of the elements defines a variable of the output dataset, the right hand side defines the source variables from the ADSL dataset in CDISC notation.

        If the lower case letter "w" is used it refers to a phase variable, if the lower case letters "xx" are used it refers to a period variable, and if both "xx" and "w" are used it refers to a subperiod variable.

        Only one type must be used, e.g., all right hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables.

        If period and subperiod variables are required, separate reference datasets must be created.

subject_keys  Variables to uniquely identify a subject
        A list of expressions where the expressions are symbols as returned by \text{exprs()} is expected.

Details

For each subject and each subperiod/period/phase where at least one of the source variable is not NA an observation is added to the output dataset.

Depending on the type of the source variable (subperiod, period, or phase) the variable ASPER, APERIOD, or APHASEN is added and set to the number of the subperiod, period, or phase.

The variables specified for new_vars (left hand side) are added to the output dataset and set to the value of the source variable (right hand side).

Value

A period reference dataset (see "Details" section)

See Also

derive_vars_period()

Creating auxiliary datasets: \text{consolidate_metadata()}, \text{create_query_data()}, \text{create_single_dose_dataset()}

Examples

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

# Create reference dataset for periods
adsl <- tribble(
create_period_dataset

```r
~USUBJID, ~AP01SDT, ~AP01EDT, ~AP02SDT, ~AP02EDT, ~TRT01A, ~TRT02A,
"1", "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07", "A", "B",
"2", "2021-02-02", "2021-03-02", "2021-03-03", "2021-04-01", "B", "A",
) %>%
mutate(
  across(matches("AP\d\d\[ES\]DT"), ymd)
) %>%
mutate(
  STUDYID = "xyz"
)

create_period_dataset(
  adsl,
  new_vars = exprs(APERSDT = APxxSDT, APEREDT = APxxEDT, TRTA = TRTxxA)
)

# Create reference dataset for phases
adsl <- tribble(
  ~USUBJID, ~PH1SDT, ~PH1EDT, ~PH2SDT, ~PH2EDT, ~APHASE1, ~APHASE2,
  "1", "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07", "TREATMENT", "FUP",
  "2", "2021-02-02", "2021-03-02", NA, NA, "TREATMENT", NA
) %>%
mutate(
  across(matches("PH\d\d\[ES\]DT"), ymd)
) %>%
mutate(
  STUDYID = "xyz"
)

create_period_dataset(
  adsl,
  new_vars = exprs(PHSDT = PHwSDT, PHEDT = PHwEDT, APHASE = APHASEw)
)

# Create reference datasets for subperiods
adsl <- tribble(
  ~USUBJID, ~P01S1SDT, ~P01S1EDT, ~P01S2SDT, ~P01S2EDT, ~P02S1SDT, ~P02S1EDT,
  "1", "2021-01-04", "2021-01-19", "2021-01-20", "2021-02-06", "2021-02-07", "2021-03-07",
  "2", "2021-02-02", "2021-03-02", NA, NA, "2021-03-03", "2021-04-01"
) %>%
mutate(
  across(matches("P\d\dS\d\[ES\]DT"), ymd)
) %>%
mutate(
  STUDYID = "xyz"
)

create_period_dataset(
  adsl,
  new_vars = exprs(ASPRSDT = PxxSwSDT, ASPREDT = PxxSwEDT)
)
```
**create_query_data**

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

**Usage**

```r
create_query_data(queries, version = NULL, get_terms_fun = NULL)
```

**Arguments**

- **queries**
  - List of queries
  - A list of `query()` objects is expected.

- **version**
  - Dictionary version
  - The dictionary version used for coding the terms should be specified. If any of the queries is a basket (SMQ, SDG, ....) or a customized query including a basket, the parameter needs to be specified.
  - **Permitted Values**: A character string (the expected format is company-specific)

- **get_terms_fun**
  - Function which returns the terms
  - For each query specified for the `queries` parameter referring to a basket (i.e., those where the `definition` field is set to a `basket_select()` object or a list which contains at least one `basket_select()` object) the specified function is called to retrieve the terms defining the query. This function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level.
  - The function must return a dataset with all the terms defining the basket.
  - The output dataset must contain the following variables.
    - **SRCVAR**: the variable to be used for defining a term of the basket, e.g., `AEDECOD`
    - **TERMNAME**: the name of the term if the variable SRCVAR is referring to is character
    - **TERMID**: the numeric id of the term if the variable SRCVAR is referring to is numeric
    - **GRPNAME**: the name of the basket. The values must be the same for all observations.
  - The function must provide the following parameters
    - **basket_select**: A `basket_select()` object.
    - **version**: The dictionary version. The value specified for the version in the `create_query_data()` call is passed to this parameter.
    - **keep_id**: If set to `TRUE`, the output dataset must contain the GRPID variable. The variable must be set to the numeric id of the basket.
• temp_env: A temporary environment is passed to this parameter. It can be used to store data which is used for all baskets in the `create_query_data()` call. For example if SMQs need to be read from a database all SMQs can be read and stored in the environment when the first SMQ is handled. For the other SMQs the terms can be retrieved from the environment instead of accessing the database again.

Details

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

• a `basket_select()` object, the terms are read from the basket database by calling the function specified for the `get_terms_fun` parameter.
• a data frame, the terms stored in the data frame are used.
• a list of data frames and `basket_select()` objects, all terms from the data frames and all terms read from the basket database referenced by the `basket_select()` objects are collated.

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

• `PREFIX`: Prefix of the variables to be created by `derive_vars_query()` as specified by the `prefix` element.
• `GRPNAME`: Name of the query as specified by the `name` element.
• `GRPID`: Id of the query as specified by the `id` element. If the `id` element is not specified for a query, the variable is set to `NA`. If the `id` element is not specified for any query, the variable is not created.
• `SCOPE`: Scope of the query as specified by the `scope` element of the `basket_select()` object. For queries not defined by a `basket_select()` object, the variable is set to `NA`. If none of the queries is defined by a `basket_select()` object, the variable is not created.
• `SCOPEN`: Numeric scope of the query. It is set to 1 if the scope is broad. Otherwise it is set to 2. If the `add_scope_num` element equals `FALSE`, the variable is set to `NA`. If the `add_scope_num` element equals `FALSE` for all baskets or none of the queries is an basket, the variable is not created.
• `SRCVAR`: Name of the variable used to identify the terms.
• `TERMNAME`: Value of the term variable if it is a character variable.
• `TERMID`: Value of the term variable if it is a numeric variable.
• `VERSION`: Set to the value of the `version` argument. If it is not specified, the variable is not created.

Value

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

See Also

derive_vars_query(), query(), basket_select(), Queries Dataset Documentation

Creating auxiliary datasets: consolidate_metadata(), create_period_dataset(), create_single_dose_dataset()
Examples

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

# creating a query dataset for a customized query
cqterms <- tribble(~TERMNAME, ~TERMID,
                      "APPLICATION SITE ERYTHEMA", 10003041L,
                      "APPLICATION SITE PRURITUS", 10003053L)

  )
  mutate(SRCVAR = "AEDECOD")

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

create_query_data(queries = list(cq))

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

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

# The get_terms function from pharmaversesdtm is used for this example. # In a real application a company-specific function must be used.
create_query_data(
  queries = list(pregsmq, bilismq),
  get_terms_fun = pharmaversesdtm::get_terms,
  version = "20.1"
)

# create a query dataset for SDGs
sdg <- query("
create_single_dose_dataset

Create dataset of single doses

Description

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

Usage

create_single_dose_dataset(
    dataset,
    dose_freq = EXDOSFRQ,
    start_date = ASTDT,
    start_datetime = NULL,
    end_date = AENDT,
    end_datetime = NULL,
    lookup_table = dose_freq_lookup,
    lookup_column = CDISC_VALUE,
    nominal_time = NULL,
    keep_source_vars = expr_c(exprs(USUBJID), dose_freq, start_date, start_datetime,
                                end_date, end_datetime)
)

Arguments

dataset          Input dataset
                The columns specified by dose_freq, start_date and the end_date parameters are expected.

dose_freq        The dose frequency
                The aggregate dosing frequency used for multiple doses in a row.
                Permitted Values: defined by lookup table.

start_date       The start date
                A date object is expected. This object cannot contain NA values.
                Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

start_datetime   The start date-time
                A date-time object is expected. This object cannot contain NA values.
                Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object.
                If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.

end_date         The end date
                A date or date-time object is expected. This object cannot contain NA values.
                Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

end_datetime     The end date-time
                A date-time object is expected. This object cannot contain NA values.
                Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object.
                If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.
lookup_table  The dose frequency value lookup table
        The table used to look up dose_freq values and determine the appropriate multi-
        plier to be used for row generation. If a lookup table other than the default is
        used, it must have columns DOSE_WINDOW, DOSE_COUNT, and CONVERSION_FACTOR.
        The default table dose_freq_lookup is described in detail here.
        Permitted Values for DOSE_WINDOW: "MINUTE", "HOUR", "DAY", "WEEK", "MONTH", 
        "YEAR"
lookup_column  The dose frequency value column in the lookup table
        The column of lookup_table.
nominal_time  The nominal relative time from first dose (NFRLT)
        Used for PK analysis, this will be in hours and should be 0 for the first dose.
        It can be derived as (VISITDY - 1) * 24 for example. This will be expanded as
        the single dose dataset is created. For example an EXDOFRQ of "QD" will result
        in the nominal_time being incremented by 24 hours for each expanded record.
        The value can be NULL if not needed.
keep_source_vars  List of variables to be retained from source dataset
        This parameter can be specified if additional information is required in the out-
        put dataset. For example EXTRT for studies with more than one drug.

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

Value
        The input dataset with a single dose per row.

See Also
        Creating auxiliary datasets: consolidate_metadata(), create_period_dataset(), create_query_data()
create_single_dose_dataset

"P01", "Q2D", ymd("2021-01-01"), ymd_hms("2021-01-01 10:30:00"),
ymd("2021-01-07"), ymd_hms("2021-01-07 11:30:00"),
"P01", "Q3D", ymd("2021-01-08"), ymd_hms("2021-01-08 12:00:00"),
ymd("2021-01-14"), ymd_hms("2021-01-14 14:00:00"),
"P01", "EVERY 2 WEEKS", ymd("2021-01-15"), ymd_hms("2021-01-15 09:57:00"),
ymd("2021-01-29"), ymd_hms("2021-01-29 10:57:00")
)

create_single_dose_dataset(data)

# Example with custom lookup

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

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

create_single_dose_dataset(data, lookup_table = custom_lookup,
lookup_column = Value,
start_datetime = ASTDTM,
end_datetime = AENDTM)

# Example with nominal time

data <- tribble(
~USUBJID, ~EXDOSFRQ, ~NFRLT, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
"P01", "BID", 0, ymd("2021-01-01"), ymd_hms("2021-01-01 08:00:00"),
ymd("2021-01-07"), ymd_hms("2021-01-07 20:00:00"),
"P01", "BID", 168, ymd("2021-01-08"), ymd_hms("2021-01-08 08:00:00"),
ymd("2021-01-14"), ymd_hms("2021-01-14 20:00:00"),
"P01", "BID", 336, ymd("2021-01-15"), ymd_hms("2021-01-15 08:00:00"),
ymd("2021-01-29"), ymd_hms("2021-01-29 20:00:00")
)

create_single_dose_dataset(data, dose_freq = EXDOSFRQ,
start_date = ASTDT,
start_datetime = ASTDTM,
end_date = AENDT,
end_datetime = AENDTM,
lookup_table = dose_freq_lookup,
lookup_column = CDISC_VALUE,
nominal_time = NFRLT,
create_single_dose_dataset

```r
define_parent_vars = exprs(  
  USUBJID, EXDOSFRQ, ASTDT, AENDT, AENDTM, NFRLT  
)

# Example - derive a single dose dataset with imputations

# For either single drug administration records, or multiple drug administration
# records covering a range of dates, fill-in of missing treatment end datetime
# 'EXENDTC' by substitution with an acceptable alternate, for example date of
# death, date of datacut may be required. This example shows the
# maximum possible number of single dose records to be derived. The example
# requires the date of datacut 'DCUTDT' to be specified correctly, or
# if not appropriate to use 'DCUTDT' as missing treatment end data and missing
# treatment end datetime could set equal to treatment start date and treatment
# start datetime. ADSL variables 'DTHDT' and 'DCUTDT' are preferred for
# imputation use.
#
# All available trial treatments are included, allowing multiple different
# last dose variables to be created in for example `use_ad_template("ADAЕ")`
# if required.

adsl <- tribble(  
  ~STUDYID, ~USUBJID, ~DTHDT,  
  "01", "1211", ymd("2013-01-14"),  
  "01", "1083", ymd("2013-08-02"),  
  "01", "1445", ymd("2014-11-01"),  
  "01", "1015", NA,  
  "01", "1023", NA  
)

ex <- tribble(  
  ~STUDYID, ~USUBJID, ~EXSEQ, ~EXTRT, ~EXDOSE, ~EXDOSU, ~EXDOSFRQ, ~EXSTDTC, ~EXENDTC,  
  "01", "1015", 1, "PLAC", 0, "mg", "QD", "2014-01-02", "2014-01-16",  
  "01", "1015", 2, "PLAC", 0, "mg", "QD", "2014-06-17", "2014-06-18",  
  "01", "1015", 3, "PLAC", 0, "mg", "QD", "2014-06-19", NA_character_,  
  "01", "1023", 1, "PLAC", 0, "mg", "QD", "2012-08-05", "2012-08-27",  
  "01", "1023", 2, "PLAC", 0, "mg", "QD", "2012-08-28", "2012-09-01",  
  "01", "1211", 2, "XANO", 54, "mg", "QD", "2012-11-29", NA_character_,  
  "01", "1083", 1, "PLAC", 0, "mg", "QD", "2013-07-22", "2013-08-01"  
)

adsl_death <- adsl %>%
  mutate(    
    DTHDM = convert_date_to_dtm(DTHDT),
    # Remove 'DCUT' setup line below if ADSL 'DCUTDT' is populated.
    DCUTDT = convert_dtc_to_dt("2015-03-06"), # Example only, enter date.
    DCUTDTM = convert_date_to_dtm(DCUTDT)
  )
```
# Select valid dose records, non-missing `EXSTDTC` and `EXDOSE`.
ex_mod <- ex %>%
  filter(!is.na(EXSTDTC) & !is.na(EXDOSE)) %>%
  derive_vars_merged(adsl_death, by_vars = exprs(STUDYID, USUBJID)) %>%
  # Example, set up missing `EXDOSFRQ` as QD daily dosing regime.
  # Replace with study dosing regime per trial treatment.
  mutate(EXDOSFRQ = if_else(is.na(EXDOSFRQ), "QD", EXDOSFRQ)) %>%
  # Create EXxxDTM variables and replace missing `EXENDTM`.
  derive_vars_dtm(
    dtc = EXSTDTC,
    new_vars_prefix = "EXST",
    date_imputation = "first",
    time_imputation = "first",
    flag_imputation = "none",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXSTDTM)) %>%
  derive_vars_dtm(
    dtc = EXENDTC,
    new_vars_prefix = "EXEN",
    max_dates = exprs(DTHDTM, DCUTDTM),
    date_imputation = "last",
    time_imputation = "last",
    flag_imputation = "none",
    highest_imputation = "Y",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXENDTM)) %>%
  # Select only unique values.
  # Removes duplicated records before final step.
  distinct(
    STUDYID, USUBJID, EXTRT, EXDOSE, EXDOSFRQ, DCUTDT, DTHDT, EXSTD, EXSTDT, EXSTDTM, EXENDT, EXENDTDM, EXSTDTC, EXENDTC
  )

create_single_dose_dataset(
  ex_mod,
  start_date = EXSTDT,
  start_datetime = EXSTDTM,
  end_date = EXENDT,
  end_datetime = EXENDTM,
  keep_source_vars = exprs(
    STUDYID, USUBJID, EXTRT, EXDOSE, EXDOSFRQ, DCUTDT, DTHDT, EXSTD, EXSTDT, EXSTDTM, EXENDT, EXENDTDM, EXSTDTC, EXENDTC
  )
)
Description

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

Usage

date_source(
  dataset_name, 
  filter = NULL, 
  date, 
  traceability_vars = NULL, 
  set_values_to = NULL 
)

Arguments

dataset_name  The name of the dataset, i.e. a string, used to search for the date.
filter          An unquoted condition for filtering dataset.
date            A variable or an expression providing a date. A date or a datetime can be specified. An unquoted symbol or expression is expected.
traceability_vars    A named list returned by exprs() defining the traceability variables, e.g. exprs(LALVDOM = "AE", LALVSEQ = AESEQ, LALVVAR = "AESTDTC"). The values must be a symbol, a character string, a numeric, an expression, or NA.
[Deprecated] Please use set_values_to instead.
set_values_to    Variables to be set

Value

An object of class date_source.

See Also

derive_var_extreme_dtm(), derive_var_extreme_dt() 
Source Objects: basket_select(), censor_source(), death_event, dthcaus_source(), event_joined(), event_source(), event(), query(), records_source(), tte_source() 

Examples

# treatment end date from ADSL
trt_end_date <- date_source(
  dataset_name = "adsl", 
  date = TRTEDT 
)

# lab date from LB where assessment was taken, i.e. not "NOT DONE"
lb_date <- date_source(
  dataset_name = "lb", 
)
death_event

    filter = LBSTAT != "NOT DONE" | is.na(LBSTAT),
    date = convert_dtc_to_dt(LBDTC)
  )

# death date from ADSL including traceability variables
depth_date <- date_source(
  dataset_name = "adsl",
  date = DTHDT,
  set_values_to = exprs(
    LALVDOM = "ADSL",
    LALVVAR = "DTHDT"
  )
)

---

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

deadth_event

lstalive_censor

ea_event

ea_ser_event

ea_gr1_event

ea_gr2_event

ea_gr3_event

ea_gr4_event

ea_gr5_event

ea_gr35_event

ea_sev_event

ea_wd_event
Details

To see the definition of the various objects simply print the object in the R console, e.g. print(death_event). For details of how to use these objects please refer to \texttt{derive\_param\_tte()}. 

See Also

\texttt{derive\_param\_tte(), tte\_source(), event\_source(), censor\_source()}

Source Objects: \texttt{basket\_select(), censor\_source(), date\_source(), dthcaus\_source(), event\_joined(), event\_source(), event(), query(), records\_source(), tte\_source()}

Examples

```r
# This shows the definition of all pre-defined `tte\_source` objects that ship
# with {admiral}
for (obj in list\_tte\_source\_objects()$object) {
cat(obj, "\n")
print(get(obj))
cat("\n")
}
```

---

\texttt{default\_qtc\_paramcd} \hspace{1cm} \textit{Get Default Parameter Code for Corrected QT}

\section*{Description}

Get Default Parameter Code for Corrected QT

\section*{Usage}

\texttt{default\_qtc\_paramcd(method)}

\section*{Arguments}

\begin{itemize}
  \item \textbf{method} \hspace{1cm} Method used to QT correction  
  
  \begin{itemize}
    \item Permitted Values: "Bazett", "Fridericia", "Sagie"
  \end{itemize}
\end{itemize}

\section*{Value}

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

\section*{See Also}

\texttt{derive\_param\_qtc()}

BDS-Findings Functions for adding Parameters/Records: derive\_expected\_records(), derive\_extreme\_event(), derive\_extreme\_records(), derive\_locf\_records(), derive\_param\_bmi(), derive\_param\_bsa(), derive\_param\_computed(), derive\_param\_doseint(), derive\_param\_exist\_flag(), derive\_param\_exposure(), derive\_param\_extreme\_record(), derive\_param\_framingham(), derive\_param\_map(), derive\_param\_qtc(), derive\_param\_rr(), derive\_param\_wbc\_abs(), derive\_summary\_records()
**derivation_slice**

**Examples**

default_qtc_paramcd("Sagie")

---

### derivation_slice

**Create a derivation_slice Object**

**Description**

Create a derivation_slice object as input for slice_derivation().

**Usage**

derivation_slice(filter, args = NULL)

**Arguments**

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

**Value**

An object of class derivation_slice

**See Also**

slice_derivation(), params()

Higher Order Functions: call_derivation(), restrict_derivation(), slice_derivation()

---

### derive_basetype_records

**Derive Basetype Variable**

**Description**

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

**Usage**

derive_basetype_records(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 `rlang::exprs()` function
  - The names correspond to the values of the newly created `BASETYPE` variables and the expressions are used to subset the input dataset.

**Details**

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

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

**Value**

The input dataset with variable `BASETYPE` added

**See Also**

BDS-Findings Functions that returns variable appended to dataset: `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr_dir()`, `derive_var_atoxgr()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_ontrtf1()`, `derive_var_pchg()`, `derive_var_shift()`

**Examples**

```r
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(rlang)

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

bds_with_basetype <- derive_basetype_records(
  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"

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

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

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

bds_with_basetype <- derive_basetype_records(
  dataset = bds,
  basetypes = exprs(
    "LAST" = TRUE,
    "WORST" = TRUE
  )
)

print(bds_with_basetype, n = Inf)

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

---

**derive_expected_records**  
* Derive Expected Records*

**Description**

Add expected records as new observations for each 'by group' when the dataset contains missing observations.

**Usage**

```r
derive_expected_records(
  dataset,
  dataset_expected_obs,
  dataset_ref,
  by_vars = NULL,
  set_values_to = NULL
)
```
Arguments

- **dataset**: Input dataset
  A data frame, the columns from `dataset_ref` and specified by the `by_vars` parameter are expected.

- **dataset_expected_obs**: Deprecated, please use `dataset_ref` instead.

- **dataset_ref**: Expected observations dataset
  Data frame with the expected observations, e.g., all the expected combinations of `PARAMCD, PARAM, AVISIT, AVISITN, ...`

- **by_vars**: Grouping variables
  For each group defined by `by_vars` those observations from `dataset_ref` are added to the output dataset which do not have a corresponding observation in the input dataset.

- **set_values_to**: Variables to be set
  The specified variables are set to the specified values for the new observations.
  A list of variable name-value pairs is expected.
  - LHS refers to a variable.
  - RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, `NA`, or expressions, e.g., `exprs(PARAMCD = "TDOSE", PARCAT1 = "OVERALL")`.

Details

For each group (the variables specified in the `by_vars` parameter), those records from `dataset_ref` that are missing in the input dataset are added to the output dataset.

Value

The input dataset with the missed expected observations added for each `by_vars`. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars` or `set_values_to`.

See Also

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exist_flag()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```r
library(tibble)

adqs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVISITN, ~AVISIT, ~AVAL,
  "1", "a", 1, "WEEK 1", 10,
  "1", "b", 1, "WEEK 1", 11,
)```
Example 1. visit variables are parameter independent

```r
parm_visit_ref <- tribble(
  ~AVISITN, ~AVISIT,  
  1, "WEEK 1",  
  2, "WEEK 2"
)
```

```r
derive_expected_records(
  dataset = adqs,  
  dataset_ref = parm_visit_ref,  
  by_vars = exprs(USUBJID, PARAMCD),  
  set_values_to = exprs(DTYPE = "DERIVED")
)
```

Example 2. visit variables are parameter dependent

```r
parm_visit_ref <- tribble(
  ~PARAMCD, ~AVISITN, ~AVISIT,  
  "a", 1, "WEEK 1",  
  "a", 2, "WEEK 2",  
  "b", 1, "WEEK 1"
)
```

```r
derive_expected_records(
  dataset = adqs,  
  dataset_ref = parm_visit_ref,  
  by_vars = exprs(USUBJID, PARAMCD),  
  set_values_to = exprs(DTYPE = "DERIVED")
)
```

---

**derive_extreme_event**  
*Add the Worst or Best Observation for Each By Group as New Records*

**Description**

Add the first available record from events for each by group as new records, all variables of the selected observation are kept. It can be used for selecting the extreme observation from a series of user-defined events. This distinguishes `derive_extreme_event()` from `derive_extreme_records()`, where extreme records are derived based on certain order of existing variables.

**Usage**

```r
derive_extreme_event(
  dataset,  
  by_vars = NULL,  
  events,
)```
derive_extreme_event

order,
mode,
source_datasets = NULL,
ignore_event_order = FALSE,
check_type = "warning",
set_values_to,
keep_source_vars = exprs(everything())
)

Arguments

dataset Input dataset
The variables specified by the order and the by_vars parameter are expected.

by_vars Grouping variables
Default: NULL
Permitted Values: list of variables created by exprs()

events Conditions and new values defining events
A list of event() or event_joined() objects is expected. Only observations listed in the events are considered for deriving extreme event. If multiple records meet the filter condition, take the first record sorted by order. The data is grouped by by_vars, i.e., summary functions like all() or any() can be used in condition.
For event_joined() events the observations are selected by calling filter_joined. The condition field is passed to the filter argument.

order Sort order
If a particular event from events has more than one observation, within the event and by group, the records are ordered by the specified order.
Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

mode Selection mode (first or last)
If a particular event from events has more than one observation, "first"/"last" is to select the first/last record of this type of events sorting by order.
Permitted Values: "first", "last"

source_datasets Source datasets
A named list of datasets is expected. The dataset_name field of event() and event_joined() refers to the dataset provided in the list.

ignore_event_order Ignore event order
If the argument is set to TRUE, all events defined by events are considered equivalent. If there is more than one observation per by group the first or last (with respect to mode and order) is select without taking the order of the events into account.
Permitted Values: TRUE, FALSE
check_type  
Check uniqueness?
If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Default: "warning"
Permitted Values: "none", "warning", "error"

set_values_to Variables to be set
The specified variables are set to the specified values for the new observations.
A list of variable name-value pairs is expected.
  • LHS refers to a variable.
  • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression, or NA, e.g., `exprs(PARAMCD = "TDOSE", PARCAT1 = "OVERALL")`.

keep_source_vars
Variables to keep from the source dataset
For each event the specified variables are kept from the selected observations.
The variables specified for `by_vars` and created by `set_values_to` are always kept.

Permitted Values: A list of expressions where each element is a symbol or a tidyselect expression, e.g., `exprs(VISIT, VISITNUM, starts_with("RS"))`.

Details

1. For each event select the observations to consider:
   (a) If the event is of class `event`, the observations of the source dataset are restricted by condition and then the first or last (mode) observation per by group (`by_vars`) is selected.
      If the event is of class `event_joined`, `filter_joined()` is called to select the observations.
   (b) The variables specified by the `set_values_to` field of the event are added to the selected observations.
   (c) Only the variables specified for the `keep_source_vars` field of the event, and the by variables (`by_vars`) and the variables created by `set_values_to` are kept.

2. For each group (with respect to the variables specified for the `by_vars` parameter) the first event is selected. If there is more than one observation per event the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is selected.

3. The variables specified by the `set_values_to` parameter are added to the selected observations.

4. The observations are added to input dataset.

Value
The input dataset with the best or worst observation of each by group added as new observations.
### derive_extreme_event

**See Also**

- `event()`, `event_joined()`

**BDS-Findings Functions for adding Parameters/Records:**
- `default_qtc_paramcd()`, `derive_expected_records()`,
- `derive_locs_records()`, `derive_param_bmi()`, `derive_param_bsa()`,
- `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`,
- `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`,
- `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

**Examples**

```r
library(tibble)
library(dplyr)
library(lubridate)

adqs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVALC, ~ADY,
  "1", "NO SLEEP", "N", 1,
  "1", "WAKE UP", "N", 2,
  "1", "FALL ASLEEP", "N", 3,
  "2", "NO SLEEP", "N", 1,
  "2", "WAKE UP", "N", 2,
  "2", "FALL ASLEEP", "N", 4,
  "3", "NO SLEEP", NA_character_, 1
)

# Add a new record for each USUBJID storing the the worst sleeping problem.
derive_extreme_event(adqs,
by_vars = exprs(USUBJID),
events = list(
  event(
    condition = PARAMCD == "NO SLEEP" & AVALC == "Y",
    set_values_to = exprs(AVALC = "No sleep", AVAL = 1)
  ),
  event(
    condition = PARAMCD == "WAKE UP" & AVALC == "Y",
    set_values_to = exprs(AVALC = "Waking up more than three times", AVAL = 2)
  ),
  event(
    condition = PARAMCD == "FALL ASLEEP" & AVALC == "Y",
    set_values_to = exprs(AVALC = "More than 30 mins to fall asleep", AVAL = 3)
  ),
  event(
    condition = all(AVALC == "N"),
    set_values_to = exprs(
      AVALC = "No sleeping problems", AVAL = 4
    )
  ),
  event(
    condition = TRUE,
    set_values_to = exprs(
      AVALC = "No sleeping problems", AVAL = 4
    )
  )
)
```

derive_extreme_event

```r
define_extreme_event()

set_values_to = exprs(AVALC = "Missing", AVAL = 99)
)
),
order = exprs(ADY),
mode = "last",
set_values_to = exprs(
  PARAMCD = "WSP",
  PARAM = "Worst Sleeping Problems"
)
)

# Use different mode by event
adhy <- tribble(~USUBJID, ~AVISITN, ~CRIT1FL,
  "1", 1, "y",
  "1", 2, "y",
  "2", 1, "y",
  "2", 2, NA_character_,
  "2", 3, "y",
  "2", 4, NA_character_)
  %>%
  mutate(
    PARAMCD = "ALKPH",
    PARAM = "Alkaline Phosphatase (U/L)"
)

derive_extreme_event(
  adhy,
  by_vars = exprs(USUBJID),
  events = list(
    event(
      condition = is.na(CRIT1FL),
      set_values_to = exprs(AVALC = "N")
    ),
    event(
      condition = CRIT1FL == "Y",
      mode = "last",
      set_values_to = exprs(AVALC = "Y")
    )
  ),
  order = exprs(AVISITN),
  mode = "first",
  keep_source_vars = exprs(AVISITN),
  set_values_to = exprs(
    PARAMCD = "ALK2",
    PARAM = "ALKPH <= 2 times ULN"
  )
)

# Derive confirmed best overall response (using event_joined())
# CR - complete response, PR - partial response, SD - stable disease
# NE - not evaluable, PD - progressive disease
adsl <- tribble(
```
derive_extreme_event

```r
~USUBJID, ~TRTSDTC, 
"1", "2020-01-01", 
"2", "2019-12-12", 
"3", "2019-11-11", 
"4", "2019-12-30", 
"5", "2020-01-01", 
"6", "2020-02-02", 
"7", "2020-02-02", 
"8", "2020-02-01"
) %>%
  mutate(TRTSDT = ymd(TRTSDTC))

adrs <- tribble(
  ~USUBJID, ~ADTC, ~AVALC, 
  "1", "2020-01-01", "PR", 
  "1", "2020-02-01", "CR", 
  "1", "2020-02-16", "NE", 
  "1", "2020-03-01", "CR", 
  "1", "2020-04-01", "SD", 
  "2", "2020-01-01", "SD", 
  "2", "2020-02-01", "PR", 
  "2", "2020-03-01", "SD", 
  "2", "2020-03-13", "CR", 
  "4", "2020-01-01", "PR", 
  "4", "2020-03-01", "NE", 
  "4", "2020-04-01", "NE", 
  "4", "2020-05-01", "PR", 
  "5", "2020-01-01", "PR", 
  "5", "2020-01-10", "PR", 
  "5", "2020-01-20", "PR", 
  "6", "2020-02-06", "PR", 
  "6", "2020-02-16", "CR", 
  "6", "2020-03-30", "PR", 
  "7", "2020-02-06", "PR", 
  "7", "2020-02-16", "CR", 
  "7", "2020-04-01", "NE", 
  "8", "2020-02-16", "PD"
) %>%
  mutate(
    ADT = ymd(ADTC),
    PARAMCD = "OVR",
    PARAM = "Overall Response by Investigator"
  ) %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(TRTSDT)
  )

derive_extreme_event(
  adrs,
  by_vars = exprs(USUBJID),
  order = exprs(ADT),
  order = exprs(ADT),
)
derive_extreme_event

mode = "first",
source_datasets = list(adsl = adsl),
events = list(
  event_joined(
    description = paste(
      "CR needs to be confirmed by a second CR at least 28 days later",
      "at most one NE is acceptable between the two assessments"
    ),
    join_vars = exprs(AVALC, ADT),
    join_type = "after",
    first_cond = AVALC.join == "CR" &
      ADT.join >= ADT + 28,
    condition = AVALC == "CR" &
      all(AVALC.join %in% c("CR", "NE")) &
      count_vals(var = AVALC.join, val = "NE") <= 1,
    set_values_to = exprs(
      AVALC = "CR"
    )
  ),
  event_joined(
    description = paste(
      "PR needs to be confirmed by a second CR or PR at least 28 days later",
      "at most one NE is acceptable between the two assessments"
    ),
    join_vars = exprs(AVALC, ADT),
    join_type = "after",
    first_cond = AVALC.join %in% c("CR", "PR") &
      ADT.join >= ADT + 28,
    condition = AVALC == "PR" &
      all(AVALC.join %in% c("CR", "PR", "NE")) &
      count_vals(var = AVALC.join, val = "NE") <= 1,
    set_values_to = exprs(
      AVALC = "PR"
    )
  ),
  event(
    description = paste(
      "CR, PR, or SD are considered as SD if occurring at least 28",
      "after treatment start"
    ),
    condition = AVALC %in% c("CR", "PR", "SD") & ADT >= TRTSDT + 28,
    set_values_to = exprs(
      AVALC = "SD"
    )
  ),
  event(
    condition = AVALC == "PD",
    set_values_to = exprs(
      AVALC = "PD"
    )
  ),
  event(
    condition = AVALC %in% c("CR", "PR", "SD", "NE"),
    set_values_to = exprs(
      AVALC = "SD"
    )
  )
)
set_values_to = exprs(
  AVALC = "NE"
),
)
),
event(
  description = "set response to MISSING for patients without records in ADRS",
  dataset_name = "adsl",
  condition = TRUE,
  set_values_to = exprs(
    AVALC = "MISSING"
  ),
  keep_source_vars = exprs(TRTSDT)
),
set_values_to = exprs(
  PARAMCD = "CBOR",
  PARAM = "Best Confirmed Overall Response by Investigator"
)
) %>%
filter(PARAMCD == "CBOR")

derive_extreme_records

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

Description

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

Usage

derive_extreme_records(
  dataset = NULL,
  dataset_add = NULL,
  dataset_ref = NULL,
  by_vars = NULL,
  order = NULL,
  mode = NULL,
  filter_add = NULL,
  check_type = "warning",
  exist_flag = NULL,
  true_value = "Y",
  false_value = "N",
  keep_source_vars = exprs(everything())),

Arguments

dataset Input dataset
If dataset_add is not specified, the new records are selected from the input dataset. In this case the variables specified by by_vars and order are expected.

dataset_add Additional dataset
Observations from the specified dataset are added as new records to the input dataset (dataset).
All observations in the specified dataset fulfilling the condition specified by filter_source are considered. If mode and order are specified, the first or last observation within each by group, defined by by_vars, is selected.
If the argument is not specified, the input dataset (dataset) is used.
The variables specified by the by_vars and order argument (if applicable) are expected.

dataset_ref Reference dataset
The variables specified for by_vars are expected. For each observation of the specified dataset a new observation is added to the input dataset.

by_vars Grouping variables
If dataset_ref is specified, this argument must be specified.

Permitted Values: list of variables created by exprs()

order Sort order
Within each by group the observations are ordered by the specified order.

Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

mode Selection mode (first or last)
If "first" is specified, the first observation of each by group is added to the input dataset. If "last" is specified, the last observation of each by group is added to the input dataset.

Permitted Values: "first", "last"

filter_add Filter for additional dataset (dataset_add)
Only observations in dataset_add fulfilling the specified condition are considered.

check_type Check uniqueness?
If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

Permitted Values: "none", "warning", "error"

exist_flag Existence flag
The specified variable is added to the output dataset.
derive_extreme_records

For by groups with at least one observation in the additional dataset (dataset_add) exist_flag is set to the value specified by the true_value argument. For all other by groups exist_flag is set to the value specified by the false_value argument.

**Permitted Values**: Variable name

**true_value**

True value

For new observations selected from the additional dataset (dataset_add), exist_flag is set to the specified value.

**false_value**

False value

For new observations not selected from the additional dataset (dataset_add), exist_flag is set to the specified value.

**keep_source_vars**

Variables to be kept in the new records

A named list or tidyselect expressions created by exprs() defining the variables to be kept for the new records. The variables specified for by_vars and set_values_to need not be specified here as they are kept automatically.

**set_values_to**

Variables to be set

The specified variables are set to the specified values for the new observations.

A list of variable name-value pairs is expected.

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression, or NA, e.g., exprs(PARAMCD = "TDOSE", PARCAT1 = "OVERALL").

**filter**

Filter for observations to consider

Deprecated, please use the above filter_add argument instead.

Only observations fulfilling the specified condition are taken into account for selecting the first or last observation. If the argument is not specified, all observations are considered.

**Permitted Values**: a condition

**Details**

1. The additional dataset (dataset_add) is restricted as specified by the filter_add argument.

2. For each group (with respect to the variables specified for the by_vars argument) the first or last observation (with respect to the order specified for the order argument and the mode specified for the mode argument) is selected.

3. If dataset_ref is specified, observations which are in dataset_ref but not in the selected records are added.

4. The variables specified by the set_values_to argument are added to the selected observations.

5. The variables specified by the keep_source_vars argument are selected along with the variables specified in by_vars and set_values_to arguments.

6. The observations are added to input dataset.
**derive_extreme_records**

**Value**

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

**See Also**

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

**Examples**

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

adlb <- tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~LBSEQ,
  "1", 1, 113, 1,
  "1", 2, 113, 2,
  "1", 3, 117, 3,
  "2", 1, 101, 1,
  "2", 2, 101, 2,
  "2", 3, 95, 3
)
# Add a new record for each USUBJID storing the minimum value (first AVAL).
# If multiple records meet the minimum criterion, take the first value by
# AVISITN. Set AVISITN = 97 and DTYPE = MINIMUM for these new records.
# Specify the variables that need to be kept in the new records.
derive_extreme_records(
  adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVAL, AVISITN),
  mode = "first",
  filter_add = !is.na(AVAL),
  keep_source_vars = exprs(AVAL),
  set_values_to = exprs(
    AVISITN = 97,
    DTYPE = "MINIMUM"
  )
)

# Add a new record for each USUBJID storing the maximum value (last AVAL).
# If multiple records meet the maximum criterion, take the first value by
# AVISITN. Set AVISITN = 98 and DTYPE = MAXIMUM for these new records.
# Specify the variables that need to be kept in the new records.
derive_extreme_records(
  adlb,
  by_vars = exprs(USUBJID),
  order = exprs(desc(AVAL), AVISITN),
  mode = "first",
)
filter_add = !is.na(AVAL), 
set_values_to = exprs(
    AVISITN = 98,
    DTYPE = "MAXIMUM"
  )
)
)

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

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

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

derive_extreme_records(
  adrs,
  dataset_ref = adsl,
  dataset_add = adrs,
  by_vars = exprs(STUDYID, USUBJID),
  mode = "first",
  set_values_to = exprs(
    AVISITN = 99,
    DTYPE = "LOV"
  )
)
```r
filter_add = PARAMCD == "OVR" & AVALC == "PD",
order = exprs(ADT),
exist_flag = AVALC,
true_value = "Y",
false_value = "N",
mode = "first",
set_values_to = exprs(
  PARAMCD = "PD",
  PARAM = "Disease Progression",
  AVAL = yn_to_numeric(AVALC),
  ANL01FL = "Y",
  ADT = ADT
)
)

# derive parameter indicating death
derive_extreme_records(
  dataset_ref = adsl,
  dataset_add = adsl,
  by_vars = exprs(STUDYID, USUBJID),
  filter_add = !is.na(DTHDT),
  exist_flag = AVALC,
  true_value = "Y",
  false_value = "N",
  mode = "first",
  set_values_to = exprs(
    PARAMCD = "DEATH",
    PARAM = "Death",
    ANL01FL = "Y",
    ADT = DTHDT
  )
)
)
```

**derive_locf_records**  
*Derive LOCF (Last Observation Carried Forward) Records*

**Description**

Adds LOCF records as new observations for each 'by group' when the dataset does not contain observations for missed visits/time points.

**Usage**

```r
derive_locf_records(
  dataset,
  dataset_expected_obs,
  dataset_ref,
  by_vars,
  analysis_var = AVAL,
  order,
```
derive_locf_records

```
keep_vars = NULL
```

**Arguments**

- **dataset**
  - Input dataset
  - The columns specified by the `by_vars`, `analysis_var`, `order`, `keep_vars` parameters are expected.

- **dataset_expected_obs**
  - **Deprecated**, please use `dataset_ref` instead.

- **dataset_ref**
  - Expected observations dataset
  - Data frame with all the combinations of `PARAMCD`, `PARAM`, `AVISIT`, `AVISITN`, ... which are expected in the dataset is expected.

- **by_vars**
  - Grouping variables
  - For each group defined by `by_vars` those observations from `dataset_ref` are added to the output dataset which do not have a corresponding observation in the input dataset or for which `analysis_var` is `NA` for the corresponding observation in the input dataset.

- **analysis_var**
  - Analysis variable.
  - **Default**: `AVAL`
  - **Permitted Values**: a variable

- **order**
  - Sort order
  - The dataset is sorted by `order` before carrying the last observation forward (e.g. `AVAL`) within each `by_vars`.

- **keep_vars**
  - Variables that need carrying the last observation forward
  - Keep variables that need carrying the last observation forward other than `analysis_var` (e.g., `PARAMN`, `VISITNUM`). If by default `NULL`, only variables specified in `by_vars` and `analysis_var` will be populated in the newly created records.

**Details**

For each group (with respect to the variables specified for the `by_vars` parameter) those observations from `dataset_ref` are added to the output dataset

- which do not have a corresponding observation in the input dataset or
- for which `analysis_var` is `NA` for the corresponding observation in the input dataset.

For the new observations, `analysis_var` is set to the non-missing `analysis_var` of the previous observation in the input dataset (when sorted by `order`) and `DTYPE` is set to "LOCF".

**Value**

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

**Author(s)**

G Gayatri
See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```r
library(dplyr)
library(tibble)

advs <- tribble(
  ~STUDYID, ~USUBJID, ~PARAMCD, ~PARAMN, ~AVAL, ~AVISITN, ~AVISIT,
  "CDISC01", "01-701-1015", "PULSE", 1, 61, 0, "BASELINE",
  "CDISC01", "01-701-1015", "PULSE", 1, 60, 2, "WEEK 6",
  "CDISC01", "01-701-1015", "DIABP", 2, 51, 0, "BASELINE",
  "CDISC01", "01-701-1015", "DIABP", 2, 50, 2, "WEEK 2",
  "CDISC01", "01-701-1015", "DIABP", 2, 51, 4, "WEEK 4",
  "CDISC01", "01-701-1015", "DIABP", 2, 50, 6, "WEEK 6",
  "CDISC01", "01-701-1015", "SYSBP", 3, 121, 0, "BASELINE",
  "CDISC01", "01-701-1015", "DIABP", 2, 50, 2, "WEEK 2",
  "CDISC01", "01-701-1015", "SYSBP", 3, 121, 2, "WEEK 2",
  "CDISC01", "01-701-1015", "SYSBP", 3, 121, 4, "WEEK 4",
  "CDISC01", "01-701-1015", "SYSBP", 3, 121, 6, "WEEK 6",
  "CDISC01", "01-701-1028", "PULSE", 1, 65, 0, "BASELINE",
  "CDISC01", "01-701-1028", "DIABP", 2, 79, 0, "BASELINE",
  "CDISC01", "01-701-1028", "DIABP", 2, 80, 2, "WEEK 2",
  "CDISC01", "01-701-1028", "DIABP", 2, NA, 4, "WEEK 4",
  "CDISC01", "01-701-1028", "DIABP", 2, NA, 6, "WEEK 6",
  "CDISC01", "01-701-1028", "SYSBP", 3, 130, 0, "BASELINE",
  "CDISC01", "01-701-1028", "SYSBP", 3, 132, 2, "WEEK 2"
)

# A dataset with all the combinations of PARAMCD, PARAM, AVISIT, AVISITN, ... which are expected.
advs_expected_obsv <- tribble(
  ~PARAMCD, ~AVISITN, ~AVISIT,
  "PULSE", 0, "BASELINE",
  "PULSE", 6, "WEEK 6",
  "DIABP", 0, "BASELINE",
  "DIABP", 2, "WEEK 2",
  "DIABP", 4, "WEEK 4",
  "DIABP", 6, "WEEK 6",
  "SYSBP", 0, "BASELINE",
  "SYSBP", 2, "WEEK 2",
  "SYSBP", 4, "WEEK 4",
  "SYSBP", 6, "WEEK 6"
)

derive_locf_records(
  dataset = advs,
)
derive_param_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.

**Note:** This is a wrapper function for the more generic `derive_param_computed()`.

**Usage**

```r
derive_param_bmi(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "BMI"),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  get_unit_expr,
  filter = NULL,
  constant_by_vars = 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.
  - Only variables specified in `by_vars` will be populated in the newly created records.
  - **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 `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.
  - **Permitted Values:** List of variable-value pairs
**derive_param_bmi**

**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

**constant_by_vars**  
By variables for when HEIGHT is constant  
When HEIGHT is constant, the HEIGHT parameters (measured only once) are merged to the other parameters using the specified variables.  
If height is constant (e.g. only measured once at screening or baseline) then use constant_by_vars to select the subject-level variable to merge on (e.g. USUBJID). This will produce BMI at all visits where weight is measured. Otherwise it will only be calculated at visits with both height and weight collected.  
*Permitted Values:* list of variables

**Details**

The analysis value of the new parameter is derived as

\[
BMI = \frac{WEIGHT}{HEIGHT^2}
\]

**Value**

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

**See Also**

compute_bmi()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()
Examples

# Example 1: Derive BMI where height is measured only once using constant_by_vars
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVISIT,  
  "01-701-1015", "HEIGHT", "Height (cm)", 147, "SCREENING", 
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.0, "SCREENING", 
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.4, "BASELINE", 
  "01-701-1015", "WEIGHT", "Weight (kg)", 53.1, "WEEK 2", 
  "01-701-1028", "HEIGHT", "Height (cm)", 163, "SCREENING", 
  "01-701-1028", "WEIGHT", "Weight (kg)", 78.5, "SCREENING", 
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.3, "BASELINE", 
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.7, "WEEK 2"
)

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

# Example 2: Derive BMI where height is measured only once and keep only one record 
# where both height and weight are measured.

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

# Example 3: Pediatric study where height and weight are measured multiple times
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVISIT,  
  "01-101-1001", "HEIGHT", "Height (cm)", 47.1, "BASELINE", 
  "01-101-1001", "HEIGHT", "Height (cm)", 59.1, "WEEK 12", 
  "01-101-1001", "HEIGHT", "Height (cm)", 64.7, "WEEK 24", 
  "01-101-1001", "HEIGHT", "Height (cm)", 68.2, "WEEK 48", 
  "01-101-1001", "WEIGHT", "Weight (kg)", 2.6, "BASELINE", 
  "01-101-1001", "WEIGHT", "Weight (kg)", 5.3, "WEEK 12", 
  "01-101-1001", "WEIGHT", "Weight (kg)", 6.7, "WEEK 24",
)
derive_param_bsa

"01-101-1001", "WEIGHT", "Weight (kg)", 7.4, "WEEK 48",
)

derive_param_bmi(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = exprs(
    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.

**Note:** This is a wrapper function for the more generic `derive_param_computed()`.

#### Usage

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

**by_vars**
Grouping variables
For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records.

*Permitted Values*: list of variables

**method**
Derivation method to use. Note that `HEIGHT` is expected in cm and `WEIGHT` is expected in kg:

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

*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 `exprs(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

**constant_by_vars**
By variables for when HEIGHT is constant
When HEIGHT is constant, the HEIGHT parameters (measured only once) are merged to the other parameters using the specified variables.

If height is constant (e.g. only measured once at screening or baseline) then use `constant_by_vars` to select the subject-level variable to merge on (e.g. `USUBJID`). This will produce BSA at all visits where weight is measured. Otherwise it will only be calculated at visits with both height and weight collected.

*Permitted Values*: list of variables
**Value**

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

**See Also**

`compute_bsa()`

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

**Examples**

```r
library(tibble)

# Example 1: Derive BSA where height is measured only once using constant_by_vars
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 170, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 75, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 78, "MONTH 1",
  "01-701-1015", "WEIGHT", "Weight (kg)", 80, "MONTH 2",
  "01-701-1015", "HEIGHT", "Height (cm)", 185, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 90, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 88, "MONTH 1",
  "01-701-1028", "WEIGHT", "Weight (kg)", 85, "MONTH 2",
)

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

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

**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

\[ MAP = \frac{SYSBP + 2 \times DIABP}{3} \]

**Usage**

```r
derive_param_computed(
  dataset = NULL,
  dataset_add = NULL,
  by_vars,
  parameters,
  analysis_var = AVAL,
  analysis_value,
  set_values_to,
  filter = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL,
  keep_nas = FALSE
)
```

**Arguments**

- **dataset**
  - Input dataset
  - The variables specified by the `by_vars` parameter 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`.

- **dataset_add**
  - Additional dataset
  - The variables specified by the `by_vars` parameter are expected.
  - The variable specified by `by_vars` and `PARAMCD` must be a unique key of the additional dataset after restricting it to the parameters specified by `parameters`.
  - If the argument is specified, the observations of the additional dataset are considered in addition to the observations from the input dataset (`dataset` restricted by `filter`).

- **by_vars**
  - Grouping variables
  - For each group defined by `by_vars` an observation is added to the output dataset.
  - Only variables specified in `by_vars` will be populated in the newly created records.
  - *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.
  - If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression the condition for selecting the records. For example
parameters = exprs(HGHT = VSTESTCD == "HEIGHT") selects the observations with VSTESTCD == "HEIGHT" from the input data (dataset and dataset_add), sets PARAMCD = "HGHT" for these observations, and adds them to the observations to consider.

Unnamed elements in the list of expressions are considered as parameter codes. For example, parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT") uses the parameter code "WEIGHT" and creates a temporary parameter code "HGHT".

**Permitted Values:** A character vector of PARAMCD values or a list of expressions

**analysis_var**  
Analysis variable

[Deprecated] Please use set_values_to instead.

The specified variable is set to the value of analysis_value for the new observations.

**Permitted Values:** An unquoted symbol

**analysis_value**  
Definition of the analysis value

[Deprecated] Please use set_values_to instead.

An expression defining the analysis value (AVAL) of the new parameter is expected. The values of variables of the parameters specified by parameters can be accessed using <variable name>..<parameter code>, e.g., AVAL.SYSBP. Variable names in the expression must not contain more than one dot.

**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. The values of variables of the parameters specified by parameters can be accessed using <variable name>..<parameter code>. For example

```r
exprs(
  AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
  PARAMCD = "MAP"
)
```

defines the analysis value and parameter code for the new parameter. Variable names in the expression must not contain more than one dot.

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

**Permitted Values:** list of variables

**constant_parameters**  
Required constant parameter codes

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

If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression the condition for selecting the records. For example, `constant_parameters = exprs(HGHT = VSTESTCD == "HEIGHT")` selects the observations with VSTESTCD == "HEIGHT" from the input data (dataset and dataset_add), sets PARAMCD = "HGHT" for these observations, and adds them to the observations to consider.

Unnamed elements in the list of expressions are considered as parameter codes. For example, `constant_parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT")` uses the parameter code "WEIGHT" and creates a temporary parameter code "HGHT".

**Permitted Values:** A character vector of PARAMCD values or a list of expressions

**keep_nas**
Keep observations with NAs

If the argument is set to TRUE, observations are added even if some of the values contributing to the computed value are NA.

**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 (dataset) or the additional dataset (dataset_add) contains exactly one observation for each parameter code specified for parameters.

For the new observations the variables specified for set_values_to are set to the provided values. The values of the other variables of the input dataset are set to NA.

**Value**

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

**See Also**

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

**Examples**

```r
library(tibble)
library(dplyr)
library(lubridate)

# Example 1: Derive MAP
advs <- tribble(
```
```
# Example 1: Derive MAP where diastolic blood pressure is measured only once
advs <- 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_param_computed(  
advs,  
by_vars = exprs(USUBJID, VISIT),  
parameters = c("SYSBP", "DIABP"),  
set_values_to = exprs(    
    AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,  
    PARAMCD = "MAP",  
    PARAM = "Mean Arterial Pressure (mmHg)",  
    AVALU = "mmHg"
  )
)
```

```
# Example 2: Derive BMI where height is measured only once
advs <- tribble(~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,  
                 "01-701-1015", "HEIGHT", "Height (cm)", 147.0, "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)", 163.0, "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_computed(  
advs,  
by_vars = exprs(USUBJID, VISIT),  
parameters = "WEIGHT",  
set_values_to = exprs(    
    AVAL = AVAL.WEIGHT / (AVAL.HEIGHT / 100)^2,  
    PARAMCD = "BMI",  
    PARAM = "Body Mass Index (kg/m^2)",  
    AVALU = "kg/m^2"
  ),  
constant_parameters = c("HEIGHT"),  
constant_by_vars = exprs(USUBJID)
)
```

```
# Example 3: Using data from an additional dataset and other variables than AVAL
qs <- tribble(~USUBJID, ~AVISIT, ~QSTESTCD, ~QSORRES, ~QSSTRESN,  
               "01-701-1015", "SCREENING", "QSTESTCD", "QSORRES", "QSSTRESN"
)
```
derive_param_computed

```r
derive_param_computed = 
  adchsf <- tribble(
    ~USUBJID, ~AVISIT, ~PARAMCD, ~QSSTRESN, ~AVAL, 
    "1", "WEEK 2", "CHSF112", NA, 1, 
    "1", "WEEK 2", "CHSF113", "Yes", NA, 
    "1", "WEEK 2", "CHSF114", NA, 1, 
    "1", "WEEK 4", "CHSF112", NA, 2, 
    "1", "WEEK 4", "CHSF113", "No", NA, 
    "1", "WEEK 4", "CHSF114", NA, 1 
  )

  adchsf <- tribble(
    ~USUBJID, ~AVISIT, ~PARAMCD, ~QSSTRESN, ~AVAL, 
    "1", "WEEK 2", "CHSF12", 1, 6, 
    "1", "WEEK 2", "CHSF14", 1, 6, 
    "1", "WEEK 4", "CHSF12", 2, 12, 
    "1", "WEEK 4", "CHSF14", 1, 6 
  )

  #> %>%
  mutate(QSORRES = NA_character_)

derive_param_computed(
  adchsf, 
  dataset_add = qs, 
  by_vars = exprs(USUBJID, AVISIT), 
  parameters = exprs(CHSF12, CHSF13 = QSTESTCD %in% c("CHSF113", "CHSF213"), CHSF14), 
  set_values_to = exprs( 
    AVAL = case_when(
      QSORRES.CHSF13 == "Not applicable" ~ 0, 
      QSORRES.CHSF13 == "Yes" ~ 38, 
      QSORRES.CHSF13 == "No" ~ if_else(
        QSSTRESN.CHSF12 > QSSTRESN.CHSF14, 
        25, 
        0 
      ) 
    ), 
    PARAMCD = "CHSF13"
  )
)

# Example 4: Computing more than one variable
adlb_tbilialk <- tribble(
  ~USUBJID, ~PARAMCD, ~AVALC, ~ADTM, ~ADTF, 
  "1", "ALK2", "Y", "2021-05-13", NA_character_, 
  "1", "TBILI2", "Y", "2021-06-30", "D", 
  "2", "ALK2", "Y", "2021-12-31", "M", 
  "2", "TBILI2", "N", "2021-11-11", NA_character_, 
  "3", "ALK2", "N", "2021-04-03", NA_character_, 
  "3", "TBILI2", "N", "2021-04-04", NA_character_ 
  )

  #> %>%
  mutate(ADTM = ymd(ADTM))

derive_param_computed(
  dataset_add = adlb_tbilialk, 
  by_vars = exprs(USUBJID), 
  parameters = c("ALK2", "TBILI2"), 
  set_values_to = exprs( 
    ...
  )
)```

**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.

**Note:** This is a wrapper function for the more generic `derive_param_computed()`.

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

**Usage**

```r
derive_param_doseint(
  dataset,
  by_vars,
  set_values_to = exprs(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
  - Only variables specified in by_vars will be populated in the newly created records.
  - Permitted Values: list of variables
derive_param_doseint

**set_values_to** Variables to be set
The specified variables are set to the specified values for the new observations. For example `exprs(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 if 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

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

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_exist_flag(), derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

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

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

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

derive_param_doseint(
  adex,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(PARAMCD = "TDOSINT2"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSDOSINT",
  zero_doses = "100"
)

derive_param_exist_flag

Add an Existence Flag Parameter

Description

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

Usage

derive_param_exist_flag(
    dataset = NULL,
    dataset_ref,
    dataset_add,
    condition,
    true_value = "Y",
    false_value = NA_character_,
    missing_value = NA_character_,
    filter_add = NULL,
    aval_fun,
    by_vars = get_admiral_option("subject_keys"),
    set_values_to,
    dataset_adsl,
    subject_keys
)

Arguments

dataset  Input dataset
  The variables specified for by_vars and the PARAMCD variable are expected.

dataset_ref Reference dataset, e.g., ADSL
  The variables specified in by_vars are expected. For each group (as defined
  by by_vars) from the specified dataset (dataset_ref), the existence flag is
  calculated and added as a new observation to the input datasets (dataset).

dataset_add Additional dataset
  The variables specified by the by_vars parameter are expected.
  This dataset is used to check if an event occurred or not. Any observation in the
  dataset fulfilling the event condition (condition) is considered as an event.

condition Event condition
  The condition is evaluated at the additional dataset (dataset_add).
  For all groups where it evaluates as TRUE at least once AVALC is set to the true
  value (true_value) for the new observations.
  For all groups where it evaluates as FALSE or NA for all observations AVALC is set
  to the false value (false_value).
  For all groups not present in the additional dataset AVALC is set to the missing
  value (missing_value).

true_value True value
  For all groups with at least one observations in the additional dataset (dataset_add)
  fulfilling the event condition (condition), AVALC is set to the specified value
  (true_value).
  Default: "Y"
  Permitted Value: A character scalar

false_value False value
For all groups with at least one observations in the additional dataset (dataset_add) but none of them is fulfilling the event condition (condition), AVALC is set to the specified value (false_value).

Default: NA_character_
Permitted Value: A character scalar

missing_value
Values used for missing information

For all groups without an observation in the additional dataset (dataset_add), AVALC is set to the specified value (missing_value).

Default: NA_character_
Permitted Value: A character scalar

filter_add
Filter for additional data

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

aval_fun
Function to map character analysis value (AVALC) to numeric analysis value (AVAL)

Deprecated, please use set_values_to instead.

by_vars
Variables to uniquely identify a group

A list of symbols created using exprs() is expected.

set_values_to
Variables to set

A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "MDIS", PARAM = "Measurable Disease at Baseline") is expected. The values must be symbols, character strings, numeric values, NA, or expressions.

Deprecated, please use dataset_ref instead.

subject_keys

Deprecated, please use by_vars instead.

Details

1. The additional dataset (dataset_add) is restricted to the observations matching the filter_add condition.

2. For each group in dataset_ref a new observation is created.
   - The AVALC variable is added and set to the true value (true_value) if for the group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE.
   - It is set to the false value (false_value) if for the group at least one observation exists and for all observations the condition evaluates to FALSE or NA.
   - Otherwise, it is set to the missing value (missing_value), i.e., for those groups not in dataset_add.

3. The variables specified by the set_values_to parameter are added to the new observations.

4. The new observations are added to input dataset.
**Value**

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

**See Also**

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

**Examples**

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

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

tu <- tribble(~USUBJID, ~VISIT, ~TUSTRESC, ~TUTESTCD, ~STUDYID = "XX1234", ~TUMIDENT = "TUMIDENT")

derive_param_exist_flag(
  dataset_ref = adsl,
  dataset_add = tu,
  filter_add = TUMIDENT == "TUMIDENT" & VISIT == "SCREENING",
  condition = TUSTRESC == "TARGET",
  false_value = "N",
  missing_value = "N",
  set_values_to = exprs(
    AVAL = yn_to_numeric(AVALC),
    PARAMCD = "MDIS",
  )
)`
derive_param_exposure

PARAM = "Measurable Disease at Baseline"
)
)

derive_param_exposure  

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

Description

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

Usage

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

Arguments

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. Only variables specified in by_vars will be populated in the newly created records.
    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, NA, or an expression. (e.g. `exprs(PARAMCD = "TDOSE",PARCAT1 = "OVERALL")`).  
Permitted Values: List of variable-value pairs

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

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

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

If the input datasets contains

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

See Also

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```r
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(stringr, warn.conflicts = FALSE)
adex <- tribble(
```
derive_param_extreme_event

```r
~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("2014-01-16"),
) %>%
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 = exprs(USUBJID),
  set_values_to = exprs(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 = exprs(USUBJID),
  filter = VISIT %in% c("WEEK 2", "WEEK 24"),
  set_values_to = exprs(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 = exprs(USUBJID),
  set_values_to = exprs(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_extreme_event**

*Add an Extreme Event Parameter*

**Description**

[Deprecated]

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

Add a new parameter for the first or last event occurring in a dataset. The variable given in `new_var` indicates if an event occurred or not. For example, the function can derive a parameter for the first disease progression.

**Usage**

```r
derive_param_extreme_event(
    dataset = NULL,
    dataset_adsl,
    dataset_source,
    filter_source,
    order = NULL,
    new_var = NULL,
    true_value = "Y",
    false_value = "N",
    mode = "first",
    subject_keys = get_admiral_option("subject_keys"),
    set_values_to,
    check_type = "warning"
)
```

**Arguments**

- **dataset** Input dataset
  
  The `PARAMCD` variable is expected.

- **dataset_adsl** ADSL input dataset
  
  The variables specified for `subject_keys` are expected. For each observation of the specified dataset a new observation is added to the input dataset.

- **dataset_source** Source dataset
  
  All observations in the specified dataset fulfilling the condition specified by `filter_source` are considered as an event.
  
  The variables specified by the `subject_keys` and `order` argument (if applicable) are expected.

- **filter_source** Source filter
  
  All observations in `dataset_source` fulfilling the specified condition are considered as an event.
  
  For subjects with at least one event `new_var` is set to `true_value`.
  
  For all other subjects `new_var` is set to `false_value`.
derive_param_extreme_event

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>Order variable List of symbols for sorting the source dataset (dataset_source). Permitted Values: list of expressions created by exprs(), e.g., exprs(AOT, desc(AVAL)).</td>
</tr>
<tr>
<td>new_var</td>
<td>New variable The name of the variable which will indicate whether an event happened or not.</td>
</tr>
<tr>
<td>true_value</td>
<td>True value For all subjects with at least one observation in the source dataset (dataset_source) fulfilling the event condition (filter_source), new_var is set to the specified value true_value.</td>
</tr>
<tr>
<td>false_value</td>
<td>False value For all other subjects in dataset_adsl without an event, new_var is set to the specified value false_value.</td>
</tr>
</tbody>
</table>
| mode            | Selection mode (first or last) If "first" is specified, the first observation of each subject is selected. If "last" is specified, the last observation of each subject is selected. Permit:
dated Values: "first", "last" |
| subject_keys    | Variables to uniquely identify a subject A list of symbols created using exprs() is expected.                                               |
| set_values_to   | Variables to set A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "PD", PARAM = "Disease Progression") is expected. The values must be symbols, character strings, numeric values, NA, or an expression. Note, if you require a date or datetime variable to be populated, this needs to be defined here. |
| check_type      | Check uniqueness? If "warning" or "error" is specified, a message is issued if the observations of the source dataset (dataset_source) restricted by filter_source are not unique with respect to the subject keys (subject_key argument) and order. Permit:
dated Values: "none", "warning", "error" |

Details

1. The source dataset (dataset_source) is restricted to observations fulfilling filter_source.
2. For each subject (with respect to the variables specified for the subject_keys argument) either the first or last observation from the restricted source dataset is selected. This is depending on mode, (with respect to order, if applicable) where the event condition (filter_source argument) is fulfilled.
3. For each observation in dataset_adsl a new observation is created. For subjects with event new_var is set to true_value. For all other subjects new_var is set to false_value. For subjects with event all variables from dataset_source are kept. For subjects without event all variables which are in both dataset_adsl and dataset_source are kept.
4. The variables specified by the set_values_to argument are added to the new observations.
5. The new observations are added to input dataset.
**derive_param_extreme_record**

**Value**

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

**See Also**

Other deprecated: `derive_var_basetype()`, `derive_var_confirmation_flag()`, `derive_var_last_dose_amt()`, `derive_var_last_dose_date()`, `derive_var_last_dose_grp()`, `derive_var_merged_cat()`, `derive_var_merged_character()`, `derive_vars_last_dose()`

---

**Description**

Generates parameter based on the first or last observation from multiple source datasets, based on user-defined filter, order and by group criteria. All variables of the selected observation are kept.

**Usage**

```r
derive_param_extreme_record(
  dataset = NULL,
  sources,
  source_datasets,
  by_vars = NULL,
  order,
  mode,
  set_values_to
)
```

**Arguments**

- `dataset` Input dataset
- `sources` Sources
  - A list of `records_source()` objects is expected.
- `source_datasets` Source datasets
  - A named list of datasets is expected. The `dataset_name` field of `records_source()` refers to the dataset provided in the list. The variables specified by the `order` and the `by_vars` arguments are expected after applying `new_vars`.
- `by_vars` By variables
  - If the argument is specified, for each by group the observations are selected separately.
derive_param_extreme_record

**order**  Sort order

If the argument is set to a non-null value, for each by group the first or last observation from the source datasets is selected with respect to the specified order. Variables created via `new_vars` e.g., imputed date variables, can be specified as well (see examples below).

Please note that `NA` is considered as the last value. I.e., if a order variable is `NA` and `mode = "last"`, this observation is chosen while for `mode = "first"` the observation is chosen only if there are no observations where the variable is not `NA`.

*Permitted Values:* list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))`

**mode**  Selection mode (first or last)

If "first" is specified, for each by group the first observation with respect to `order` is included in the output dataset. If "last" is specified, the last observation is included in the output dataset.

*Permitted Values:* "first", "last"

**set_values_to**  Variables to be set

The specified variables are set to the specified values for the new observations. A list of variable name-value pairs is expected.

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or `NA`, e.g., `exprs(PARAMCD = "PD", PARAM = "First Progressive Disease")`.

**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.
2. Variables specified by `new_vars` are created for each source dataset.
3. The first or last observation (with respect to the `order` variable) for each by group (specified by `by_vars`) from multiple sources is selected and added to the input dataset.

**Value**

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

**See Also**

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`
Examples

```r
eaevent_samp <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~RSSTDTC,
  "1", "PD", "First Progressive Disease", "2022-04-01",
  "2", "PD", "First Progressive Disease", "2021-04-01",
  "3", "PD", "First Progressive Disease", "2023-04-01"
)

cm <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~CMDECOD, ~CMSTDTC,
  "1001", "1", "ACT", "2021-12-25"
)

pr <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~PRDECOD, ~PRSTDTC,
  "1001", "1", "ACS", "2021-12-27",
  "1001", "2", "ACS", "2020-12-25",
  "1001", "3", "ACS", "2022-12-25"
)

derive_param_extreme_record(
  dataset = aevent_samp,
  sources = list(
    records_source(
      dataset_name = "cm",
      filter = CMDECOD == "ACT",
      new_vars = exprs(
        ADT = convert_dtc_to_dt(CMSTDTC),
        AVALC = CMDECOD
      )
    ),
    records_source(
      dataset_name = "pr",
      filter = PRDECOD == "ACS",
      new_vars = exprs(
        ADT = convert_dtc_to_dt(PRSTDTC),
        AVALC = PRDECOD
      )
    )
  ),
  source_datasets = list(cm = cm, pr = pr),
  by_vars = exprs(USUBJID),
  order = exprs(ADT),
  mode = "first",
  set_values_to = exprs(
    PARAMCD = "FIRSTACT",
    PARAM = "First Anti-Cancer Therapy"
  )
)
)
derive_param_framingham

Adds a Parameter for Framingham Heart Study Cardiovascular Disease 10-Year Risk Score

Description

Adds a record for framingham score (FCVD101) for each by group (e.g., subject and visit) where
the source parameters are available.

Usage

```r
derive_param_framingham(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "FCVD101"),
  sysbp_code = "SYSBP",
  chol_code = "CHOL",
  cholhdl_code = "CHOLHDL",
  age = AGE,
  sex = SEX,
  smokefl = SMOKEFL,
  diabetfl = DIABETFL,
  trthypfl = TRTHYPFL,
  get_unit_expr,
  filter = NULL
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input dataset</td>
</tr>
<tr>
<td>by_vars</td>
<td>Grouping variables</td>
</tr>
<tr>
<td>set_values_to</td>
<td>Variables to be set</td>
</tr>
<tr>
<td>sysbp_code</td>
<td>The variables specified by the by_vars parameter, PARAMCD, and AVAL are ex-</td>
</tr>
<tr>
<td>chol_code</td>
<td>pected.</td>
</tr>
<tr>
<td>cholhdl_code</td>
<td>The variable specified by by_vars and PARAMCD must be a unique key of the</td>
</tr>
<tr>
<td>age</td>
<td>input dataset after restricting it by the filter condition (filter parameter)</td>
</tr>
<tr>
<td>sex</td>
<td>and to the parameters specified by sysbp_code, chol_code and hdl_code.</td>
</tr>
<tr>
<td>smokefl</td>
<td>Only variables specified in by_vars will be populated in the newly created</td>
</tr>
<tr>
<td>diabetfl</td>
<td>records.</td>
</tr>
<tr>
<td>trthypfl</td>
<td>Permitted Values: list of variables</td>
</tr>
<tr>
<td>get_unit_expr</td>
<td>The specified variables are set to the specified values for the new observa-</td>
</tr>
<tr>
<td>filter</td>
<td>tions.</td>
</tr>
</tbody>
</table>

Permitted Values: List of variable-value pairs
derive_param_framingham

**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

**chol_code**  
Total serum cholesterol code  
The observations where PARAMCD equals the specified value are considered as  
the total cholesterol assessments. This must be measured in mg/dL.  
*Permitted Values:* character value

**cholhdl_code**  
HDL serum cholesterol code  
The observations where PARAMCD equals the specified value are considered as  
the HDL cholesterol assessments. This must be measured in mg/dL.  
*Permitted Values:* character value

**age**  
Subject age  
A variable containing the subject’s age.  
*Permitted Values:* A numeric variable name that refers to a subject age column  
of the input dataset

**sex**  
Subject sex  
A variable containing the subject’s sex.  
*Permitted Values:* A character variable name that refers to a subject sex column  
of the input dataset

**smokefl**  
Smoking status flag  
A flag indicating smoking status.  
*Permitted Values:* A character variable name that refers to a smoking status  
column of the input dataset

**diabetfl**  
Diabetic flag  
A flag indicating diabetic status.  
*Permitted Values:* A character variable name that refers to a diabetic status col-  
umn of the input dataset

**trthypfl**  
Treated with hypertension medication flag  
A flag indicating if a subject was treated with hypertension medication.  
*Permitted Values:* A character variable name that refers to a column that indi-  
cates whether a subject is treated for high blood pressure

**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 values of age, sex, smokefl, diabetfl and trthypfl will be added to the by_vars list. The predicted probability of having cardiovascular disease (CVD) within 10-years according to Framingham formula. See AHA Journal article General Cardiovascular Risk Profile for Use in Primary Care for reference.

For Women:

<table>
<thead>
<tr>
<th>Factor</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>2.32888</td>
</tr>
<tr>
<td>Total Chol</td>
<td>1.20904</td>
</tr>
<tr>
<td>HDL Chol</td>
<td>-0.70833</td>
</tr>
<tr>
<td>Sys BP</td>
<td>2.76157</td>
</tr>
<tr>
<td>Sys BP + Hypertension Meds</td>
<td>2.82263</td>
</tr>
<tr>
<td>Smoker</td>
<td>0.52873</td>
</tr>
<tr>
<td>Non-Smoker</td>
<td>0</td>
</tr>
<tr>
<td>Diabetic</td>
<td>0.69154</td>
</tr>
<tr>
<td>Not Diabetic</td>
<td>0</td>
</tr>
<tr>
<td>Average Risk</td>
<td>26.1931</td>
</tr>
<tr>
<td>Risk Period</td>
<td>0.95012</td>
</tr>
</tbody>
</table>

For Men:

<table>
<thead>
<tr>
<th>Factor</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>3.06117</td>
</tr>
<tr>
<td>Total Chol</td>
<td>1.12370</td>
</tr>
<tr>
<td>HDL Chol</td>
<td>-0.93263</td>
</tr>
<tr>
<td>Sys BP</td>
<td>1.93303</td>
</tr>
<tr>
<td>Sys BP + Hypertension Meds</td>
<td>2.99881</td>
</tr>
<tr>
<td>Smoker</td>
<td>.65451</td>
</tr>
<tr>
<td>Non-Smoker</td>
<td>0</td>
</tr>
<tr>
<td>Diabetic</td>
<td>0.57367</td>
</tr>
<tr>
<td>Not Diabetic</td>
<td>0</td>
</tr>
<tr>
<td>Average Risk</td>
<td>23.9802</td>
</tr>
<tr>
<td>Risk Period</td>
<td>0.88936</td>
</tr>
</tbody>
</table>

The equation for calculating risk:

\[ RiskFactors = (\log(Age) \times AgeFactor) + (\log(TotalChol) \times TotalCholFactor) + (\log(CholHDL) \times CholHDLFactor) \]

\[ Risk = 100 \times (1 - RiskPeriodFactor^{RiskFactors}) \]

Value

The input dataset with the new parameter added
See Also

compute_framingham()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_extreme_record(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

library(tibble)

adcvrisk <- tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU,
  ~VISIT, ~AGE, ~SEX, ~SMOKEFL, ~DIABETFL, ~TRTHYPFL,
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "mmHg", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 115, "mmHg", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 216.16, "mg/dL", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 210.78, "mg/dL", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 101, "mmHg", "BASELINE", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 101, "mmHg", "WEEK 2", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOL", "Total Cholesterol (mg/dL)", 292.01, "mg/dL", "BASELINE", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOL", "Total Cholesterol (mg/dL)", 246.73, "mg/dL", "WEEK 2", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 44.62, "mg/dL", "WEEK 2", 55, "M", "Y", "Y", "Y"
)

adcvrisk %>%
  derive_param_framingham(
    by_vars = exprs(USUBJID, VISIT),
    set_values_to = exprs(
      PARAMCD = "FCVD101",
      PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
    ),
    get_unit_expr = AVALU
  )
derive_param_framingham(
  adcvrisk,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "FCVD101",
    PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
  ),
  get_unit_expr = extract_unit(PARAM)
)

derive_param_map  

**Description**

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

**Note:** This is a wrapper function for the more generic derive_param_computed().

**Usage**

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

**Arguments**

- **dataset**  
  Input dataset

- **by_vars**  
  Grouping variables

  For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records.

  **Permitted Values:** list of variables

  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`.

- **set_values_to**  
  
  The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected.
**derive_param_map**

**set_values_to** Variables to be set
The specified variables are set to the specified values for the new observations.
For example `exprs(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

\[
\frac{2\text{DIABP} + \text{SYSBP}}{3}
\]

if it is based on diastolic and systolic blood pressure and

\[
\text{DIABP} + 0.01e^{4.14 - \frac{40.74}{\pi}}(\text{SYSBP} - \text{DIABP})
\]

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

**Value**

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

See Also

compute_map()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

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

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

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

# Derive MAP based on diastolic and systolic blood pressure and heart rate
derive_param_map(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  hr_code = "PULSE",
  set_values_to = exprs(
    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)"
  ),
  get_unit_expr = extract_unit(PARAM)
derive_param_qtc

Adds a Parameter for Corrected QT (an ECG measurement)

Description

Adds a record for corrected QT using either Bazett’s, Fridericia’s or Sagie’s formula for each by

group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic derive_param_computed().

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
  Only variables specified in by_vars will be populated in the newly created
  records.
  Permitted Values: list of variables

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

set_values_to Variables to be set
  The specified variables are set to the specified values for the new observations.
  For example exprs(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. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

**See Also**
compute_qtc()  
compute_qtc()

BDS-Findings Functions for adding Parameters/Records:  
default_qtc_paramcd(), derive_expected_records(),  
derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(),  
derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(),  
derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(),  
derive_param_map(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()  

**Examples**

```r  
library(tibble)  
adeg <- 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",  
)  ```
derive_param_qtc(  
adeg,  
  by_vars = exprs(USUBJID, VISIT),  
  method = "Bazett",  
  set_values_to = exprs(  
    PARAMCD = "QTCBR",  
    PARAM = "QTcB - Bazett's Correction Formula Rederived (msec)",  
    AVALU = "msec"  
  ),  
  get_unit_expr = AVALU  
)

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

derive_param_qtc(  
adeg,  
  by_vars = exprs(USUBJID, VISIT),  
  method = "Sagie",  
  set_values_to = exprs(  
    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 (an ECG measurement)**

**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.

**Note:** This is a wrapper function for the more generic `derive_param_computed()`.  
The analysis value of the new parameter is derived as

\[
\frac{60000}{HR}
\]
Usage

derive_param_rr(
    dataset,
    by_vars,
    set_values_to = exprs(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. Only variables specified in by_vars will be populated in the newly created records.
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 exprs(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

Value

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

**Derive a Time-to-Event Parameter**

**Description**

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

**Usage**

```r
derive_param_tte(
    dataset = NULL,
    dataset_adsl,
)```

**See Also**

- `compute_rr()`
- BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_wbc_abs()`, `derive_summary_records()`
source_datasets,
by_vars = NULL,
start_date = TRTSDT,
event_conditions,
censor_conditions,
create_datetime = FALSE,
set_values_to,
subject_keys = get_admiral_option("subject_keys")
)

Arguments

dataset Input dataset
  The PARAMCD variable is expected.
dataset_ads1 ADSL input dataset
  The variables specified for start_date, 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.
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 exprs() defining the variables to be set for the new parameter, e.g. exprs(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 exprs() 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 date part is copied.
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 date part is copied.
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 variable specified for start_date is joined from the ADSL dataset. Only subjects in both datasets are kept, i.e., subjects with both an event or censoring and an observation in dataset_adsl.
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).
4. The new observations are added to the output dataset.

**Value**

The input dataset with the new parameter added
See Also
event_source(), censor_source()

Examples

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

adsl <- admiral_adsl

# derive overall survival parameter
death <- event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = exprs(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)

last_alive_dt <- censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = exprs(
    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 = exprs(
    PARAMCD = "OS",
    PARAM = "Overall Survival"
  )
)
%
select(-STUDYID) %>%
  filter(row_number() %in% 20:30)

# derive duration of response
# only observations for subjects in dataset_adsl are created
adsl <- tribble(
  ~USUBJID, ~DTHFL, ~DTHDT, ~RSPDT, "01", "Y", ymd("2021-06-12"), ymd("2021-03-04"),
  ~USUBJID, ~DTHFL, ~DTHDT, ~RSPDT, "02", "N", ymd("2021-06-12"), ymd("2021-03-04"),
  ~USUBJID, ~DTHFL, ~DTHDT, ~RSPDT, "03", "Y", ymd("2021-06-12"), ymd("2021-03-04"),
  ~USUBJID, ~DTHFL, ~DTHDT, ~RSPDT, "04", "Y", ymd("2021-06-12"), ymd("2021-03-04"))
`derive_param_tte`

```r

# Mutate STUDYID to "AB42"
mutate(STUDYID = "AB42")

adrs <- tribble(~USUBJID, ~AVALC, ~ADT, ~ASEQ,
                 "01", "SD", ymd("2021-01-03"), 1,
                 "02", "PR", ymd("2021-03-04"), 2,
                 "03", "PD", ymd("2021-05-05"), 3,
                 "04", "CR", ymd("2021-05-15"), 3)

# Mutate STUDYID to "AB42", PARAMCD = "OVR"
mutate(STUDYID = "AB42", PARAMCD = "OVR")

pd <- event_source(
dataset_name = "adrs",
filter = AVALC == "PD",
date = ADT,
set_values_to = exprs(
  EVENTDESC = "PD",
  SRCDOM = "ADRS",
  SRCVAR = "ADTM",
  SRCSEQ = ASEQ
)
)

dead <- event_source(
dataset_name = "adrs",
filter = DTHFL == "Y",
date = DTHDT,
set_values_to = exprs(
  EVENTDESC = "DEATH",
  SRCDOM = "ADRS",
  SRCVAR = "ADTM",
  SRCSEQ = "ASEQ"
)
)

lastvisit <- censor_source(
dataset_name = "adrs",
date = ADT,
censor = 1,
set_values_to = exprs(
  EVENTDESC = "LAST TUMOR ASSESSMENT",
  SRCDOM = "ADRS",
  SRCVAR = "ADTM",
  SRCSEQ = "ASEQ"
)
)
```

first_response <- censor_source(
  dataset_name = "adsl",
  date = RSPDT,
  censor = 1,
  set_values_to = exprs(
    EVENTDESC = "FIRST RESPONSE",
    SRCDOM = "ADSL",
    SRCVAR = "RSPDT"
  )
)

derive_param_tte(
  dataset_adsl = filter(adsl, !is.na(RSPDT)),
  start_date = RSPDT,
  event_conditions = list(pd, death),
  censor_conditions = list(lastvisit, first_response),
  source_datasets = list(adsl = adsl, adrs = adrs),
  set_values_to = exprs(
    PARAMCD = "DURRSP",
    PARAM = "Duration of Response"
  )
)

# derive time to adverse event for each preferred term
adsl <- 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 <- tribble(~USUBJID, ~AESTDTC, ~ASEQ, ~AEDECOD,
              "01", "2021-01-03T10:56", 1, "Flu",
              "01", "2021-03-04", 2, "Cough",
              "01", "2021", 3, "Flu")  
  %>%
  mutate(STUDYID = "AB42")

ae_ext <- derive_vars_dt(
  ae,
  dtc = AESTDTC,
  new_vars_prefix = "AEST",
  highest_imputation = "M",
  flag_imputation = "none"
  )

ttae <- event_source(
  dataset_name = "ae",
  date = AESTDT,
  set_values_to = exprs(
    EVENTDESC = "AE",
    SRCDOM = "AE"
```r
SRCVAR = "AESTDTC",
SRCSEQ = AESEQ
)
)

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

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

---

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

**Description**

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

**Usage**

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

Arguments

**dataset**
- Input dataset
- The variables specified by the `by_vars` argument, `PARAMCD`, and `AVAL` are expected to be present.
- The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset, and to the parameters specified by `wbc_code` and `diff_code`.

**by_vars**
- Grouping variables
- Permitted Values: list of variables

**set_values_to**
- Variables to set
- A named list returned by `exprs()` defining the variables to be set for the new parameter, e.g. `exprs(PARAMCD = "LYMPH", PARAM = "Lymphocytes Abs (10^9/L)")` is expected.

**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 containing unit from the input dataset, or a function call, for example, `get_unit_expr = extract_unit(PARAM)`.

**wbc_unit**
- A string containing the required unit of the WBC parameter
- Default: "10^9/L"

**wbc_code**
- White Blood Cell (WBC) parameter
- The observations where `PARAMCD` equals the specified value are considered as the WBC absolute results to use for converting the differentials.
- Default: "WBC"
- Permitted Values: character value

**diff_code**
- White blood differential parameter
- The observations where `PARAMCD` equals the specified value are considered as the white blood differential lab results in fraction or percentage value to be converted into absolute value.

**diff_type**
- A string specifying the type of differential
- Permitted Values: "percent", "fraction"
- Default: fraction

Details

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

\[
\text{WhiteBloodCellCount} \times \frac{\text{PercentageValue}}{100}
\]

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

\[
\text{WhiteBloodCellCount} \times \text{FractionValue}
\]

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

The input dataset with the new parameter added

See Also

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_extreme_record()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_summary_records()`

Examples

```r
library(tibble)

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

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

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,  # Variables to consider for generation of groupwise summary records. Providing the names of variables in exprs() will create a groupwise summary and generate summary records for the specified groups.
  filter = NULL,  # 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.
  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 = NULL
)

arguments

dataset  # A data frame.
by_vars  # Variables to consider for generation of groupwise summary records. Providing the names of variables in exprs() 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  # Variables to be set
  The specified variables are set to the specified values for the new observations.
  A list of variable name-value pairs is expected.
  • LHS refers to a variable.
  • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression, or NA, e.g., exprs(PARAMCD = "TDose", PARCAT1 = "OVERALL").

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.

See Also

get_summary_records()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(),
derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(),
derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(),
derive_param_exposure(), derive_param_extreme_record(), derive_param_framingham(),
derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs()

Examples

library(tibble)
library(dplyr, warn.conflicts = TRUE)

deg <- 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(
  deg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = exprs(DTYPE = "AVERAGE")
)

devs <- tribble(
  ~USUBJID, ~VSSEQ, ~PARAM, ~AVAL, ~VSSTRESU, ~VISIT, ~VSDTC,
)
derive_summary_records

"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 = exprs(USUBJID, PARAM),
  analysis_var = AVAL,
  summary_fun = max,
  set_values_to = exprs(DTYPE = "MAXIMUM")
)

derive_summary_records(
  by_vars = exprs(USUBJID, PARAM),
  analysis_var = AVAL,
  summary_fun = mean,
  set_values_to = exprs(DTYPE = "AVERAGE")
)

# Sample ADEG dataset with triplicate record for only AVISIT = 'Baseline'
adeg <- 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-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 = exprs(USUBJID, PARAM, AVISIT),
  filter = n() > 2,
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = exprs(DTYPE = "AVERAGE")
)
derive_vars_aage

Derive Analysis Age

Description

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

Note: This is a wrapper function for the more generic derive_vars_duration().

Usage

derive_vars_aage(
  dataset,
  start_date = BRTHDT,
  end_date = RANDDT,
  unit = "years",
  age_unit = "years",
  type = "interval"
)

Arguments

dataset  Input dataset
  The columns specified by the start_date and the end_date parameter are ex-
  pected.

start_date  The start date
  A date or date-time object is expected.
  Refer to derive_vars_dt() to impute and derive a date from a date character
  vector to a date object.
  Default: BRTHDT

end_date  The end date
  A date or date-time object is expected.
  Refer to derive_vars_dt() to impute and derive a date from a date character
  vector to a date object.
  Default: RANDDT

unit  Deprecated, please use age_unit instead.

age_unit  Age unit
  The age is derived in the specified unit
  Default: 'years'
  Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'

type  lubridate duration type.
  See below for details.
  Permitted Values: "duration", "interval"
Details

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

The `lubridate` package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The interval between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The duration between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the `lubridate time span reference page`.

Value

The input dataset with AAGE and AAGEU added

See Also

- `derive_vars_duration()`

ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_var_extreme_dt()`, `derive_vars_period()`

Examples

```r
library(tibble)
library(lubridate)

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

derive_vars_aage(data)
```

---

**Derive ATC Class Variables**

**Description**

Add Anatomical Therapeutic Chemical class variables from FACM to ADCM.

**Note:** This is a wrapper function for the more generic `derive_vars_transposed()`.
derive_vars_atc

Usage

derive_vars_atc(
  dataset,
  dataset_facm,
  by_vars = exprs(USUBJID, CMREFID = FAREFID),
  value_var = FASTRESC
)

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 and value_var parameters, FAGRPID
and FATESTCD are required

by_vars          Keys used to merge dataset_facm with dataset

Permitted Values: list of variables

value_var        The variable of dataset_facm containing the values of the transposed variables

Default: FASTRESC

Value

The input dataset with ATC variables added

See Also

OCCDS Functions: derive_var_trtemfl(), derive_vars_query(), get_terms_from_db()

Examples

library(tibble)

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

cm_facm <- 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",
)
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 argument) and the date imputation flag ("--DTF") can be added.

Usage

```r
derive_vars_dt(
  dataset,
  new_vars_prefix,
  dtc,
  highest_imputation = "n",
  date_imputation = "first",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

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

- **new_vars_prefix**
  Prefix used for the output variable(s).
  A character scalar is expected. For the date variable "DT" is appended to the specified prefix and for the date imputation flag "DTF". I.e., for `new_vars_prefix = "AST"` the variables ASTDT and ASTDTF are created.
The '---DTC' date to impute
A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. Trailing components can be omitted and - is a valid "missing" value for any component.

highest_imputation
Highest imputation level
The highest_imputation argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.
If a component at a higher level than the highest imputation level is missing, NA_character_ is returned. For example, for highest_imputation = "D" "2020" results in NA_character_ because the month is missing.
If "n" is specified no imputation is performed, i.e., if any component is missing, NA_character_ is returned.
If "Y" is specified, date_imputation should be "first" or "last" and min_dates or max_dates should be specified respectively. Otherwise, NA_character_ is returned if the year component is missing.
Default: "n"
Permitted Values: "Y" (year, highest level), "M" (month), "D" (day), "n" (none, lowest level)

date_imputation
The value to impute the day/month when a datepart is missing.
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 (The year can not be specified; for imputing the year "first" or "last" together with min_dates or max_dates argument can be used (see examples).),
- or as a keyword: "first", "mid", "last" to impute to the first/mid/last day/month.
The argument is ignored if highest_imputation is less then "D".
Default: "first"

flag_imputation
Whether the date imputation flag must also be derived.
If "auto" is specified, the date imputation flag is derived if the date_imputation argument is not null.
Default: "auto"
Permitted Values: "auto", "date" or "none"

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

impute_dtc_dtm(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11"),
  ),
  highest_imputation = "M"
)

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

max_dates

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

preserve

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

Details

In {admiral} we don’t allow users to pick any single part of the date/time to impute, we only enable to impute up to a highest level, i.e. you couldn’t choose to say impute months, but not days.
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.

See Also

Date/Time Derivation Functions that returns variable appended to dataset: derive_var_trtdurd(), derive_vars_dtm_to_dt(), derive_vars_dtm_to_tm(), derive_vars_dtm(), derive_vars_duration(), derive_vars_dy()

Examples

library(tibble)
library(lubridate)

mhdt <- tribble(
  ~MHSTDTC,  
  "2019-07-18T15:25:40",
)
derive_vars_dt
"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,
    highest_imputation = "M"
)

# Impute partial dates to 6th of April
derive_vars_dt(
    mhdt,
    new_vars_prefix = "AST",
    dtc = MHSTDTC,
    highest_imputation = "M",
    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,
    highest_imputation = "M",
    date_imputation = "last"
)

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

Derive/Impute a Datetime from a Date Character Vector

Description

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

Usage

```
derive_vars_dtm(
  dataset,
  new_vars_prefix,
  dtc,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
)
```
derive_vars_dtm

flag_imputation = "auto",
min_dates = NULL,
max_dates = NULL,
preserve = FALSE,
ignore_seconds_flag = FALSE
)

Arguments

dataset
Input dataset

The date character vector (dtc) must be present.

new_vars_prefix
Prefix used for the output variable(s).

A character scalar is expected. For the date variable "DT" is appended to the
specified prefix, for the date imputation flag "DTF", and for the time imputation
flag "TMF". I.e., for new_vars_prefix = "AST" the variables ASTDT, ASTDTF,
and ASTTMF are created.

dtc
The '---DTC' date to impute

A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss.
Trailing components can be omitted and - is a valid "missing" value for any
component.

highest_imputation
Highest imputation level

The highest_imputation argument controls which components of the DTC
value are imputed if they are missing. All components up to the specified level
are imputed.

If a component at a higher level than the highest imputation level is miss-
ing, NA_character_ is returned. For example, for highest_imputation = "D"
"2020" results in NA_character_ because the month is missing.

If "n" is specified, no imputation is performed, i.e., if any component is missing,
NA_character_ is returned.

If "Y" is specified, date_imputation should be "first" or "last" and min_dates
or max_dates should be specified respectively. Otherwise, NA_character_ is
returned if the year component is missing.

Default: "h"

Permitted Values: "Y" (year, highest level), "M" (month), "D" (day), "h" (hour),
"m" (minute), "s" (second), "n" (none, lowest level)

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

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 (The year can not be specified; for imputing the year "first" or
  "last" together with min_dates or max_dates argument can be used (see
  examples).),
• or as a keyword: "first", "mid", "last" to impute to the first/mid/last
day/month.
derive_vars_dtm

The argument is ignored if highest_imputation is less than "D".
Default: "first".

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.
The argument is ignored if highest_imputation = "n".
Default: "first".

flag_imputation
Whether the date/time imputation flag(s) must also be derived.
If "auto" is specified, the date imputation flag is derived if the date_imputation argument is not null and the time imputation flag is derived if the time_imputation argument is not null
Default: "auto"
Permitted Values: "auto", "date", "time", "both", or "none"

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

```r
impute_dtc_dtm(  
"2020-11",  
min_dates = list(  
  ymd_hms("2020-12-06T12:12:12"),  
  ymd_hms("2020-11-11T11:11:11")  
),  
  highest_imputation = "M"  
)
```
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).
For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

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

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

**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**

In `admiral` we don't allow users to pick any single part of the date/time to impute, we only enable to impute up to a highest level, i.e. you couldn't choose to say impute months, but not days.

The presence of a 'DTC' 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.

**See Also**

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_dt()`, `derive_vars_duration()`, `derive_vars_dy()`

**Examples**

```r
library(tibble)
library(lubridate)

```
derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M"
)

# 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 <- 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",
  highest_imputation = "M",
  date_imputation = "last",
  time_imputation = "last",
  max_dates = exprs(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 <- tribble(~MHSTDTC,
  "2019-07-18T15:25",
  "2019-07-18T15:25",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  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(
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 exprs() 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.

See Also
Date/Time Derivation Functions that returns variable appended to dataset: derive_var_trtdurd(), derive_vars_dtm_to_tm(), derive_vars_dtm(), derive_vars_dt(), derive_vars_duration(), derive_vars_dy()

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

adcm <- 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", "2017-02-25 13:00:00",
"PAT01", "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-04-29 14:00:00",
)
Derive Time Variables from Datetime Variables

**Description**

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

**Usage**

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

**Arguments**

- `dataset` : Input dataset
- `source_vars` : A list of datetime variables created using `exprs()` 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.

**See Also**

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm()`, `derive_vars_dt()`, `derive_vars_duration()`, `derive_vars_dy()`
Examples

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

adcm <- tribble(
  ~USUBJID, ~TRTSDTM, ~ASTDTM, ~AENDTM,
  "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",
) %>%
mutable(
  TRTSDTM = as_datetime(TRTSDTM),
  ASTDTM = as_datetime(ASTDTM),
  AENDTM = as_datetime(AENDTM)
)

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

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

---

derive vars duration Derive Duration

Description

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

Usage

derive vars duration(
  dataset,
  new var,
  new var unit = NULL,
  start date,
  end date,
  in unit = "days",
  out unit = "days",
  floor in = TRUE,
  add one = TRUE,
  trunc out = FALSE,
  type = "duration"
)
Arguments

- **dataset**: Input dataset
  The variables specified by the `start_date` and the `end_date` parameter are expected.

- **new_var**: Name of variable to create

- **new_var_unit**: Name of the unit variable
  If the parameter is not specified, no variable for the unit is created.

- **start_date**: The start date
  A date or date-time variable is expected. This variable must be present in specified input dataset.
  Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

- **end_date**: The end date
  A date or date-time variable is expected. This variable must be present in specified input dataset.
  Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

- **in_unit**: Input unit
  See `floor_in` and `add_one` parameter for details.
  Default: 'days'
  Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'min', 'seconds', 'sec'

- **out_unit**: Output unit
  The duration is derived in the specified unit
  Default: 'days'
  Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'min', 'seconds', 'sec'

- **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

- **type**: lubridate duration type.
  See below for details.
  Permitted Values: "duration", "interval"
The duration is derived as time from start to end date in the specified output unit. If the end date is before the start date, the duration is negative. The start and end date variable must be present in the specified input dataset.

The `lubridate` package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The `interval` between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The `duration` between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the `lubridate time span reference page`.

The input dataset with the duration and unit variable added

See Also

- `compute_duration()`
- Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trttdurd()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_dtm()`, `derive_vars_dt()`, `derive_vars_dy()`

Examples

```r
library(lubridate)
library(tibble)

# Derive age in years
data <- tribble(
  ~USUBJID, ~BRTHDT, ~RANDDT,
  "P01", ymd("1984-09-06"), ymd("2020-02-24"),
  "P02", ymd("1985-01-01"), NA,
  "P03", NA, ymd("2021-03-10"),
  "P04", NA, NA
)

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 adverse event duration in days

data <- tribble(
  ~USUBJID, ~ASTDT, ~AENDT,
  "P01", ymd("2021-03-05"), ymd("2021-03-02"),
  "P02", ymd("2019-09-18"), ymd("2019-09-18"),
  "P03", ymd("1985-01-01"), NA,
  "P04", NA, NA
)

derive_vars_duration(data, 
  new_var = ADURN, 
  new_var_unit = ADURU, 
  start_date = ASTDT, 
  end_date = AENDT, 
  out_unit = "days"
)

# Derive adverse event duration in minutes

data <- tribble(
  ~USUBJID, ~ADTM, ~TRTSDTM,
  "P01", ymd_hms("2019-08-09T04:30:56"), ymd_hms("2019-08-09T05:00:00"),
  "P02", ymd_hms("2019-11-11T10:30:00"), ymd_hms("2019-11-11T11:30:00"),
  "P03", ymd_hms("2019-11-11T00:00:00"), ymd_hms("2019-11-11T00:00:00"),
  "P04", NA, ymd_hms("2019-11-11T12:34:56"),
  "P05", NA, ymd_hms("2019-11-11T12:34:56"),
)

derive_vars_duration(data, 
  new_var = ADURN, 
  new_var_unit = ADURU, 
  start_date = ADTM, 
  end_date = TRTSDTM, 
  in_unit = "minutes", 
  out_unit = "minutes", 
  add_one = FALSE
)

# Derive adverse event start time since last dose in hours

data <- tribble(
  ~USUBJID, ~ASTDTM, ~LDOSEDTM,
  "P01", ymd_hms("2019-08-09T04:30:56"), ymd_hms("2019-08-08T10:05:00"),
  "P02", ymd_hms("2019-11-11T10:30:00"), ymd_hms("2019-11-11T11:30:00"),
  "P03", ymd_hms("2019-11-11T00:00:00"), ymd_hms("2019-11-10T23:59:59"),
  "P04", ymd_hms("2019-11-11T12:34:56"), NA,
  "P05", NA, ymd_hms("2019-09-28T12:34:56"),
)

derive_vars_duration(
  data, 
  new_var = LDRELTM, 
  new_var_unit = LDRELTMU, 
  start_date = LDOSEDTM, 
  end_date = ASTDTM, 
  in_unit = "hours"
Derive Relative Day Variables

Description

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

Usage

\[
\text{derive_vars_dy}(\text{dataset, reference_date, source_vars})
\]

Arguments

- **dataset**: Input dataset
  - The columns specified by the `reference_date` and the `source_vars` parameter are expected.
- **reference_date**: A date or date-time column, e.g., date of first treatment or date-time of last exposure to treatment.
  - Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
- **source_vars**: A list of datetime or date variables created using `exprs()` 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. `exprs(TRTSDTM, ASTDTM, AENDT)` or `exprs(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

See Also

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_dtm()`, `derive_vars_dt()`, `derive_vars_duration()`
Examples

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

datain <- tribble(
  ~TRTSDTM, ~ASTDTM, ~AENDT,
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDT = ymd(AENDT)
  )

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

# specifying name of new variables

datain <- 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 = exprs(TRTSDT, DEATHDY = DTHDT)
)

---

derive_vars Joined  Add Variables from an Additional Dataset Based on Conditions from Both Datasets

Description

The function adds variables from an additional dataset to the input dataset. The selection of the observations from the additional dataset can depend on variables from both datasets. For example, add the lowest value (nadir) before the current observation.
Usage

derive_vars_joined(
    dataset,
    dataset_add,
    by_vars = NULL,
    order = NULL,
    new_vars = NULL,
    join_vars = NULL,
    filter_add = NULL,
    filter_join = NULL,
    mode = NULL,
    missing_values = NULL,
    check_type = "warning"
)

Arguments

dataset        Input dataset
    The variables specified by by_vars are expected.
dataset_add    Additional dataset
    The variables specified by the by_vars, the new_vars, the join_vars, and the
    order argument are expected.
by_vars        Grouping variables
    The two datasets are joined by the specified variables. Variables from the addi-
    tional dataset can be renamed by naming the element, i.e., by_vars = exprs(<name in input dataset>
    Permitted Values: list of variables created by exprs()
order          Sort order
    If the argument is set to a non-null value, for each observation of the input
    dataset the first or last observation from the joined dataset is selected with re-
    spect to the specified order. The specified variables are expected in the addi-
    tional dataset (dataset_add). If a variable is available in both dataset and
    dataset_add, the one from dataset_add is used for the sorting.
    If an expression is named, e.g., exprs(EXSTD = convert_dtc_to_dt(EXSTDTC, 
    EXSEQ), a corresponding variable (EXSTD) is added to the additional dataset
    and can be used in the filter conditions (filter_add, filter_join) and for
    join_vars and new_vars. The variable is not included in the output dataset.
    Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, 
    desc(AVAL)) or NULL
new_vars       Variables to add
    The specified variables from the additional dataset are added to the output dataset.
    Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>).
    For example new_vars = exprs(var1, var2) adds variables var1 and var2
    from dataset_add to the input dataset.
    And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2
    from dataset_add and adds them to the input dataset renaming old_var2 to
    new_var2.
Values of the added variables can be modified by specifying an expression. For example, `new_vars = LASTRSP = exprs(str_to_upper(AVALC))` adds the variable LASTRSP to the dataset and sets it to the upper case value of AVALC.

If the argument is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added.

**Permitted Values:** list of variables or named expressions created by `exprs()`

**join_vars**

Variables to use from additional dataset

Any extra variables required from the additional dataset for filter_join should be specified for this argument. Variables specified for new_vars do not need to be repeated for join_vars. If a specified variable exists in both the input dataset and the additional dataset, the suffix ".join" is added to the variable from the additional dataset.

If an expression is named, e.g., `exprs(EXTDT = convert_dtc_to_dt(EXSTDTC))`, a corresponding variable is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for new_vars. The variable is not included in the output dataset.

The variables are not included in the output dataset.

**Permitted Values:** list of variables or named expressions created by `exprs()`

**filter_add**

Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by order or new_vars arguments can be used in the condition.

**Permitted Values:** a condition

**filter_join**

Filter for the joined dataset

The specified condition is applied to the joined dataset. Therefore variables from both datasets dataset and dataset_add can be used.

Variables created by order or new_vars arguments can be used in the condition.

**Permitted Values:** a condition

**mode**

Selection mode

Determines if the first or last observation is selected. If the order argument is specified, mode must be non-null.

If the order argument is not specified, the mode argument is ignored.

**Permitted Values:** "first", "last", NULL

**missing_values**

Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

**Permitted Values:** named list of expressions, e.g., `exprs(BASEC = "MISSING", BASE = -1)`

**check_type**

Check uniqueness?

If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) joined dataset are not unique with respect to the by variables and the order.
This argument is ignored if `order` is not specified. In this case an error is issued independent of `check_type` if the restricted joined dataset contains more than one observation for any of the observations of the input dataset.

Permitted Values: "none", "warning", "error"

Details

1. The variables specified by `order` are added to the additional dataset (`dataset_add`).
2. The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
3. The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.
4. The input dataset and the (restricted) additional dataset are left joined by the grouping variables (by_vars). If no grouping variables are specified, a full join is performed.
5. The joined dataset is restricted by the `filter_join` condition.
6. If `order` is specified, for each observation of the input dataset the first or last observation (depending on `mode`) is selected.
7. The variables specified for `new_vars` are created (if requested) and merged to the input dataset. I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the joined dataset the new variables are set as specified by `missing_values` (or to NA for variables not in `missing_values`). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_exist_flag()`, `derive_var_merged_summary()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `derive_vars_transposed()`, `get_summary_records()`

Examples

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

# Add AVISIT (based on time windows), AWLO, and AWHI
addbs <- tribble(
  ~USUBJID, ~ADY,
  "1",    -33,
  "1",    -2,
  "1",     3,
  "1",    24,
```
```
# derive_vars_joined

windows <- tribble(
  ~AVISIT, ~AWLO, ~AWHI,
  "BASELINE", -30, 1,
  "WEEK 1", 2, 7,
  "WEEK 2", 8, 15,
  "WEEK 3", 16, 22,
  "WEEK 4", 23, 30
)

derive_vars_joined(
  adbds,
  dataset_add = windows,
  filter_join = AWLO <= ADY & ADY <= AWHI
)

# derive the nadir after baseline and before the current observation
adbds <- tribble(
  ~USUBJID, ~ADY, ~AVAL,
  "1", -7, 70,
  "1", 1, 12,
  "1", 8, 11,
  "1", 15, 9,
  "1", 20, 14,
  "1", 24, 12,
  "2", 13, 8
)

derive_vars_joined(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(USUBJID),
  order = exprs(AVAL),
  new_vars = exprs(NADIR = AVAL),
  join_vars = exprs(ADY),
  filter_add = ADY > 0,
  filter_join = ADY.join < ADY,
  mode = "first",
  check_type = "none"
)

# add highest hemoglobin value within two weeks before AE,
# take earliest if more than one
adae <- tribble(
  ~USUBJID, ~ASTDY,
  "1", 3,
  "1", 22,
  "2", 2
)

dadb <- tribble(
```

derive_vars_joined

```r
derive_vars_joined(
  adae,
  dataset_add = adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVAL, desc(ADY)),
  new_vars = exprs(HGB_MAX = AVAL, HGB_DY = ADY),
  filter_add = PARAMCD == "HGB",
  filter_join = ASTDY - 14 <= ADY & ADY <= ASTDY,
  mode = "last"
)
```

# Add APERIOD, APERIODC based on ADSL
adsl <- tribble(
  ~USUBJID, ~AP01SDT, ~AP01EDT, ~AP02SDT, ~AP02EDT,
  "1", "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07",
  "2", "2021-02-02", "2021-03-02", "2021-03-03", "2021-04-01"
) %>%
  mutate(across(ends_with("DT"), ymd)) %>%
  mutate(STUDYID = "xyz")

period_ref <- create_period_dataset(
  adsl,
  new_vars = exprs(APERSDT = APxxSDT, APEREDT = APxxEDT)
)

period_ref

adae <- tribble(
  ~USUBJID, ~ASTDT,
  "1", "2021-01-01",
  "1", "2021-01-05",
  "1", "2021-02-05",
  "1", "2021-03-05",
  "1", "2021-04-05",
  "2", "2021-02-15",
) %>%
  mutate(
    ASTDT = ymd(ASTDT),
    STUDYID = "xyz"
  )

derive_vars_joined(
  adae,
  dataset_add = adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVAL, desc(ADY)),
  new_vars = exprs(HGB_MAX = AVAL, HGB_DY = ADY),
  filter_add = PARAMCD == "HGB",
  filter_join = ASTDY - 14 <= ADY & ADY <= ASTDY,
  mode = "last"
)
```

~USUBJID, ~PARAMCD, ~ADY, ~AVAL,
"1", "HGB", 1, 8.5,
"1", "HGB", 3, 7.9,
"1", "HGB", 5, 8.9,
"1", "HGB", 8, 8.0,
"1", "HGB", 9, 8.0,
"1", "HGB", 16, 7.4,
"1", "HGB", 24, 8.1,
"1", "ALB", 1, 42,
)
derive_vars_last_dose

Derive Last Dose

Description

[Deprecated]

This function is deprecated, please use derive_vars_joined() instead.

Usage

derive_vars_last_dose(dataset,
  
adae,
  
dataset_add = period_ref,
  
by_vars = exprs(STUDYID, USUBJID),
  
join_vars = exprs(APERSDT, APEREDT),
  
filter_join = APERSDT <= ASTDT & ASTDT <= APEREDT
  
)

# Add day since last dose (LDRELD)
adae <- tribble(
  ~USUBJID, ~ASTDT, ~AESEQ,
  "1", "2020-02-02", 1,
  "1", "2020-02-04", 2
) %>%
  mutate(ASTDT = ymd(ASTDT))

ex <- tribble(
  ~USUBJID, ~EXSDTC,
  "1", "2020-01-10",
  "1", "2020-01",
  "1", "2020-01-20",
  "1", "2020-02-03"
)

## Please note that EXSDT is created via the order argument and then used
## for new_vars, filter_add, and filter_join

derive_vars_joined(
  adae,
  
dataset_add = ex,
  
by_vars = exprs(USUBJID),
  
order = exprs(EXSDT = convert_dtc_to_dt(EXSDTC)),
  
new_vars = exprs(LDRELD = compute_duration(  
    start_date = EXSDT, end_date = ASTDT  
  )),
  
filter_add = !is.na(EXSDT),
  
filter_join = EXSDT <= ASTDT,
  
mode = "last"
  
)
derive_vars_last_dose

dataset_ex,
filter_ex = NULL,
by_vars = exprs(STUDYID, USUBJID),
dose_id = exprs(),
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 rlang::exprs).
dose_id     Variables to identify unique dose (created by rlang::exprs). Defaults to empty exprs().
dose_date   The EX dose date variable. A date or date-time object is expected.
analysis_date The analysis date variable. A date or date-time object is expected.
single_dose_condition The condition for checking if 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 rlang::exprs (e.g. exprs(VISIT)), or named list returned by exprs() (e.g. exprs(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 exprs() listing the traceability variables, e.g. exprs(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

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

When doing date comparison to identify last dose, date-time imputations are done as follows:

- dose_date: time is imputed to 00:00:00 if the variable is a date variable
derive_vars_merged

Add New Variable(s) to the Input Dataset Based on Variables from Another Dataset

Description

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

Value

Input dataset with EX source variables from last dose added.

See Also

Other deprecated: derive_param_extreme_event(), derive_var_basetype(), derive_var_confirmation_flag(), derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character()
Usage

```r
derive_vars_merged(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars = NULL,
  filter_add = NULL,
  mode = NULL,
  match_flag = NULL,
  missing_values = NULL,
  check_type = "warning",
  duplicate_msg = NULL
)
```

Arguments

dataset | Input dataset
---|---
The variables specified by the `by_vars` argument are expected.
dataset_add | Additional dataset
---|---
The variables specified by the `by_vars`, the `new_vars`, and the `order` argument are expected.
by_vars | Grouping variables
---|---
The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. Variables from the additional dataset can be renamed by naming the element, i.e., `by_vars = exprs(<name in input dataset> = <name in additional dataset>)`, similar to the dplyr joins.

*Permitted Values:* list of variables created by `exprs()`
order | Sort order
---|---
If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.
Variables defined by the `new_vars` argument can be used in the sort order.

*Permitted Values:* list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or NULL
new_vars | Variables to add
---|---
The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., `new_vars = exprs(<new name> = <old name>)`. For example `new_vars = exprs(var1, var2)` adds variables `var1` and `var2` from `dataset_add` to the input dataset.
And `new_vars = exprs(var1, new_var2 = old_var2)` takes `var1` and `old_var2` from `dataset_add` and adds them to the input dataset renaming `old_var2` to `new_var2`. 
Values of the added variables can be modified by specifying an expression. For example, `new_vars = LASTRSP = exprs(str_to_upper(AVALC))` adds the variable `LASTRSP` to the dataset and sets it to the upper case value of `AVALC`.

If the argument is not specified or set to `NULL`, all variables from the additional dataset (dataset_add) are added.

**Permitted Values:** list of variables or named expressions created by `exprs()`

### filter_add
Filter for additional dataset (dataset_add)

Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered.

Variables defined by the `new_vars` argument can be used in the filter condition.

**Permitted Values:** a condition

### mode
Selection mode

Determines if the first or last observation is selected. If the `order` argument is specified, mode must be non-null.

If the `order` argument is not specified, the `mode` argument is ignored.

**Permitted Values:** "first", "last", NULL

### match_flag
Match flag

If the argument is specified (e.g., `match_flag = FLAG`), the specified variable (e.g., `FLAG`) is added to the input dataset. This variable will be `TRUE` for all selected records from dataset_add which are merged into the input dataset, and `NA` otherwise.

**Permitted Values:** Variable name

### missing_values
Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

**Permitted Values:** named list of expressions, e.g., `exprs(BASEC = "MISSING", BASE = -1)`

### check_type
Check uniqueness?

If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

**Permitted Values:** "none", "warning", "error"

### duplicate_msg
Message of unique check

If the uniqueness check fails, the specified message is displayed.

**Default:**

```
paste("Dataset \`dataset_add\` contains duplicate records with respect to", enumerate(vars2chr(by_vars)))
```

### Details

1. The new variables (`new_vars`) are added to the additional dataset (dataset_add).
derive_vars_merged

2. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.

3. If order is specified, for each by group the first or last observation (depending on mode) is selected.

4. The variables specified for new_vars are merged to the input dataset using left_join(). I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the additional dataset the new variables are set as specified by missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_transposed(), get_summary_records()

Examples

library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~VSTESTCD, ~VISIT, ~VSTRESN, ~VSTRESU, ~VSDTC,
  "PILOT01", "VS", "01-1302", "HEIGHT", "SCREENING", 177.8, "cm", "2013-08-20",
  "PILOT01", "VS", "01-1302", "WEIGHT", "SCREENING", 81.19, "kg", "2013-08-20",
  "PILOT01", "VS", "01-1302", "WEIGHT", "BASELINE", 82.1, "kg", "2013-08-29",
  "PILOT01", "VS", "01-1302", "WEIGHT", "WEEK 2", 81.19, "kg", "2013-09-15",
  "PILOT01", "VS", "01-1302", "WEIGHT", "WEEK 4", 82.56, "kg", "2013-09-24",
  "PILOT01", "VS", "01-1302", "WEIGHT", "WEEK 6", 80.74, "kg", "2013-10-08",
  "PILOT01", "VS", "01-1302", "WEIGHT", "WEEK 8", 82.1, "kg", "2013-10-22",
  "PILOT01", "VS", "01-1302", "WEIGHT", "WEEK 12", 82.1, "kg", "2013-11-05",
  "PILOT01", "VS", "17-1344", "HEIGHT", "SCREENING", 163.5, "cm", "2014-01-01",
  "PILOT01", "VS", "17-1344", "WEIGHT", "SCREENING", 58.06, "kg", "2014-01-01",
  "PILOT01", "VS", "17-1344", "WEIGHT", "WEEK 4", 57.97, "kg", "2014-02-07",
  "PILOT01", "VS", "17-1344", "WEIGHT", "WEEK 6", 58.97, "kg", "2014-02-19",
  "PILOT01", "VS", "17-1344", "WEIGHT", "WEEK 8", 57.79, "kg", "2014-03-14"
)

dm <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~AGE, ~AGEU,
  "PILOT01", "DM", "01-1302", 61, "YEARS",
  "PILOT01", "DM", "17-1344", 64, "YEARS")
# Merging all dm variables to vs
derive_vars_merged(
    vs,
    dataset_add = select(dm, -DOMAIN),
    by_vars = exprs(STUDYID, USUBJID)
) %>%
  select(STUDYID, USUBJID, VSTESTCD, VISIT, VSSTRESN, AGE, AGEU)

# Merge last weight to adsl
adsl <- tribble(
    ~STUDYID, ~USUBJID, ~AGE, ~AGEU, 
    "PILOT01", "01-1302", 61, "YEARS", 
    "PILOT01", "17-1344", 64, "YEARS"
)

derive_vars_merged(
    adsl,
    dataset_add = vs,
    by_vars = exprs(STUDYID, USUBJID),
    order = exprs(convert_dtc_to_dtm(VSDTC)),
    mode = "last",
    new_vars = exprs(LASTWGT = VSSTRESN, LASTWGTU = VSSTRESU),
    filter_add = VSTESTCD == "WEIGHT",
    match_flag = vsdatafl
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, LASTWGT, LASTWGTU, vsdatafl)

# Derive treatment start datetime (TRTSDTM)
ex <- tribble(
    ~STUDYID, ~DOMAIN, ~USUBJID, ~EXSTDY, ~EXENDY, ~EXSTDTC, ~EXENDTC, 
    "PILOT01", "EX", "01-1302", 1, 18, "2013-08-29", "2013-09-15", 
    "PILOT01", "EX", "01-1302", 19, 69, "2013-09-16", "2013-11-05", 
)
## Impute exposure start date to first date/time
ex_ext <- derive_vars_dtm(
    ex,
    dtc = EXSTDTC,
    new_vars_prefix = "EXST",
    highest_imputation = "M",
)
## Add first exposure datetime and imputation flags to adsl
derive_vars_merged(
    select(dm, STUDYID, USUBJID),
    dataset_add = ex_ext,
    by_vars = exprs(STUDYID, USUBJID),
    new_vars = exprs(TRTSDTM = EXSTDTM, TRTSDTF = EXSTDTF, TRTSTMF = EXSTTMF),
    order = exprs(EXSTDTM),
    mode = "first"
# Derive treatment end datetime (TRTEDTM)
## Impute exposure end datetime to last time, no date imputation
ex_ext <- derive_vars_dtm(
  ex,
  dtc = EXENDTC,
  new_vars_prefix = "EXEN",
  time_imputation = "last",
)
## Add last exposure datetime and imputation flag to adsl
derive_vars_merged(
  select(adsl, STUDYID, USUBJID),
  dataset_add = ex_ext,
  filter_add = !is.na(EXENDTM),
  by_vars = exprs(STUDYID, USUBJID),
  new_vars = exprs(TRTEDTM = EXENDTM, TRTETMF = EXENTMF),
  order = exprs(EXENDTM),
  mode = "last"
)
# Modify merged values and set value for non matching observations
adsl <- tribble(
  ~USUBJID, ~SEX, ~COUNTRY,
  "ST42-1", "F", "AUT",
  "ST42-2", "M", "MWI",
  "ST42-3", "M", "NOR",
  "ST42-4", "F", "UGA"
)
adv <- tribble(
  ~USUBJID, ~PARAMCD, ~AVISIT, ~AVISITN, ~AVAL,
  "ST42-1", "WEIGHT", "BASELINE", 0, 66,
  "ST42-1", "WEIGHT", "WEEK 2", 1, 68,
  "ST42-2", "WEIGHT", "BASELINE", 0, 88,
  "ST42-3", "WEIGHT", "WEEK 2", 1, 55,
  "ST42-3", "WEIGHT", "WEEK 4", 2, 50
)
derive_vars_merged(
  adsl,
  dataset_add = adv,
  by_vars = exprs(USUBJID),
  new_vars = exprs(
    LSTVSCAT = if_else(AVISIT == "BASELINE", "BASELINE", "POST-BASELINE")
  ),
  order = exprs(AVISITN),
  mode = "last",
  missing_values = exprs(LSTVSCAT = "MISSING")
)
derive_vars_merged_lookup

*Merge Lookup Table with Source Dataset*

**Description**

Merge user-defined lookup table with the input dataset. Optionally print a list of records from the input dataset that do not have corresponding mapping from the lookup table.

**Usage**

```r
derive_vars_merged_lookup(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars = NULL,
  mode = NULL,
  filter_add = NULL,
  check_type = "warning",
  duplicate_msg = NULL,
  print_not_mapped = TRUE
)
```

**Arguments**

- `dataset` Input dataset
  - The variables specified by the `by_vars` argument are expected.
- `dataset_add` Lookup table
  - The variables specified by the `by_vars` argument are expected.
- `by_vars` Grouping variables
  - The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. Variables from the additional dataset can be renamed by naming the element, i.e., `by_vars = exprs(<name in input dataset> = <name in additional dataset>)`, similar to the dplyr joins.
  - *Permitted Values*: list of variables created by `exprs()`
- `order` Sort order
  - If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.
  - Variables defined by the `new_vars` argument can be used in the sort order.
  - *Permitted Values*: list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or NULL
**new_vars**
Variables to add
The specified variables from the additional dataset are added to the output dataset.
Variables can be renamed by naming the element, i.e., `new_vars = exprs(<new name> = <old name>)`.
For example `new_vars = exprs(var1, var2)` adds variables `var1` and `var2` from `dataset_add` to the input dataset.
And `new_vars = exprs(var1, new_var2 = old_var2)` takes `var1` and `old_var2` from `dataset_add` and adds them to the input dataset renaming `old_var2` to `new_var2`.
Values of the added variables can be modified by specifying an expression.
For example, `new_vars = LASTRSP = exprs(str_to_upper(AVALC))` adds the variable `LASTRSP` to the dataset and sets it to the upper case value of `AVALC`.
If the argument is not specified or set to `NULL`, all variables from the additional dataset (`dataset_add`) are added.

*Permitted Values:* list of variables or named expressions created by `exprs()`

**mode**
Selection mode
Determines if the first or last observation is selected. If the `order` argument is specified, mode must be non-null.
If the `order` argument is not specified, the `mode` argument is ignored.

*Permitted Values:* "first", "last", `NULL`

**filter_add**
Filter for additional dataset (`dataset_add`)
Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered.
Variables defined by the `new_vars` argument can be used in the filter condition.

*Permitted Values:* a condition

**check_type**
Check uniqueness?
If "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

*Permitted Values:* "none", "warning", "error"

**duplicate_msg**
Message of unique check
If the uniqueness check fails, the specified message is displayed.

*Default:* 

```
paste("Dataset `dataset_add` contains duplicate records with respect to",
     enumerate(vars2chr(by_vars)))
```

**print_not_mapped**
Print a list of unique `by_vars` values that do not have corresponding records from the lookup table?

*Default:* `TRUE`

*Permitted Values:* `TRUE`, `FALSE`

**Value**
The output dataset contains all observations and variables of the input dataset, and add the variables specified in `new_vars` from the lookup table specified in `dataset_add`. Optionally prints a list of unique `by_vars` values that do not have corresponding records from the lookup table (by specifying `print_not_mapped = TRUE`).
See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

Examples

```r
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~VISIT, ~VSTESTCD, ~VSTEST,
  "PILOT01", "VS", "01-1028", "SCREENING", "HEIGHT", "Height",
  "PILOT01", "VS", "01-1028", "SCREENING", "TEMP", "Temperature",
  "PILOT01", "VS", "01-1028", "BASELINE", "TEMP", "Temperature",
  "PILOT01", "VS", "01-1028", "WEEK 4", "TEMP", "Temperature",
  "PILOT01", "VS", "01-1028", "SCREENING 1", "WEIGHT", "Weight",
  "PILOT01", "VS", "01-1028", "BASELINE", "WEIGHT", "Weight",
  "PILOT01", "VS", "01-1028", "WEEK 4", "WEIGHT", "Weight",
  "PILOT01", "VS", "04-1325", "SCREENING", "HEIGHT", "Height",
  "PILOT01", "VS", "04-1325", "SCREENING", "TEMP", "Temperature",
  "PILOT01", "VS", "04-1325", "BASELINE", "TEMP", "Temperature",
  "PILOT01", "VS", "04-1325", "WEEK 4", "TEMP", "Temperature",
  "PILOT01", "VS", "04-1325", "SCREENING 1", "WEIGHT", "Weight",
  "PILOT01", "VS", "04-1325", "BASELINE", "WEIGHT", "Weight",
  "PILOT01", "VS", "04-1325", "WEEK 4", "WEIGHT", "Weight",
  "PILOT01", "VS", "10-1027", "SCREENING", "HEIGHT", "Height",
  "PILOT01", "VS", "10-1027", "SCREENING", "TEMP", "Temperature",
  "PILOT01", "VS", "10-1027", "BASELINE", "TEMP", "Temperature",
  "PILOT01", "VS", "10-1027", "WEEK 4", "TEMP", "Temperature",
  "PILOT01", "VS", "10-1027", "SCREENING 1", "WEIGHT", "Weight",
  "PILOT01", "VS", "10-1027", "BASELINE", "WEIGHT", "Weight",
  "PILOT01", "VS", "10-1027", "WEEK 4", "WEIGHT", "Weight"
)

param_lookup <- tribble(
  ~VSTESTCD, ~VSTEST, ~PARAMCD, ~PARAM,
  "SYSBP", "Systolic Blood Pressure", "SYSBP", "Syst Blood Pressure (mmHg)",
  "WEIGHT", "Weight", "WEIGHT", "Weight (kg)",
  "HEIGHT", "Height", "HEIGHT", "Height (cm)",
  "TEMP", "Temperature", "TEMP", "Temperature (C)",
  "MAP", "Mean Arterial Pressure", "MAP", "Mean Art Pressure (mmHg)",
  "BMI", "Body Mass Index", "BMI", "Body Mass Index(kg/m^2)",
  "BSA", "Body Surface Area", "BSA", "Body Surface Area(m^2)"
)

derive_vars_merged_lookup(
  dataset = vs,
  dataset_add = param_lookup,
  by_vars = exprs(VSTESTCD),
  new_vars = exprs(PARAMCD, PARAM),
  print_notMapped = TRUE
)
**derive_vars_period**  
Add Subperiod, Period, or Phase Variables to ADSL

**Description**

The function adds subperiod, period, or phase variables like `P01S1SDT`, `P01S2SDT`, `AP01SDTM`, `AP02SDTM`, `TRT01A`, `TRT02A`, `PH1SDT`, `PH2SDT`, ... to the input dataset. The values of the variables are defined by a period reference dataset which has one observations per patient and subperiod, period, or phase.

**Usage**

```r
derive_vars_period(
  dataset,
  dataset_ref,
  new_vars,
  subject_keys = get_admiral_option("subject_keys")
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| `dataset`      | ADSL dataset  
The variables specified by `subject_keys` are expected. |
| `dataset_ref`  | Period reference dataset  
The variables specified by `new_vars` and `subject_keys` are expected.  
If subperiod variables are requested, `APERIOD` and `ASPER` are expected. If period variables are requested, `APERIOD` is expected. If phase variables are requested, `APHASE` is expected. |
| `new_vars`     | New variables  
A named list of variables like `exprs(PHwSDT = PHSDT, PHwEDT = PHEDT, APHASEw = APHASE)` is expected. The left hand side of the elements defines a set of variables (in CDISC notation) to be added to the output dataset. The right hand side defines the source variable from the period reference dataset.  
If the lower case letter "w" is used it refers to a phase variable, if the lower case letters "xx" are used it refers to a period variable, and if both "xx" and "w" are used it refers to a subperiod variable.  
Only one type must be used, e.g., all left hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables.  
If period and subperiod variables are required, separate calls must be used. |
| `subject_keys` | Variables to uniquely identify a subject  
A list of expressions where the expressions are symbols as returned by `exprs()` is expected. |
derive_vars_period

Details

For each subperiod/period/phase in the period reference dataset and each element in new_vars a variable (LHS value of new_vars) is added to the output dataset and set to the value of the source variable (RHS value of new_vars).

Value

The input dataset with subperiod/period/phase variables added (see "Details" section)

See Also

create_period_dataset()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_var_dthcaus(), derive_var_extreme_dtm(), derive_var_extreme_dt(), derive_vars_aage()

Examples

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

adsl <- tibble(STUDYID = "xyz", USUBJID = c("1", "2"))

# Add period variables to ADSL
period_ref <- tribble(
  ~USUBJID, ~APERIOD, ~APERSDT, ~APEREDT,
  "1", 1, "2021-01-04", "2021-02-06",
  "1", 2, "2021-02-07", "2021-03-07",
  "2", 1, "2021-02-02", "2021-03-02",
  "2", 2, "2021-03-03", "2021-04-01"
) %>%
mutate(
  STUDYID = "xyz",
  APERIOD = as.integer(APERIOD),
  across(matches("APER\[ES\]DT"), ymd)
)

derive_vars_period(
  adsl,
  dataset_ref = period_ref,
  new_vars = exprs(APxxSDT = APERSDT, APxxEDT = APEREDT)
) %>%
select(STUDYID, USUBJID, AP01SDT, AP01EDT, AP02SDT, AP02EDT)

# Add phase variables to ADSL
phase_ref <- tribble(
  ~USUBJID, ~APHASEN, ~PHSDT, ~PHEDT, ~APHASE,
  "1", 1, "2021-01-04", "2021-02-06", "TREATMENT",
  "1", 2, "2021-02-07", "2021-03-07", "FUP",
  "2", 1, "2021-02-02", "2021-03-02", "TREATMENT"
) %>%

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

adsl <- tibble(STUDYID = "xyz", USUBJID = c("1", "2"))

# Add period variables to ADSL
period_ref <- tribble(
  ~USUBJID, ~APERIOD, ~APERSDT, ~APEREDT,
  "1", 1, "2021-01-04", "2021-02-06",
  "1", 2, "2021-02-07", "2021-03-07",
  "2", 1, "2021-02-02", "2021-03-02",
  "2", 2, "2021-03-03", "2021-04-01"
) %>%
mutate(
  STUDYID = "xyz",
  APERIOD = as.integer(APERIOD),
  across(matches("APER\[ES\]DT"), ymd)
)

derive_vars_period(
  adsl,
  dataset_ref = period_ref,
  new_vars = exprs(APxxSDT = APERSDT, APxxEDT = APEREDT)
) %>%
select(STUDYID, USUBJID, AP01SDT, AP01EDT, AP02SDT, AP02EDT)

# Add phase variables to ADSL
phase_ref <- tribble(
  ~USUBJID, ~APHASEN, ~PHSDT, ~PHEDT, ~APHASE,
  "1", 1, "2021-01-04", "2021-02-06", "TREATMENT",
  "1", 2, "2021-02-07", "2021-03-07", "FUP",
  "2", 1, "2021-02-02", "2021-03-02", "TREATMENT"
) %>%
```
```

```

### Derive Query Variables

**Description**

Derive Query Variables

**Usage**

```r
derive_vars_query(dataset, dataset_queries)
```
Arguments

- **dataset**
  Input dataset.
- **dataset_queries**
  A dataset containing required columns `PREFIX`, `GRNAME`, `SRCVAR`, `TERMNAME`, `TERMID`, and optional columns `GRPID`, `SCOPE`, `SCOPEN`.

The content of the dataset will be verified by `assert_valid_queries()`.
`create_query_data()` can be used to create the dataset.

Details

This function can be used to derive CDISC variables such as `SMQzzNAM`, `SMQzzCD`, `SMQzzSC`, `SMQzzSCN`, and `CQzzNAM` in ADAE and ADMH, and variables such as `SDGzzNAM`, `SDGzzCD`, and `SDGzzSC` in ADCM. An example usage of this function can be found in the [OCCDS vignette](#).

A query dataset is expected as an input to this function. See the [Queries Dataset Documentation vignette](#) for descriptions, or call `data("queries")` for an example of a query dataset.

For each unique element in `PREFIX`, the corresponding "NAM" variable will be created. For each unique `PREFIX`, if `GRPID` is not "" or NA, then the corresponding "CD" variable is created; similarly, if `SCOPE` is not "" or NA, then the corresponding "SC" variable will be created; if `SCOPEN` is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in `dataset`, the "NAM" variable takes the value of `GRNAME` if the value of `TERMNAME` or `TERMID` in `dataset_queries` matches the value of the respective `SRCVAR` in `dataset`. Note that `TERMNAME` in `dataset_queries` dataset may be NA only when `TERMID` is non-NA and vice versa. The "CD", "SC", and "SCN" variables are derived accordingly based on `GRPID`, `SCOPE`, and `SCOPEN` respectively, whenever not missing.

Value

The input dataset with query variables derived.

See Also

`create_query_data()`, `assert_valid_queries()`

OCCDS Functions: `derive_var_trtemfl()`, `derive_vars_atc()`, `get_terms_from_db()`

Examples

```r
library(tibble)
data("queries")
adae <- 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_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

```r
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
See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_exist_flag()`, `derive_var_merged_summary()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_joined()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `get_summary_records()`

Examples

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

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

facm <- 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", "DA0",
                 "BP40257-1001", "1", "2007001", "CMATC3CD", "DA0A",
                 "BP40257-1001", "1", "2007001", "CMATC4CD", "DA0AA",
                 "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 = exprs(USUBJID, CMREFID = FAREFID),
    key_var = FATESTCD,
    value_var = FASTRESC
  ) %>%
  select(USUBJID, CMDECOD, starts_with("CMATC"))
```
**derive_var_age_years**

**Derive Age in Years**

**Description**

Converts the given age variable (age_var) to the unit 'years' from the current units given in the age_var+U variable or age_unit argument and stores in a new variable (new_var).

**Usage**

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

**Arguments**

- **dataset**
  - Input dataset.
  - The column specified by the age_var argument is expected.

- **age_var**
  - Age variable.
  - A numeric object is expected.

- **age_unit**
  - Age unit.
  - The age_unit argument is only expected when there is NOT a variable age_var+U in dataset. This gives the unit of the age_var variable and is used to convert AGE to 'years' so that grouping can occur.
  - Default: NULL
  - Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds'

- **new_var**
  - New age variable to be created in years. The returned values are doubles and NOT integers.

**Details**

This function is used to convert an age variable into the unit 'years' which can then be used to create age groups. The resulting column contains the equivalent years as a double. Note, underlying computations assume an equal number of days in each year (365.25).

**Value**

The input dataset (dataset) with new_var variable added in years.

**See Also**

- `derive_vars_duration()`

  ADSL Functions that returns variable appended to dataset: `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_var_extreme_dt()`, `derive_vars_aage()`, `derive_vars_period()`
Examples

library(tibble)

# Derive age with age units specified
data <- tribble(~AGE, ~AGEU,
    27, "days",
    24, "months",
    3, "years",
    4, "weeks",
    1, "years"
)
derive_var_age_years(data, AGE, new_var = AAGE)

# Derive age without age units variable specified
data <- tribble(~AGE,
    12,
    24,
    36,
    48
)
derive_var_age_years(data, AGE, age_unit = "months", new_var = AAGE)

derive_var_analysis_ratio

Derive Ratio Variable

Description

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

Usage

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

Arguments

dataset Input dataset

numer_var Variable containing numeric values to be used in the numerator of the ratio calculation.

denom_var Variable containing numeric values to be used in the denominator of the ratio calculation.

new_var A user-defined variable that will be appended to the dataset. The default behavior will take the denominator variable and prefix it with R2 and append to the dataset. Using this argument will override this default behavior. Default is NULL.
derive_var_analysis_ratio

Details

A user wishing to calculate a Ratio to Baseline, AVAL / BASE will have returned a new variable R2BASE that will be appended to the input dataset. Ratio to Analysis Range Lower Limit AVAL / ANRLO will return a new variable R2ANRLO, and Ratio to Analysis Range Upper Limit AVAL / ANRHI will return a new variable R2ANRHI. Please note how the denominator variable has the prefix R2----. A user can override the default returned variables by using the new_var argument. Also, values of 0 in the denominator will return NA in the derivation.

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

Value

The input dataset with a ratio variable appended

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_atoxgr(), derive_var_base(), derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()

Examples

library(tibble)

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

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

# Returns user-defined variables

data %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = BASE, new_var = R01BASE) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRLO, new_var = R01ANRLO) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRHI, new_var = R01ANRHI)
**derive_var_anrind**

---

**Description**

Derive Reference Range Indicator

**Usage**

`derive_var_anrind(dataset, use_a1hia1lo = FALSE)`

**Arguments**

- `dataset`: The input dataset
- `use_a1hia1lo`: A logical value indicating whether to use A1H1 and A1LO in the derivation of ANRIND.

**Details**

In the case that A1H1 and A1LO are to be used, 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

In the case that A1H1 and A1LO are not to be used, 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
- "HIGH" if AVAL is greater than ANRHI

**Value**

The input dataset with additional column ANRIND

**See Also**

BDS-Findings Functions that returns variable appended to dataset: `derive_basertype_records()`, `derive_var_analysis_ratio()`, `derive_var_atoxgr_dir()`, `derive_var_atoxgr()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_ontrtfl()`, `derive_var_pchg()`, `derive_var_shift()`
derive_var_atoxgr

Examples

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

vs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ANRLO, ~ANRHI, ~A1LO, ~A1HI,
    "P01",    "PUL",  70,   60,  100,   40,  110,
    "P01",    "PUL",  57,   60,  100,   40,  110,
    "P01",    "PUL",  60,   60,  100,   40,  110,
    "P01",    "DIABP", 109,   60,  100,   40,  110,
    "P02",    "PUL",  100,   60,  100,   40,  110,
    "P02",    "DIABP",  80,   60,  100,   40,  110,
    "P03",    "PUL",   40,   60,  100,   40,  110,
)

vs %>% derive_var_anrind(use_a1hia1lo = TRUE)
vs %>% derive_var_anrind(use_a1hia1lo = FALSE)

derive_var_atoxgr    Derive Lab High toxicity Grade 0 - 4 and Low Toxicity Grades 0 - (-4)

Description

Derives character lab grade based on high and low severity/toxicity grade(s).

Usage

derive_var_atoxgr(
  dataset,
  lotox_description_var = ATOXDSCL,
  hitox_description_var = ATOXDSCH
)

Arguments

dataset           Input data set
  The columns ATOXGRL, ATOXGRH and specified by lotox_description_var, and hitox_description_var parameters are expected.

lotox_description_var           Variable containing the toxicity grade description for low values, eg. "Anemia"

hitox_description_var           Variable containing the toxicity grade description for high values, eg. "Hemoglobin Increased".
Details

Created variable ATOXGR will contain values "-4", "-3", "-2", "-1" for low values and "1", "2", "3", "4" for high values, and will contain "0" if value is gradable and does not satisfy any of the criteria for high or low values. ATOXGR is set to missing if information not available to give a grade.

Function applies the following rules:

• High and low missing - overall missing
• Low grade not missing and > 0 - overall holds low grade
• High grade not missing and > 0 - overall holds high grade
• (Only high direction OR low direction is NORMAL) and high grade normal - overall NORMAL
• (Only low direction OR high direction is NORMAL) and low grade normal - overall NORMAL
• otherwise set to missing

Value

The input data set with the character variable added

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()

Examples

library(tibble)

adlb <- tribble(
  ~ATOXDSCL, ~ATOXDSCH, ~ATOXGRL, ~ATOXGRH,
  "Hypoglycemia", "Hyperglycemia", NA_character_, "0",
  "Hypoglycemia", "Hyperglycemia", "0", "1",
  "Hypoglycemia", "Hyperglycemia", "0", "0",
  NA_character_, "INR Increased", NA_character_, "0",
  "Hypophosphatemia", NA_character_, "1", NA_character_
)

derive_var_atoxgr(adlb)
Description

Derives a character lab grade based on severity/toxicity criteria.

Usage

```
derive_var_atoxgr_dir(
    dataset,
    new_var,
    tox_description_var,
    meta_criteria,
    criteria_direction,
    get_unit_expr,
    signif_dig = 15
)
```

Arguments

dataset  Input data set
The columns specified by tox_description_var parameter is expected.

new_var  Name of the character grade variable to create, for example, ATOXGRH or ATOXGRL.

tox_description_var  Variable containing the description of the grading criteria. For example: "Anemia" or "INR Increased".

meta_criteria  Metadata data set holding the criteria (normally a case statement)
Permitted Values: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv5, atoxgr_criteria_daids

- atoxgr_criteria_ctcv4 implements Common Terminology Criteria for Adverse Events (CTCAE) v4.0
- atoxgr_criteria_ctcv5 implements Common Terminology Criteria for Adverse Events (CTCAE) v5.0
- atoxgr_criteria_daids implements Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events

The metadata should have the following variables:

- TERM: variable to hold the term describing the criteria applied to a particular lab test, eg. "Anemia" or "INR Increased". Note: the variable is case insensitive.
- DIRECTION: variable to hold the direction of the abnormality of a particular lab test value. "L" is for LOW values, "H" is for HIGH values. Note: the variable is case insensitive.
- SI_UNIT_CHECK: variable to hold unit of particular lab test. Used to check against input data if criteria is based on absolute values.
• **VAR_CHECK**: variable to hold comma separated list of variables used in criteria. Used to check against input data that variables exist.
• **GRADE_CRITERIA_CODE**: variable to hold code that creates grade based on defined criteria.
• **FILTER**: Required only for DAIDS grading, specifies admiral code to filter the lab data based on a subset of subjects (e.g. AGE > 18 YEARS)

**criteria_direction**
- Direction (L= Low, H = High) of toxicity grade.
- Permitted Values: "L", "H"

**get_unit_expr**
- An expression providing the unit of the parameter
- The result is used to check the units of the input parameters. Compared with SI_UNIT_CHECK in metadata (see meta_criteria parameter).
- Permitted Values: A variable containing unit from the input dataset, or a function call, for example, `get_unit_expr = extract_unit(PARAM)`.

**signif_dig**
- Number of significant digits to use when comparing a lab value against another value.
- Significant digits used to avoid floating point discrepancies when comparing numeric values.

**Details**

new_var is derived with values NA, "0", "1", "2", "3", "4", where "4" is the most severe grade
- "4" is where the lab value satisfies the criteria for grade 4.
- "3" is where the lab value satisfies the criteria for grade 3.
- "2" is where the lab value satisfies the criteria for grade 2.
- "1" is where the lab value satisfies the criteria for grade 1.
- "0" is where a grade can be derived and is not grade "1", "2", "3" or "4".
- NA is where a grade cannot be derived.

**Value**
- The input dataset with the character variable added

**See Also**
- BDS-Findings Functions that returns variable appended to dataset: `derive_basetype_records()`, `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_ontrtf1()`, `derive_var_pchg()`, `derive_var_shift()`

**Examples**
```
library(tibble)

data <- tribble(
  ~ATOXDSCL, ~AVAL, ~ANRLO, ~ANRHI, ~PARAM, 
  "Hypoglycemia", 119, 4, 7, "Glucose (mmol/L)" ,
)```
Derive Baseline Variables

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

Note: This is a wrapper function for the more generic derive_vars_merged().

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 the condition specified in 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 vari-

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(),
derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_atoxgr(),
derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()

Examples

library(tibble)

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

## Derive 'BASE' variable from 'AVAL'

derive_var_base(
derive_var_basetype

```r
dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = AVAL,
  new_var = BASE
)

## Derive `BASEC` variable from `AVALC`
衍生_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = AVALC,
  new_var = BASEC
)

## Derive `BNRIND` variable from `ANRIND`
衍生_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = ANRIND,
  new_var = BNRIND
)
```

### Description

**[Deprecated]**

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

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

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 `rlang::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

See Also

Other deprecated: derive_param_extreme_event(), derive_var_confirmation_flag(), derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character(), derive_vars_last_dose()

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

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_atoxgr(), derive_var_base(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()
Examples

```r
library(tibble)

advs <- 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_confirmation_flag**

*Derive Confirmation Flag*

**Description**

*Deprecated*

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

**Usage**

```r
derive_var_confirmation_flag(
  dataset,
  by_vars,
  order,
  new_var,
  tmp_obs_nr_var = NULL,
  join_vars,
  join_type,
  first_cond = NULL,
  filter,
  true_value = "Y",
  false_value = NA_character_,
  check_type = "warning"
)
```

**Arguments**

- **dataset**
  - Input dataset
  - The variables specified by the `by_vars` and `join_vars` parameter are expected.

- **by_vars**
  - By variables
  - The specified variables are used as by variables for joining the input dataset with itself.
derive_var_confirmation_flag

order  

The observations are ordered by the specified order.

new_var  

New variable

The specified variable is added to the input dataset.

tmp_obs_nr_var  

Temporary observation number

The specified variable is added to the input dataset and set to the observation number with respect to order. For each group by group (by_vars) the observation number starts with 1. The variable can be used in the conditions (filter, first_cond). It is not included in the output dataset. It can be used to flag consecutive observations or the last observation (see last example below).

join_vars  

Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix "join". For example to flag all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit join_vars = exprs(AVALC, AVISITN) and filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join could be specified.

The *.join variables are not included in the output dataset.

join_type  

Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if join_type = "after" is specified all observations after the original observations are kept.

For example for confirmed response or BOR in the oncology setting or confirmed deterioration in questionnaires the confirmatory assessment must be after the assessment to be flagged. Thus join_type = "after" could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation to be flagged. For example, to flag AEs occurring on or after seven days before a COVID AE. Thus join_type = "all" could be used.

Permitted Values: "before", "after", "all"

first_cond  

Condition for selecting range of data

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if filter contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example see the third example below.

filter  

Condition for selecting observations

The filter is applied to the joined dataset for flagging the confirmed observations. The condition can include summary functions. The joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example, filter = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count vals(var = AVALC.join, val = "NE") <= 1 selects observations with response "CR" and for all observations up to the confirmation observation the response is "CR" or "NE" and there is at most one "NE".
true_value: Value of new_var for flagged observations
Default: "Y"

false_value: Value of new_var for observations not flagged
Default: NA_character_

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

The following steps are performed to produce the output dataset.

Step 1:
The input dataset is joined with itself by the variables specified for by_vars. From the right hand
side of the join only the variables specified for join_vars are kept. The suffix ".join" is added to
these variables.

For example, for by_vars = USUBJID, join_vars = exprs(AVISITN, AVALC) and input dataset

# A tibble: 2 x 4
USUBJID AVISITN AVALC AVAL
<chr> <dbl> <chr> <dbl>
1 1 Y 1
1 2 N 0

the joined dataset is

A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join
<chr> <dbl> <chr> <dbl> <chr> <chr>
1 1 Y 1 1 Y
1 1 Y 1 2 N
1 2 N 0 1 Y
1 2 N 0 2 N

Step 2:
The joined dataset is restricted to observations with respect to join_type and order.
The dataset from the example in the previous step with join_type = "after" and order = exprs(AVISITN)
is restricted to

A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join
<chr> <dbl> <chr> <dbl> <chr> <chr>
1 1 Y 1 1 Y
1 1 Y 1 2 N
1 2 N 0 1 Y
1 2 N 0 2 N
Step 3:
If `first_cond` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond` is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

Step 4:
The joined dataset is grouped by the observations from the input dataset and restricted to the observations fulfilling the condition specified by `filter`.

Step 5:
The first observation of each group is selected

Step 6:
The variable specified by `new_var` is added to the input dataset. It is set to `true_value` for all observations which were selected in the previous step. For the other observations it is set to `false_value`.

Value
The input dataset with the variable specified by `new_var` added.

See Also
Other deprecated: `derive_param_extreme_event()`, `derive_var_basetype()`, `derive_var_last_dose_amt()`, `derive_var_last_dose_date()`, `derive_var_last_dose_grp()`, `derive_var_merged_cat()`, `derive_var_merged_character()`, `derive_vars_last_dose()`

---

**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 = get_admiral_option("subject_keys")
)
```
**Arguments**

- **dataset**
  Input dataset. The variables specified by `subject_keys` are required.

- **source_datasets**
  Objects of class "dthcaus_source" created by `dthcaus_source()`.

- **subject_keys**
  Variables to uniquely identify a subject
  A list of expressions where the expressions are symbols as returned by `exprs()` 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.

**See Also**

- `dthcaus_source()`
- ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_var_extreme_dtm()`, `derive_var_extreme_dt()`, `derive_vars_aage()`, `derive_vars_period()`

**Examples**

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

adsl <- tribble(
    ~STUDYID, ~USUBJID,
    "STUDY01", "PAT01",
    "STUDY01", "PAT02",
    "STUDY01", "PAT03"
  )
ae <- tribble(
    ~STUDYID, ~USUBJID, ~AESEQ, ~AEDECOD, ~AEOUT, ~AEDTHDTC,
    "STUDY01", "PAT01", 12, "SUDDEN DEATH", "FATAL", "2021-04-04"
  )
ds <- 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", "PAT03", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
    "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
)"
derive_var_dthcaus

) # Derive 'DTHCAUS' only - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive 'DTHCAUS' and add traceability variables - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD,
  set_values_to = exprs(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM,
  set_values_to = exprs(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive 'DTHCAUS' as above - now including post-study deaths with different 'DTHCAUS' value
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD,
  set_values_to = exprs(DTHDOM = "AE", DTHSEQ = AESEQ)
)

ds <- mutate(
derive_var_extreme_dt

Derive First or Last Date from Multiple Sources

Description

Add the first or last date from multiple sources to the dataset, e.g., the last known alive date (LASTALVDT).

Note: This is a wrapper function for the function derive_var_extreme_dtm().

Usage

derive_var_extreme_dt(
    dataset,
    new_var,
    ...,  
    source_datasets,
    mode,
    subject_keys = get_admiral_option("subject_keys")
)
Arguments

- **dataset**: Input dataset
  The variables specified by subject_keys are required.

- **new_var**: Name of variable to create

- **source_datasets**: Source(s) of dates. One or more date_source() objects are expected.
  A named list containing datasets in which to search for the first or last date

- **mode**: Selection mode (first or last)
  If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected.
  Permitted Values: "first", "last"

- **subject_keys**: Variables to uniquely identify a subject
  A list of expressions where the expressions are symbols as returned by exprs() 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 and observations where date is NA are removed. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable or expression specified by the date element.
3. The variables specified by the set_values_to element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.
6. The time part is removed from the new variable.

Value

The input dataset with the new variable added.

See Also

date_source(), derive_var_extreme_dtm(), derive_vars_merged()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_var_dthcaus(), derive_var_extreme_dtm(), derive_vars_aage(), derive_vars_period()

Examples

```r
library(dplyr, warn.conflicts = FALSE)
ae <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~AESEQ, ~AESTDTC, ~AEENDTC,
  "PILOT01", "AE", "01-1130", 6, "2014-05-22", NA,
  "PILOT02", "AE", "01-2230", 8, "2014-05-22", NA,
)
```
```r
# Derive variables extreme dt

derive_var_extreme_dt

# adsl

adsl <- tribble(
  ~STUDYID, ~USUBJID, ~TRTEDTM, ~TRTEDT, 
  "PILOT01", "01-1130", "2014-08-16 23:59:59", "2014-08-16", 
) %>% mutate(  
  across(TRTEDTM:TRTEDT, as.Date)
)

# lb

lb <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~LBSEQ, ~LBDTC, 
  "PILOT01", "LB", "01-1130", 322, "2014-08-16T13:10", 
  "PILOT01", "LB", "01-1133", 268, "2013-04-18T15:30", 
  "PILOT01", "LB", "01-1133", 304, "2013-04-29T10:13", 
  "PILOT01", "LB", "01-1211", 8, "2012-10-30T14:26", 
  "PILOT01", "LB", "01-1211", 162, "2013-01-08T12:13", 
)

# dm

dm <- tribble(
  ~STUDYID, ~USUBJID, ~LBSEQ, ~LBDTC, 
  "PILOT01", "01-1130", 8, "2014-05-22", NA, 
  "PILOT01", "01-1130", 7, "2014-05-22", NA, 
  "PILOT01", "01-1130", 2, "2014-03-09", "2014-03-09", 
  "PILOT01", "01-1130", 1, "2014-03-09", "2014-03-16", 
  "PILOT01", "01-1130", 3, "2014-03-09", "2014-03-16", 
  "PILOT01", "AE", "01-1130", 1, "2012-12-27", NA, 
  "PILOT01", "AE", "01-1130", 3, "2012-12-27", NA, 
  "PILOT01", "AE", "01-1130", 2, "2012-12-27", NA, 
  "PILOT01", "AE", "01-1130", 4, "2012-12-27", NA, 
  "PILOT01", "AE", "01-1211", 5, "2012-11-29", NA, 
  "PILOT01", "AE", "01-1211", 1, "2012-11-16", NA, 
  "PILOT01", "AE", "01-1211", 7, "2013-01-11", NA, 
  "PILOT01", "AE", "01-1211", 8, "2013-01-11", NA, 
  "PILOT01", "AE", "01-1211", 4, "2012-11-22", NA, 
  "PILOT01", "AE", "01-1211", 3, "2012-11-21", NA, 
  "PILOT01", "AE", "01-1211", 6, "2012-12-09", NA, 
  "PILOT01", "AE", "09-1081", 2, "2014-05-01", NA, 
  "PILOT01", "AE", "09-1081", 1, "2014-04-07", NA, 
  "PILOT01", "AE", "09-1088", 1, "2014-05-08", NA, 
  "PILOT01", "AE", "09-1088", 2, "2014-08-02", NA
)
```
~STUDYID, ~DOMAIN, ~USUBJID, ~AGE, ~AGEU,
"PILOT01", "DM", "01-1130", 84, "YEARS",
"PILOT01", "DM", "01-1133", 81, "YEARS",
"PILOT01", "DM", "01-1211", 76, "YEARS",
"PILOT01", "DM", "09-1081", 86, "YEARS",
"PILOT01", "DM", "09-1088", 69, "YEARS"
)

ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M")
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M")
)

ae_ext <- ae %>%
derive_vars_dt(
  dtc = AESTDTC,
  new_vars_prefix = "AEST",
  highest_imputation = "M"
) %>%
derive_vars_dt(
  dtc = AEENDTC,
  new_vars_prefix = "AEEN",
  highest_imputation = "M"
)

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC)
)

lb_ext <- derive_vars_dt(
  lb,
  dtc = LBDTC,
  new_vars_prefix = "LB"
)

adsl_date <- date_source(dataset_name = "adsl", date = TRTEDT)

dm %>%
derive_var_extreme_dt(
  new_var = LSTALVDT,
  ae_start, ae_end, lb_date, adsl_date,
  source_datasets = list(
    adsl = adsl,
    ae = ae_ext,
    lb = lb_ext
  ),
  mode = "last"
) %>%
# derive last alive date and traceability variables

```r
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC),
  set_values_to = exprs(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDT,
  set_values_to = exprs(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDT"
  )
)

dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
)```
derive_var_extreme_dtm

Derive First or Last Datetime from Multiple Sources

Description
Add the first or last datetime from multiple sources to the dataset, e.g., the last known alive datetime (LSTALVDM).

Usage
```
derive_var_extreme_dtm(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments
- **dataset**: Input dataset
  - The variables specified by `subject_keys` are required.
- **new_var**: Name of variable to create
- **source_datasets**: Source(s) of dates. One or more `date_source()` objects are expected.
- **mode**: Selection mode (first or last)
  - If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected.
  - Permitted Values: "first", "last"
- **subject_keys**: Variables to uniquely identify a subject
  - A list of expressions where the expressions are symbols as returned by `exprs()` 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 and observations where `date` is `NA` are removed. Then for each patient the first or last observation (with respect to `date` and `mode`) is selected.
2. The new variable is set to the variable or expression specified by the date element. If this is a date variable (rather than datetime), then the time is imputed as "00:00:00".

3. The variables specified by the set_values_to element are added.

4. The selected observations of all source datasets are combined into a single dataset.

5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.

Value
The input dataset with the new variable added.

See Also

date_source(), derive_var_extreme_dt(), derive_vars_merged()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_vars_aage(), derive_vars_period()

Examples

```r
library(dplyr, warn.conflicts = FALSE)
library(lubridate)
dm <- tribble(
  ~STUDYID, ~DOMAIN, ~SUBJID, ~AGE, ~AGEU,
  "PILOT01", "DM", "01-1130", 84, "YEARS",
  "PILOT01", "DM", "01-1133", 81, "YEARS",
  "PILOT01", "DM", "01-1211", 76, "YEARS",
  "PILOT01", "DM", "09-1081", 86, "YEARS",
  "PILOT01", "DM", "09-1088", 69, "YEARS"
)

ae <- tribble(
  ~STUDYID, ~DOMAIN, ~SUBJID, ~AESEQ, ~AESTDTC, ~AEENDTC,
  "PILOT01", "AE", "01-1133", 6, "2014-05-22", NA,
  "PILOT01", "AE", "01-1133", 8, "2014-05-22", NA,
  "PILOT01", "AE", "01-1130", 7, "2014-05-22", NA,
  "PILOT01", "AE", "01-1130", 2, "2014-03-09", "2014-03-09",
  "PILOT01", "AE", "01-1130", 1, "2014-03-09", "2014-03-16",
  "PILOT01", "AE", "01-1130", 3, "2014-03-09", "2014-03-16",
  "PILOT01", "AE", "01-1133", 1, "2012-12-27", NA,
  "PILOT01", "AE", "01-1133", 3, "2012-12-27", NA,
  "PILOT01", "AE", "01-1133", 2, "2012-12-27", NA,
  "PILOT01", "AE", "01-1133", 4, "2012-12-27", NA,
  "PILOT01", "AE", "01-1211", 5, "2012-11-29", NA,
  "PILOT01", "AE", "01-1211", 1, "2012-11-16", NA,
  "PILOT01", "AE", "01-1211", 7, "2013-01-11", NA,
  "PILOT01", "AE", "01-1211", 8, "2013-01-11", NA,
  "PILOT01", "AE", "01-1211", 4, "2012-11-22", NA,
  "PILOT01", "AE", "01-1211", 3, "2012-11-21", NA,
  "PILOT01", "AE", "01-1211", 6, "2012-12-09", NA,
)
"PILOT01", "LB", "01-1130", 322, "2014-08-16T13:10",
"PILOT01", "LB", "01-1133", 268, "2013-04-18T15:30",
"PILOT01", "LB", "01-1133", 304, "2013-04-29T10:13",
"PILOT01", "LB", "01-1211", 8, "2012-10-30T14:26",
"PILOT01", "LB", "01-1211", 162, "2013-01-08T12:13",
)

adsl <- tribble(
  ~STUDYID, ~USUBJID, ~TRTEDTM,
  "PILOT01", "01-1130", "2014-08-16 23:59:59",
  "PILOT01", "01-1211", "2013-01-12 23:59:59",
  "PILOT01", "09-1088", "2014-10-09 23:59:59"
) %>%
  mutate(
    TRTEDTM = as_datetime(TRTEDTM)
  )

# derive last known alive datetime (LSTALVDTM)
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
)
ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
)
ae_ext <- ae %>%
derive_vars_dtm(
  dtc = AESTDTC,
  new_vars_prefix = "AEST",
  highest_imputation = "M"
) %>%
derive_vars_dtm(
  dtc = AEENDTC,
  new_vars_prefix = "AEEN",
  highest_imputation = "M"
)
derive_var_extreme_dtm

```r
lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
)

lb_ext <- derive_vars_dtm(
  lb,
  dtc = LBDTC,
  new_vars_prefix = "LB"
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDTM
)

dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM)

# derive last alive datetime and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
  set_values_to = exprs(
```
derive_var_extreme_flag

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,
  flag_all = FALSE,
  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 the first or last observation is flagged.
  - Permitted Values: "first", "last"

**flag_all**
- Flag setting
  - A logical value where if set to TRUE, all records are flagged and no error or warning is issued if the first or last record is not unique.

**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). In the case where the user wants to flag multiple records of a grouping, for example records that all happen on the same visit and time, the argument flag_all can be set to TRUE. Otherwise, new_var is set to NA. Thus, the direction of "worst" is considered fixed for all parameters in the dataset depending on the order and the mode, i.e. for every parameter the first or last record will be flagged across the whole dataset.

Value

The input dataset with the new flag variable added

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()
Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
example_vs <- tribble(
  ~USUBJID, ~VSTESTCD, ~VISIT, ~VISITNUM, ~VSTPTNUM, ~VSSTRESN,
  "1001", "DIABP", "SCREENING", 1, 10, 64,
  "1001", "DIABP", "SCREENING", 1, 11, 66,
  "1001", "DIABP", "BASELINE", 2, 100, 68,
  "1001", "DIABP", "BASELINE", 2, 101, 68,
  "1001", "DIABP", "WEEK 2", 3, 200, 72,
  "1001", "DIABP", "WEEK 2", 3, 201, 71,
  "1001", "DIABP", "WEEK 4", 4, 300, 70,
  "1001", "DIABP", "WEEK 4", 4, 301, 70
)

# Flag last value for each patient, test, and visit, baseline observations are ignored
example_vs %>%
  restrict_derivation(
    derivation = derive_var_extreme_flag,
    args = params(
      by_vars = exprs(USUBJID, VSTESTCD, VISIT),
      order = exprs(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 <- 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", "WEEK 2", as.Date("2021-04-30"), 12.0, NA,
  "TEST01", "PAT02", "PARAM02", "SCREENING", as.Date("2021-04-27"), 15.0, "AVERAGE",
  "TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-25"), 14.0, NA,
)
```
derive_var_extreme_flag

"TEST01", "PAT02", "PARAM02", "WEEK 1", as.Date("2021-04-27"), 10.0, NA,
"TEST01", "PAT02", "PARAM02", "BASELINE", as.Date("2021-04-30"), 12.0, NA
)

# Last observation
restrict_derivation(  
  input,  
  derivation = derive_var_extreme_flag,  
  args = params(    
    by_vars = exprs(USUBJID, PARAMCD),    
    order = exprs(ADT),    
    new_var = ABLFL,    
    mode = "last"  
  ),  
  filter = AVISIT == "BASELINE"  )

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

# Worst observation - Direction = Lo
restrict_derivation(  
  input,  
  derivation = derive_var_extreme_flag,  
  args = params(    
    by_vars = exprs(USUBJID, PARAMCD),    
    order = exprs(desc(AVAL), ADT),    
    new_var = ABLFL,    
    mode = "last"  
  ),  
  filter = AVISIT == "BASELINE"  )

# Average observation
restrict_derivation(  
  input,  
  derivation = derive_var_extreme_flag,  
  args = params(    
    by_vars = exprs(USUBJID, PARAMCD),    
    order = exprs(ADT, desc(AVAL)),    
    new_var = ABLFL,    
    mode = "last"  
  ),
filter = AVISIT == "BASELINE" & DTYPE == "AVERAGE"
)

# OCCURDS Examples
table <- tribble(
  ~USUBJID, ~AEBODSYS, ~AEDECOD, ~AESEV, ~AESTDY, ~AESEQ,
  "1015", "GENERAL DISORDERS", "ERYTHEMA", "MILD", 2, 1,
  "1015", "GENERAL DISORDERS", "PRURITUS", "MILD", 2, 2,
  "1015", "GI DISORDERS", "DIARRHOEA", "MILD", 8, 3,
  "1023", "CARDIAC DISORDERS", "AV BLOCK", "MILD", 22, 4,
  "1023", "SKIN DISORDERS", "ERYTHEMA", "MILD", 3, 1,
  "1023", "SKIN DISORDERS", "ERYTHEMA", "SEVERE", 5, 2,
  "1023", "SKIN DISORDERS", "ERYTHEMA", "MILD", 8, 3
)

# Most severe AE first occurrence per patient
table <- table
  .%>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = exprs(USUBJID),
    order = exprs(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first"
  )
  %>%
  arrange(USUBJID, AESTDY, AESEQ)
  %>%
  select(USUBJID, AEDECOD, AESEV, AESTDY, AOCCIFL)

# Most severe AE first occurrence per patient (flag all cases)
table <- table
  .%>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = exprs(USUBJID),
    order = exprs(TEMP_AESEVN, AESTDY),
    mode = "first",
    flag_all = TRUE
  )
  %>%
  arrange(USUBJID, AESTDY)
  %>%
  select(USUBJID, AEDECOD, AESEV, AESTDY, AOCCIFL)

# Most severe AE first occurrence per patient per body system
table <- table
  .%>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = exprs(USUBJID),
    order = exprs(TEMP_AESEVN, AESTDY),
    mode = "first"
  )
  %>%
  arrange(USUBJID, AESTDY)
  %>%
  select(USUBJID, AEDECOD, AESEV, AESTDY, AOCCIFL)
Derive a flag based on an existing flag

**Description**

Derive a flag which depends on other observations of the dataset. For example, flagging events which need to be confirmed by a second event.

**Usage**

```r
derive_var_joined_exist_flag(
  dataset,
  by_vars,
  order,
  new_var,
  tmp_obs_nr_var = NULL,
  join_vars,
  join_type,
  first_cond = NULL,
  filter,
  true_value = "Y",
  false_value = NA_character_,
  check_type = "warning"
)
```

**Arguments**

- **dataset**  
  Input dataset  
  The variables specified by the `by_vars` and `join_vars` parameter are expected.

- **by_vars**  
  By variables  
  The specified variables are used as by variables for joining the input dataset with itself.

- **order**  
  Order  
  The observations are ordered by the specified order.

- **new_var**  
  New variable  
  The specified variable is added to the input dataset.
The specified variable is added to the input dataset and set to the observation number with respect to order. For each by group (\texttt{by_vars}) the observation number starts with 1. The variable can be used in the conditions (\texttt{filter, first_cond}). It is not included in the output dataset. It can be used to flag consecutive observations or the last observation (see last example below).

\textbf{join_vars} \hspace{1cm} Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to flag all observations with \texttt{AVALC == \"Y\"} and \texttt{AVALC == \"Y\"} for at least one subsequent visit \texttt{join_vars = exprs(AVALC, AVISITN)} and \texttt{filter = AVALC == \"Y\" & AVALC.join == \"Y\" & AVISITN < AVISITN.join} could be specified.

The *.join variables are not included in the output dataset.

\textbf{join_type} \hspace{1cm} Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if \texttt{join_type = \"after\"} is specified all observations after the original observations are kept.

For example for confirmed response or BOR in the oncology setting or confirmed deterioration in questionnaires the confirmatory assessment must be after the assessment to be flagged. Thus \texttt{join_type = \"after\"} could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation to be flagged. For example, to flag AEs occurring on or after seven days before a COVID AE. Thus \texttt{join_type = \"all\"} could be used.

\textit{Permitted Values:} \"before\", \"after\", \"all\"

\textbf{first_cond} \hspace{1cm} Condition for selecting range of data

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if \texttt{filter} contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example see the third example below.

\textbf{filter} \hspace{1cm} Condition for selecting observations

The filter is applied to the joined dataset for flagging the confirmed observations. The condition can include summary functions. The joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example, \texttt{filter = AVALC == \"CR\" & all(AVALC.join \texttt{\%in\% c(\"CR\", \"NE\")) \& count_vals(var = AVALC.join, \texttt{val = \"NE\") <= 1 selects observations with response \"CR\" and for all observations up to the confirmation observation the response is \"CR\" or \"NE\" and there is at most one \"NE\".}

\textbf{true_value} \hspace{1cm} Value of new_var for flagged observations

\textit{Default: \"Y\"}

\textbf{false_value} \hspace{1cm} Value of new_var for observations not flagged

\textit{Default: NA_character_}
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
An example usage might be flagging if a patient received two required medications within a certain
timeframe of each other.

In the oncology setting, for example, the function could be used to flag if a response value can
be confirmed by an other assessment. This is commonly used in endpoints such as best overall
response.

The following steps are performed to produce the output dataset.

**Step 1:**
The input dataset is joined with itself by the variables specified for by_vars. From the right hand
side of the join only the variables specified for join_vars are kept. The suffix "join" is added to
these variables.
For example, for by_vars = USUBJID, join_vars = exprs(AVISITN, AVALC) and input dataset

```r
# A tibble: 2 x 4
USUBJID AVISITN AVALC AVAL
<chr> <dbl> <chr> <dbl>
1 1 Y 1
1 2 N 0
```

the joined dataset is

```r
A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join
<chr> <dbl> <chr> <dbl> <chr> <chr>
1 1 Y 1 1 Y
1 1 Y 1 2 N
1 2 N 0 1 Y
1 2 N 0 2 N
```

**Step 2:**
The joined dataset is restricted to observations with respect to join_type and order.
The dataset from the example in the previous step with join_type = "after" and order = exprs(AVISITN)
is restricted to

```r
A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join
<chr> <dbl> <chr> <dbl> <chr> <chr>
1 1 Y 1 2 N
```
Step 3:
If first_cond is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where first_cond is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

Step 4:
The joined dataset is grouped by the observations from the input dataset and restricted to the observations fulfilling the condition specified by filter.

Step 5:
The first observation of each group is selected

Step 6:
The variable specified by new_var is added to the input dataset. It is set to true_value for all observations which were selected in the previous step. For the other observations it is set to false_value.

Value
The input dataset with the variable specified by new_var added.

See Also
filter_joined()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

Examples
library(tibble)
library(admiral)

# flag observations with a duration longer than 30 and # at, after, or up to 7 days before a COVID AE (ACOVFL == "Y")
adae <- tribble(~USUBJID, ~ADY, ~ACOVFL, ~ADURN,
"1", 10, "N", 1,
"1", 21, "N", 50,
"1", 23, "Y", 14,
"1", 32, "N", 31,
"1", 42, "N", 20,
"2", 11, "Y", 13,
"2", 23, "N", 2,
"3", 13, "Y", 12,
"4", 14, "N", 32,
"4", 21, "N", 41
)
derive_var_joined_exist_flag(
  adae,
  new_var = ACOVFL,
  by_vars = exprs(USUBJID),
  join_vars = exprs(ACOVFL, ADY),
  join_type = "all",
  order = exprs(ADY),
  filter = ADURN > 30 & ACOVFL.join == "Y" & ADY >= ADY.join - 7
)

# flag observations with AVALC == "Y" and AVALC == "Y" at one subsequent visit
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1", 1, "Y",
  "1", 2, "N",
  "1", 3, "Y",
  "1", 4, "N",
  "2", 1, "Y",
  "2", 2, "N",
  "3", 1, "Y",
  "4", 1, "N",
  "4", 2, "N",
)

derive_var_joined_exist_flag(
  data,
  by_vars = exprs(USUBJID),
  new_var = CONFFL,
  join_vars = exprs(AVALC, AVISITN),
  join_type = "after",
  order = exprs(AVISITN),
  filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join
)

# select observations with AVALC == "CR", AVALC == "CR" at a subsequent visit,
# only "CR" or "NE" in between, and at most one "NE" in between
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1", 1, "PR",
  "1", 2, "CR",
  "1", 3, "NE",
  "1", 4, "CR",
  "1", 5, "NE",
  "2", 1, "CR",
  "2", 2, "PR",
  "2", 3, "CR",
  "3", 1, "CR",
  "4", 1, "CR",
  "4", 2, "NE",
  "4", 3, "NE",
  "4", 4, "CR",
  "4", 5, "PR"
)
derive_var_joined_exist_flag(data, by_vars = exprs(USUBJID), join_vars = exprs(AVALC), join_type = "after", order = exprs(AVISITN), new_var = CONFFL, first_cond = AVALC.join == "CR", filter = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1)

# flag observations with AVALC == "PR", AVALC == "CR" or AVALC == "PR"
# at a subsequent visit at least 20 days later, only "CR", "PR", or "NE"
# in between, at most one "NE" in between, and "CR" is not followed by "PR"

derive_var_joined_exist_flag(data, by_vars = exprs(USUBJID), join_vars = exprs(AVALC, ADY), join_type = "after", order = exprs(ADY), new_var = CONFFL, first_cond = AVALC.join %in% c("CR", "PR") & ADY.join - ADY >= 20, filter = AVALC == "PR" & all(AVALC.join %in% c("CR", "PR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1 & (min_cond(var = ADY.join, cond = AVALC.join == "CR") > maxCond(var = ADY.join, cond = AVALC.join == "PR") | count_vals(var = AVALC.join, val = "CR") == 0)
derive_var_last_dose_amt

Derive Last Dose Amount

Description

Add a variable for dose amount from the last dose to the input dataset.

Note: This is a wrapper function for the function derive_vars_last_dose().

[Deprecated]

This function is deprecated, please use derive_vars_joined() instead.

Usage

derive_var_last_dose_amt(
    dataset,
    dataset_ex, filter_ex = NULL,
    by_vars = exprs(STUDYID, USUBJID),
    dose_id = exprs(),
    dose_date,
derive_var_last_dose_amt

```r
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 `rlang::exprs`).

- **dose_id**
  - Variables to identify unique dose (created by `rlang::exprs`). Defaults to empty `exprs()`.

- **dose_date**
  - The EX dose date variable. A date or date-time object is expected.

- **analysis_date**
  - The analysis date variable. A date or date-time object is expected.

- **single_dose_condition**
  - The condition for checking if `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 `exprs()` listing the traceability variables, e.g. `exprs(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`.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function `create_single_dose_dataset()` can be used to generate single doses from aggregate dose information and satisfy `single_dose_condition`.

**Value**

Input dataset with additional column `new_var`. 
**derive_var_last_dose_date**

See Also

Other deprecated: derive_var_basetype(), derive_var_confirmation_flag(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character(), derive_vars_last_dose()

---

### Derive Last Dose Date-Time

**Description**

Add a variable for the dose date or datetime of the last dose to the input dataset.

**Note:** This is a wrapper function for the function derive_vars_last_dose().

**[Deprecated]**

This function is deprecated, please use derive_vars_joined() instead.

**Usage**

```r
derive_var_last_dose_date(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = exprs(STUDYID, USUBJID),
  dose_id = exprs(),
  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 `rlang::exprs`).
- **dose_id**: Variables to identify unique dose (created by `rlang::exprs`). Defaults to empty `exprs()`. 
derive_var_last_dose_grp

Dose date variable. A date or date-time object is expected.

analysis_date

The analysis date variable. A date or date-time object is expected.

single_dose_condition

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

Details

The last dose date is derived as the maximum dose date where the dose_date is lower to or equal to the analysis_date per by_vars for each observation in dataset. When output_datetime is TRUE and time is missing, then the last dose date time is imputed to 00:00:00. However, if date is missing, then no imputation is done.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function create_single_dose_dataset() can be used to generate single doses from aggregate dose information and satisfy single_dose_condition.

Value

Input dataset with additional column new_var.

See Also

Other deprecated: derive_param_extreme_event(), derive_var_basetype(), derive_var_confirmation_flag(), derive_var_last_dose_amt(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character(), derive_vars_last_dose()
**derive_var_last_dose_grp**

**Usage**

```r
derive_var_last_dose_grp(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = exprs(STUDYID, USUBJID),
  dose_id = exprs(),
  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 `rlang::exprs`).
- **dose_id**: Variables to identify unique dose (created by `rlang::exprs`). Defaults to empty `exprs()`.
- **dose_date**: The EX dose date variable. A date or date-time object is expected.
- **analysis_date**: The analysis date variable. A date or date-time object is expected.
- **single_dose_condition**: The condition for checking if `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.
merge_categorization

Description

[Deprecated]

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

Merge a categorization variable from a dataset to the input dataset. The observations to merge can be selected by a condition and/or selecting the first or last observation for each by group.
**Usage**

```r
derive_var_merged_cat(
    dataset,
    dataset_add,
    by_vars,
    order = NULL,
    new_var,
    source_var,
    cat_fun,
    filter_add = NULL,
    mode = NULL,
    missing_value = NA_character_
)
```

**Arguments**

- **dataset**
  - Input dataset
  - The variables specified by the `by_vars` argument are expected.

- **dataset_add**
  - Additional dataset
  - The variables specified by the `by_vars`, the `source_var`, and the `order` argument are expected.

- **by_vars**
  - Grouping variables
  - The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. Variables from the additional dataset can be renamed by naming the element, i.e., `by_vars = exprs(<name in input dataset> = <name in additional dataset>)`, similar to the dplyr joins.
  - **Permitted Values**: list of variables created by `exprs()`

- **order**
  - Sort order
  - If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.
  - Variables defined by the `new_vars` argument can be used in the sort order.
  - **Permitted Values**: list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or `NULL`

- **new_var**
  - New variable
  - The specified variable is added to the additional dataset and set to the categorized values, i.e., `cat_fun(<source variable>)`.

- **source_var**
  - Source variable

- **cat_fun**
  - Categorization function
  - A function must be specified for this argument which expects the values of the source variable as input and returns the categorized values.

- **filter_add**
  - Filter for additional dataset (dataset_add)
  - Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered.
Variables defined by the `new_vars` argument can be used in the filter condition.

Permitted Values: a condition

mode
Selection mode
Determines if the first or last observation is selected. If the `order` argument is specified, `mode` must be non-null.
If the `order` argument is not specified, the `mode` argument is ignored.

Permitted Values: "first", "last", NULL

missing_value
Values used for missing information
The new variable is set to the specified value for all by groups without observations in the additional dataset.

Default: NA_character_

Details
1. The additional dataset is restricted to the observations matching the `filter_add` condition.
2. The categorization variable is added to the additional dataset.
3. If `order` is specified, for each by group the first or last observation (depending on `mode`) is selected.
4. The categorization variable is merged to the input dataset.

Value
The output dataset contains all observations and variables of the input dataset and additionally the variable specified for `new_var` derived from the additional dataset (`dataset_add`).

See Also
Other deprecated: `derive_param_extreme_event()`, `derive_var_basetype()`, `derive_var_confirmation_flag()`, `derive_var_last_dose_amt()`, `derive_var_last_dose_date()`, `derive_var_last_dose_grp()`, `derive_var_merged_character()`, `derive_vars_last_dose()`

---

**derive_var_merged_character**

*Merge a Character Variable*

**Description**

[Deprecated]
This function is deprecated, please use `derive_vars_merged()` instead.

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

derive_var_merged_character(
    dataset,
    dataset_add,
    by_vars,
    order = NULL,
    new_var,
    source_var,
    case = NULL,
    filter_add = NULL,
    mode = NULL,
    missing_value = NA_character_
)

Arguments

dataset Input dataset
    The variables specified by the by_vars argument are expected.

dataset_add Additional dataset
    The variables specified by the by_vars, the source_var, and the order argument are expected.

by_vars Grouping variables
    The input dataset and the selected observations from the additional dataset are merged by the specified by variables. The by variables must be a unique key of the selected observations. Variables from the additional dataset can be renamed by naming the element, i.e., by_vars = exprs(<name in input dataset> = <name in additional dataset>) similar to the dplyr joins.

Permitted Values: list of variables created by exprs()

order Sort order
    If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.
    Variables defined by the new_vars argument can be used in the sort order.

Permitted Values: list of expressions created by exprs(), e.g., exprs(AJT, desc(AVAL)) or NULL

new_var New variable
    The specified variable is added to the additional dataset and set to the transformed value with respect to the case argument.

source_var Source variable

case Change case
    Changes the case of the values of the new variable.

Default: NULL

Permitted Values: NULL, "lower", "upper", "title"
derive_var_merged_exist_flag

**filter_add**  
Filter for additional dataset (dataset_add)  
Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered. Variables defined by the new_vars argument can be used in the filter condition.  
**Permitted Values:** a condition

**mode**  
Selection mode  
Determines if the first or last observation is selected. If the order argument is specified, mode must be non-null. If the order argument is not specified, the mode argument is ignored.  
**Permitted Values:** "first", "last", NULL

**missing_value**  
Values used for missing information  
The new variable is set to the specified value for all by groups without observations in the additional dataset.  
**Default:** NA_character_  
**Permitted Value:** A character scalar

**Details**

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The (transformed) character variable is added to the additional dataset.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The character variable is merged to the input dataset.

**Value**

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

**See Also**

Other deprecated: derive_param_extreme_event(), derive_var_basetype(), derive_var_confirmation_flag(), derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_vars_last_dose()

derive_var_merged_exist_flag

*Merge an Existence Flag*

**Description**

Adds a flag variable to the input dataset which indicates if there exists at least one observation in another dataset fulfilling a certain condition.  
**Note:** This is a wrapper function for the more generic derive_vars_merged().
derive_var_merged_exist_flag

dataset, dataset_add, by_vars, new_var, condition,
true_value = "Y",
false_value = NA_character_,
missing_value = NA_character_,
filter_add = NULL
)

Arguments

dataset Input dataset
The variables specified by the by_vars argument are expected.
dataset_add Additional dataset
The variables specified by the by_vars argument are expected.
by_vars Grouping variables
Permitted Values: list of variables
new_var New variable
The specified variable is added to the input dataset.
condition Condition
The condition is evaluated at the additional dataset (dataset_add). For all by
groups where it evaluates as TRUE at least once the new variable is set to the true
value (true_value). For all by groups where it evaluates as FALSE or NA for all
observations the new variable is set to the false value (false_value). The new
variable is set to the missing value (missing_value) for by groups not present
in the additional dataset.
true_value True value
Default: "Y"
false_value False value
Default: NA_character_
missing_value Values used for missing information
The new variable is set to the specified value for all by groups without observa-
tions in the additional dataset.
Default: NA_character_
Permitted Value: A character scalar
filter_add Filter for additional data
Only observations fulfilling the specified condition are taken into account for
flagging. If the argument is not specified, all observations are considered.
Permitted Values: a condition
derive_var_merged_exist_flag

Details

1. The additional dataset is restricted to the observations matching the `filter_add` condition.
2. The new variable is added to the input dataset and set to the true value (`true_value`) if for the by group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE. It is set to the false value (`false_value`) if for the by group at least one observation exists and for all observations the condition evaluates to FALSE or NA. Otherwise, it is set to the missing value (`missing_value`).

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_summary()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_joined()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `derive_vars_transposed()`, `get_summary_records()`

Examples

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

dm <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~AGE, ~AGEU,
  "PILOT01", "DM", "01-1028", 71, "YEARS",
  "PILOT01", "DM", "04-1127", 84, "YEARS",
  "PILOT01", "DM", "06-1049", 60, "YEARS"
)

ae <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~AETERM, ~AEREL,
  "PILOT01", "AE", "01-1028", "ERYTHEMA", "POSSIBLE",
  "PILOT01", "AE", "01-1028", "PRURITUS", "PROBABLE",
  "PILOT01", "AE", "06-1049", "SYNCOPE", "POSSIBLE",
  "PILOT01", "AE", "06-1049", "SYNCOPE", "PROBABLE"
)

derive_var_merged_exist_flag(
  dm,
  dataset_add = ae,
  by_vars = exprs(STUDYID, USUBJID),
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, AERELFL)

vs <- tribble(
```

derive_var_merged_summary

Merge a Summary Variable

Description

Merge a summary variable from a dataset to the input dataset.

Note: This is a wrapper function for the more generic `derive_vars_merged`.

Usage

```r
derive_var_merged_summary(
  dataset,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  false_value = "N",
  missing_value = "M"
)
```

**derive_var_merged_summary**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input dataset. The variables specified by the by_vars argument are expected.</td>
</tr>
<tr>
<td>dataset_add</td>
<td>Additional dataset. The variables specified by the by_vars and the analysis_var arguments are expected.</td>
</tr>
<tr>
<td>by_vars</td>
<td>Grouping variables. The values of analysis_var are summarized by the specified variables. The summarized values are merged to the input dataset (dataset) by the specified by variables. <em>Permitted Values:</em> list of variables created by exprs()</td>
</tr>
<tr>
<td>new_var</td>
<td>Variable to add. The specified variable is added to the input dataset (dataset) and set to the summarized values.</td>
</tr>
<tr>
<td>filter_add</td>
<td>Filter for additional dataset (dataset_add). Only observations fulfilling the specified condition are taken into account for summarizing. If the argument is not specified, all observations are considered. <em>Permitted Values:</em> a condition</td>
</tr>
<tr>
<td>analysis_var</td>
<td>Analysis variable. The values of the specified variable are summarized by the function specified for summary_fun.</td>
</tr>
<tr>
<td>summary_fun</td>
<td>Summary function. The specified function that takes as input 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).</td>
</tr>
</tbody>
</table>

**Details**

1. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
2. The values of the analysis variable (analysis_var) are summarized by the summary function (summary_fun) for each by group (by_vars) in the additional dataset (dataset_add).
3. The summarized values are merged to the input dataset as a new variable (new_var). For observations without a matching observation in the additional dataset the new variable is set to NA. Observations in the additional dataset which have no matching observation in the input dataset are ignored.

**Value**

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for new_var.
See Also

```r
derive_summary_records(), get_summary_records()
```

General Derivation Functions for all ADaMs that returns variable appended to dataset:

- `derive_var_extreme_flag()`,
- `derive_var_joined_exist_flag()`,
- `derive_var_merged_exist_flag()`,
- `derive_var_obs_number()`,
- `derive_var_relative_flag()`,
- `derive_vars_joined()`,
- `derive_vars_merged_lookup()`,
- `derive_vars_merged()`,
- `derive_vars_transposed()`,
- `get_summary_records()`

Examples

```r
library(tibble)

# Add a variable for the mean of AVAL within each visit
adbds <- tribble(
  ~USUBJID, ~AVISIT, ~ASEQ, ~AVAL,
  "1",  "WEEK 1", 1, 10,
  "1",  "WEEK 1", 2, NA,
  "1",  "WEEK 2", 3, NA,
  "1",  "WEEK 3", 4, 42,
  "1",  "WEEK 4", 5, 12,
  "1",  "WEEK 4", 6, 12,
  "1",  "WEEK 4", 7, 15,
  "2",  "WEEK 1", 1, 21,
  "2",  "WEEK 4", 2, 22
)

derive_var_merged_summary(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(USUBJID, AVISIT),
  new_var = MEANVIS,
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE)
)

# Add a variable listing the lesion ids at baseline
adsl <- tribble(~USUBJID, "1", "2", "3")

adtr <- tribble(~USUBJID, ~AVISIT, ~LESIONID, ~ASEQ, ~AVAL,
  "1", "BASELINE", "INV-T1", 1, 10,
  "1", "BASELINE", "INV-T2", 2, NA,
  "1", "BASELINE", "INV-T3", 3, NA,
  "1", "BASELINE", "INV-T4", 4, 42,
  "1", "WEEK 1", "INV-T1", 5, 12,
  "1", "WEEK 1", "INV-T2", 6, 12,
  "1", "WEEK 1", "INV-T3", 7, 15,
  "1", "WEEK 1", "INV-T4", 8, 21,
  "1", "WEEK 1", "INV-T4", 9, 22)
```

```r
```
```
"2", "BASELINE", "INV-T1",
"2", "BASELINE", "INV-T2",
"2", "BASELINE", "INV-T3",
"2", "WEEK 1", "INV-T1",
"2", "WEEK 1", "INV-N1"
)

derive_var_merged_summary(
  adsl,
  dataset_add = adtr,
  by_vars = exprs(USUBJID),
  filter_add = AVISIT == "BASELINE",
  new_var = LESIONSBL,
  analysis_var = LESIONID,
  summary_fun = function(x) paste(x, collapse = ", ")
)
```

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| dataset   | Input dataset
| by_vars   | Grouping variables
| order     | Sort order
| new_var   | Name of variable to create

The variables specified by the order and the by_vars parameter are expected.

Permitted Values: list of variables

Within each by group the observations are ordered by the specified order.

Permitted Values: list of variables or functions of variables

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.

See Also
General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

Examples
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
~STUDYID, ~DOMAIN, ~USUBJID, ~VSTESTCD, ~VISITNUM, ~VSTPTNUM,
"PILOT01", "VS", "01-703-1182", "DIABP", 3, 815,
"PILOT01", "VS", "01-703-1182", "DIABP", 3, 816,
"PILOT01", "VS", "01-703-1182", "DIABP", 4, 815,
"PILOT01", "VS", "01-703-1182", "DIABP", 4, 816,
"PILOT01", "VS", "01-703-1182", "PULSE", 3, 815,
"PILOT01", "VS", "01-703-1182", "PULSE", 3, 816,
"PILOT01", "VS", "01-703-1182", "PULSE", 4, 815,
"PILOT01", "VS", "01-703-1182", "PULSE", 4, 816,
"PILOT01", "VS", "01-703-1182", "SYSBP", 3, 815,
"PILOT01", "VS", "01-703-1182", "SYSBP", 3, 816,
"PILOT01", "VS", "01-703-1182", "SYSBP", 4, 815,
"PILOT01", "VS", "01-703-1182", "SYSBP", 4, 816,
"PILOT01", "VS", "01-716-1229", "DIABP", 3, 815,
"PILOT01", "VS", "01-716-1229", "DIABP", 3, 816,
"PILOT01", "VS", "01-716-1229", "DIABP", 4, 815,
"PILOT01", "VS", "01-716-1229", "DIABP", 4, 816,
"PILOT01", "VS", "01-716-1229", "PULSE", 3, 815,
"PILOT01", "VS", "01-716-1229", "PULSE", 3, 816,
"PILOT01", "VS", "01-716-1229", "PULSE", 4, 815,
"PILOT01", "VS", "01-716-1229", "PULSE", 4, 816,
"PILOT01", "VS", "01-716-1229", "SYSBP", 3, 815,
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,
  ignore_time_for_ref_end_date = TRUE,
  filter_pre_timepoint = NULL,
  span_period = FALSE
)
```

Arguments

- **dataset**: Input dataset. Required columns are `start_date`, `end_date`, `ref_start_date` and `ref_end_date`
- **new_var**: On-treatment flag variable name to be created. Default is `ONTRTFL`
- **start_date**: The start date (e.g. AESDT) or assessment date (e.g. ADT) Required; A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
- **end_date**: The end date of assessment/event (e.g. AENDT) A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
Optional; Default is null. If the used and date value is missing on an observation, it is assumed the medication is ongoing and ONTRTFL is set to "Y".

**ref_start_date**
The lower bound of the on-treatment period Required; A date or date-time object column is expected.
Refer to derive_vars.dt() to impute and derive a date from a date character vector to a date object.

**ref_end_date**
The upper bound of the on-treatment period A date or date-time object column is expected.
Refer to derive_vars.dt() to impute and derive a date from a date character vector to a date object.
Optional; This can be null and everything after ref_start_date will be considered on-treatment. Default is NULL.

**ref_end_window**
A window to add to the upper bound ref_end_date measured in days (e.g. 7 if 7 days should be added to the upper bound) Optional; default is 0.

**ignore_time_for_ref_end_date**
If the argument is set to TRUE, the time part is ignored for checking if the event occurred more than ref_end_window days after reference end date.
*Permitted Values:* TRUE, FALSE

**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 logical scalar. If TRUE, events that started prior to the ref_start_date and are ongoing or end after the ref_start_date are flagged as "Y". Optional; default is FALSE.

**Details**
On-Treatment is calculated by determining whether the assessment date or start/stop dates fall between 2 dates. The following logic is used to assign on-treatment = "Y":

1. start_date is missing and ref_start_date is non-missing
2. No timepoint filter is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date
3. A timepoint is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date and the filter provided in filter_pre_timepoint is not true.
4. ref_end_date is not provided and ref_start_date < start_date
5. ref_end_date is provided and ref_start_date < start_date <= ref_end_date + ref_end_window.

If the end_date is provided and the end_date < ref_start_date then the ONTRTFL is set to NULL. This would be applicable to cases where the start_date is missing and ONTRTFL has been assigned as "Y" above.
If the span_period is TRUE, this allows the user to assign ONTRTFL as "Y" to cases where the record started prior to the ref_start_date and was ongoing or ended after the ref_start_date.

Any date imputations needed should be done prior to calling this function.

Value

The input dataset with an additional column named ONTRTFL with a value of "Y" or NA

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_atoxgr(), derive_var_base(), derive_var_chg(), derive_var_pchg(), derive_var_shift()

Examples

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

advs <- 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 <- 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 <- tribble(  
~USUBJID, ~ADTM, ~TRTSDTM, ~TRTEDTM,  
"P01", ymd_hm("2020-01-02T12:00"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),  
"P02", ymd("2020-01-01"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),  
"P03", ymd("2019-12-31"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),


```r
derive_var_pchg <- function(advs, 
                           new_var = "PCHG", 
                           start_date = ADTM, 
                           ref_start_date = TRTSDTM, 
                           ref_end_date = TRTEDTM, 
                           span_period = TRUE, 
                           filter_pre_timepoint = TPT == "PRE") 
{
  derive_var_ontrtfl( 
    advs, 
    start_date = ADTM, 
    ref_start_date = TRTSDTM, 
    ref_end_date = TRTEDTM, 
    filter_pre_timepoint = TPT == "PRE"
  )
  derive_var_ontrtfl( 
    advs, 
    start_date = ASTDT, 
    end_date = AENDT, 
    ref_start_date = TRTSDT, 
    ref_end_date = TRTEDT, 
    ref_end_window = 60, 
    span_period = TRUE
  )
  derive_var_ontrtfl( 
    advs, 
    new_var = ONTR01FL, 
    start_date = ASTDT, 
    end_date = AENDT, 
    ref_start_date = AP01SDT, 
    ref_end_date = AP01EDT, 
    span_period = TRUE
  )
}
```

**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

See Also

derive_var_chg()

derive_basetype_records()

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_atoxgr(), derive_var_base(), derive_var_chg(), derive_var_ontrtf1(), derive_var_shift()

Examples

```
library(tibble)

advs <- 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_relative_flag

Flag Observations Before or After a Condition is Fulfilled

Description

Flag all observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to flag for each subject all observations before the first disease progression or to flag all AEs after a specific AE.
Usage

derive_var_relative_flag(
    dataset,
    by_vars,
    order,
    new_var,
    condition,
    mode,
    selection,
    inclusive,
    flag_no_ref_groups = TRUE,
    check_type = "warning"
)

Arguments

dataset  Input dataset
The variables specified by the order and the by_vars argument are expected.

by_vars  Grouping variables
Permitted Values: list of variables created by exprs()

order  Sort order
Within each by group the observations are ordered by the specified order.
Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

new_var  New variable
The variable is added to the input dataset and set to "Y" for all observations before or after the condition is fulfilled. For all other observations it is set to NA.

condition  Condition for Reference Observation
The specified condition determines the reference observation. In the output dataset all observations before or after (selection argument) the reference observation are flagged.

mode  Selection mode (first or last)
If "first" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the first time is flagged in the output dataset. If "last" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the last time is flagged in the output dataset.
Permitted Values: "first", "last"

selection  Flag observations before or after the reference observation?
Permitted Values: "before", "after"

inclusive  Flag the reference observation?
Permitted Values: TRUE, FALSE
flag_no_ref_groups
Should by groups without reference observation be flagged?
Permitted Values: TRUE, FALSE

check_type Check uniqueness?
If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.
Permitted Values: "none", "warning", "error"

Details
For each by group (by_vars argument) the observations before or after (selection argument) the observations where the condition (condition argument) is fulfilled the first or last time (order argument and mode argument) is flagged in the output dataset.

Value
The input dataset with the new variable (new_var) added

See Also
General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

Examples
library(tibble)
library(dplyr, warn.conflicts = FALSE)

# Flag all AEs after the first COVID AE
adae <- tribble(~USUBJID, ~ASTDY, ~ACOVFL, ~AESEQ,

"1", 2, NA, 1,
"1", 5, "Y", 2,
"1", 5, NA, 3,
"1", 17, NA, 4,
"1", 27, "Y", 5,
"1", 32, NA, 6,
"2", 8, NA, 1,
"2", 11, NA, 2,
)

derive_var_relative_flag(
  adae,
  by_vars = exprs(USUBJID),
  order = exprs(ASTDY, AESEQ),
  new_var = PSTCOVFL,
  condition = ACOVFL == "Y",
  mode = "first",
)


derive_var_relative_flag

```
selection = "after",
inclusive = FALSE,
flag_no_ref_groups = FALSE
)

response <- tribble(~USUBJID, ~AVISITN, ~AVALC,
                      "1", 0, "PR",
                      "1", 1, "CR",
                      "1", 2, "CR",
                      "1", 3, "SD",
                      "1", 4, "NE",
                      "2", 0, "SD",
                      "2", 1, "PD",
                      "2", 2, "PD",
                      "3", 0, "SD",
                      "4", 0, "SD",
                      "4", 1, "PR",
                      "4", 2, "PD",
                      "4", 3, "SD",
                      "4", 4, "PR"
)

# Flag observations up to first PD for each patient
response %>%
derive_var_relative_flag(
  by_vars = exprs(USUBJID),
  order = exprs(AVISITN),
  new_var = ANL02FL,
  condition = AVALC == "PD",
  mode = "first",
  selection = "before",
  inclusive = TRUE
)

# Flag observations up to first PD excluding baseline (AVISITN = 0) for each patient
response %>%
restrict_derivation(
  derivation = derive_var_relative_flag,
  args = params(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    new_var = ANL02FL,
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  ),
  filter = AVISITN > 0
) %>%
arrange(USUBJID, AVISITN)
```
derive_var_shift  Derive Shift

Description

Derives a character shift variable containing concatenated shift in values based on user-defined pairing, e.g., shift from baseline to analysis value, shift from baseline grade to analysis grade, ...

Usage

```r
derive_var_shift(
  dataset,
  new_var,
  from_var,
  to_var,
  na_val,
  missing_value = "NULL",
  sep_val = " to "
)
```

Arguments

- `dataset` Input dataset
  The columns specified by `from_var` and the `to_var` parameters are expected.
- `new_var` Name of the character shift variable to create.
- `from_var` Variable containing value to shift from.
- `to_var` Variable containing value to shift to.
- `na_val` `Deprecated`, please use `missing_value` instead.
- `missing_value` Character string to replace missing values in `from_var` or `to_var`.
  Default: "NULL"
- `sep_val` Character string to concatenate values of `from_var` and `to_var`.
  Default: " to "

Details

`new_var` is derived by concatenating the values of `from_var` to values of `to_var` (e.g. "NORMAL to HIGH"). When `from_var` or `to_var` has missing value, the missing value is replaced by `missing_value` (e.g. "NORMAL to NULL").

Value

The input dataset with the character shift variable added
Derive Total Treatment Duration (Days)

Derives total treatment duration (days) (TRTDURD).

**Note:** This is a wrapper function for the more generic derive_vars_duration().
Usage

```r
derive_var_trtdurd(dataset, start_date = TRTSDT, end_date = TRTEDT)
```

Arguments

- **dataset**
  - Input dataset
  - The columns specified by the `start_date` and the `end_date` parameter are expected.

- **start_date**
  - The start date
  - A date or date-time object is expected.
  - Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
  - Default: TRTSDT

- **end_date**
  - The end date
  - A date or date-time object is expected.
  - Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
  - Default: TRTEDT

Details

The total treatment duration is derived as the number of days from start to end date plus one.

Value

The input dataset with TRTDURD added

See Also

- `derive_vars_duration()`
- Date/Time Derivation Functions that returns variable appended to dataset: `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_dtm()`, `derive_vars_dt()`, `derive_vars_duration()`, `derive_vars_dy()`

Examples

```r
library(tibble)
library(lubridate)

data <- tribble(
  ~TRTSDT, ~TRTEDT,
  ymd("2020-01-01"), ymd("2020-02-24")
)
derive_var_trtdurd(data)
```
Derive treatment emergent analysis flag (e.g., TRTEMFL).

**Usage**

```
derive_var_trtemfl(
  dataset,
  new_var = TRTEMFL,
  start_date = ASTDTM,
  end_date = AENDTM,
  trt_start_date = TRTSDTM,
  trt_end_date = NULL,
  end_window = NULL,
  ignore_time_for_trt_end = TRUE,
  initial_intensity = NULL,
  intensity = NULL
)
```

**Arguments**

- **dataset**: Input dataset.
  - The variables specified by `start_date`, `end_date`, `trt_start_date`, `trt_end_date`, `initial_intensity`, and `intensity` are expected.

- **new_var**: New variable.
  - `start_date`, `end_date`, `trt_start_date`, `trt_end_date`, `initial_intensity`, and `intensity`.

- **start_date**: Event start date.
  - *Permitted Values*: A symbol referring to a date or datetime variable of the input dataset.

- **end_date**: Event end date.
  - *Permitted Values*: A symbol referring to a date or datetime variable of the input dataset.

- **trt_start_date**: Treatment start date.
  - *Permitted Values*: A symbol referring to a date or datetime variable of the input dataset.

- **trt_end_date**: Treatment end date.
  - *Permitted Values*: A symbol referring to a date or datetime variable of the input dataset or NULL.

- **end_window**: If the argument is specified, events starting more than the specified number of days after end of treatment, are not flagged.
  - *Permitted Values*: A non-negative integer or NULL.
ignore_time_for_trt_end
If the argument is set to TRUE, the time part is ignored for checking if the event occurred more than end_window days after end of treatment.

Permitted Values: TRUE, FALSE

initial_intensity
Initial severity/intensity or toxicity
This derivation assumes AE data collection method as single record per AE with “initial” and “most extreme” severity/intensity recorded separately.
If the argument is specified, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged.
The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.

Permitted Values: A symbol referring to a variable of the input dataset or NULL

intensity
Severity/intensity or toxicity
If the argument is specified, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged.
The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.

Permitted Values: A symbol referring to a variable of the input dataset or NULL

Details
For the derivation of the new variable the following cases are considered in this order. The first case which applies, defines the value of the variable.

- not treated: If trt_start_date is NA, it is set to NA_character_.
- event before treatment: If end_date is before trt_start_date (and end_date is not NA), it is set to NA_character_.
- no event date: If start_date is NA, it is set to "Y" as in such cases it is usually considered more conservative to assume the event was treatment-emergent.
- event started during treatment:
  - if end_window is not specified: if start_date is on or after trt_start_date, it is set to "Y",
  - if end_window is specified: if start_date is on or after trt_start_date and start_date is on or before trt_end_date + end_window days, it is set to "Y",
- event started before treatment and (possibly) worsened on treatment:
  - if initial_intensity and intensity is specified: if initial_intensity < intensity and start_date is before trt_start_date and end_date is on or after trt_start_date or end_date is NA, it is set to "Y".
- Otherwise it is set to NA_character_.

**Value**

The input dataset with the variable specified by `new_var` added

**See Also**

OCCDS Functions: `derive_vars_atc()`, `derive_vars_query()`, `get_terms_from_db()`

**Examples**

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

adae <- expected <- tribble(
  ~USUBJID, ~ASTDTM, ~AENDTM, ~AEITOXGR, ~AETOXGR,
  # before treatment
  "1", "2021-12-13T20:15", "2021-12-15T12:45", "1", "1",
  "1", "2021-12-14T20:15", "2021-12-14T22:00", "1", "3",
  # starting before treatment and ending during treatment
  "1", "2021-12-30T20:00", "2022-01-14T11:00", "1", "3",
  "1", "2021-12-31T20:15", "2022-01-01T01:23", "1", "1",
  # starting during treatment
  "1", "2022-01-01T12:00", "2022-01-02T23:25", "3", "4",
  # after treatment
  "1", "2022-05-10T11:00", "2022-05-10T13:05", "2", "2",
  # missing dates
  "1", "", "", "3", "4",
  "1", "2021-12-30T09:00", "", "3", "4",
  "1", "2021-12-30T11:00", "", "3", "3",
  "1", "", "2022-01-04T09:00", "3", "4",
  "1", "", "2022-12-24T19:00", "3", "4",
  "1", "", "2022-06-04T09:00", "3", "4",
  # without treatment
  "2", "", "2021-12-03T12:00", "1", "2",
  "2", "2021-12-01T12:00", "2021-12-03T12:00", "1", "2",
  "2", "2021-12-06T18:00", "", "1", "2"
)

# derive TRTEMFL without considering treatment end and worsening
derive_var_trtemfl(adae) %>% select(ASTDTM, AENDTM, TRTSDTM, TRTEMFL)

# derive TRTEM2FL taking treatment end and worsening into account
derive_var_trtemfl(adae)
```
dose_freq_lookup

```r
new_var = TRTEM2FL,
trt_end_date = TRTEDTM,
end_window = 10,
initial_intensity = AEITOXGR,
intensity = AETOXGR
}) %>% select(ASTDTM, AENDTM, AEITOXGR, AETOXGR, TRTEM2FL)
```

---

**Description**

See `dplyr::desc` for details.

---

**dose_freq_lookup**

*Pre-Defined Dose Frequencies*

---

**Description**

These pre-defined dose frequencies are sourced from CDISC. The number of rows to generate using `create_single_dose_dataset()` arguments `start_date` and `end_date` is derived from `DOSE_COUNT`, `DOSE_WINDOW`, and `CONVERSION_FACTOR` with appropriate functions from `lubridate`.

**Usage**

`dose_freq_lookup`

**Format**

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 86 rows and 5 columns.

**Details**

`NCI_CODE` and `CDISC_VALUE` are included from the CDISC source for traceability.

`DOSE_COUNT` represents the number of doses received in one single unit of `DOSE_WINDOW`. For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `DOSE_WINDOW=="MONTH"` and `DOSE_COUNT==10`. Similarly, for `CDISC_VALUE=="EVERY 2 WEEKS"`, `DOSE_WINDOW=="WEEK"` and `DOSE_COUNT==0.5` (to yield one dose every two weeks).

`CONVERSION_FACTOR` is used to convert `DOSE_WINDOW` units "WEEK", "MONTH", and "YEAR" to the unit "DAY".

For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `CONVERSION_FACTOR` is 0.0329. One day of a month is assumed to be 1 / 30.4375 of a month (one day is assumed to be 1/365.25 of a year). Given only `start_date` and `end_date` in the aggregate dataset, `CONVERSION_FACTOR` is used to calculate specific dates for `start_date` and `end_date` in the resulting single dose dataset for the doses that occur. In such cases, doses are assumed to occur at evenly spaced increments over the interval.

To see the entire table in the console, run `print(dose_freq_lookup)`.
See Also

create_single_dose_dataset()

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv5, atoxgr_criteria_daids

---

**dthcaus_source**

*Create a dthcaus_source Object*

**Description**

Create a dthcaus_source Object

**Usage**

```r
dthcaus_source(
    dataset_name, 
    filter, 
    date, 
    order = NULL, 
    mode = "first", 
    dthcaus, 
    set_values_to = NULL, 
    traceability_vars = NULL
)
```

**Arguments**

- **dataset_name**: The name of the dataset, i.e. a string, used to search for the death cause.
- **filter**: An expression used for filtering dataset.
- **date**: A date or datetime variable or an expression to be used for sorting dataset.
- **order**: Sort order
  Additional variables/expressions to be used for sorting the dataset. The dataset is ordered by date and order. Can be used to avoid duplicate record warning.
  Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) or NULL
- **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, an expression, or a string literal
  If a variable name is specified, e.g., AEDECOD, it is the variable in the source dataset to be used to assign values to DTHCAUS; if an expression, e.g., str_to_upper(AEDECOD), it is evaluated in the source dataset and the results is assigned to DTHCAUS; if a string literal, e.g. "Adverse Event", it is the fixed value to be assigned to DTHCAUS.
- **set_values_to**: Variables to be set to trace the source dataset.
traceability_vars

A named list returned by `expres()` listing the traceability variables, e.g. `expres(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, numbers, symbols, or expressions referring to existing variables.

[Deprecated] Please use `set_values_to` instead.

Value

An object of class "dthcaus_source".

See Also

derive_var_dthcaus()

Source Objects: `basket_select()`, `censor_source()`, `date_source()`, `death_event`, `event_joined()`, `event_source()`, `event()`, `query()`, `records_source()`, `tte_source()`

Examples

```r
# Deaths sourced from AE
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDT,
  mode = "first",
  dthcaus = AEDECOD
)

# Deaths sourced from DS
src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH",
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)
```

---

**event Create a event Object**

**Description**

The `event` object is used to define events as input for the `derive_extreme_event()` function.
**event**

Usage

```r
event(
    dataset_name = NULL,
    condition = NULL,
    mode = NULL,
    order = NULL,
    set_values_to = NULL,
    keep_source-vars = NULL,
    description = NULL
)
```

**Arguments**

- **dataset_name**
  - Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for `source_datasets` in `derive_extreme_event()`. If the argument is not specified, the input dataset (dataset) of `derive_extreme_event()` is used.

- **condition**
  - An unquoted condition for selecting the observations, which will contribute to the extreme event. If the condition contains summary functions like `all()`, they are evaluated for each by group separately.
  - **Permitted Values**: an unquoted condition

- **mode**
  - If specified, the first or last observation with respect to order is selected for each by group.
  - **Permitted Values**: "first", "last", NULL

- **order**
  - The specified variables or expressions are used to select the first or last observation if `mode` is specified.
  - **Permitted Values**: list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or NULL

- **set_values_to**
  - A named list returned by `exprs()` defining the variables to be set for the event, e.g. `exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems")`. The values can be a symbol, a character string, a numeric value, NA or an expression.

- **keep_source-vars**
  - Variables to keep from the source dataset
  - The specified variables are kept for the selected observations. The variables specified for `by_vars` (of `derive_extreme_event()`) and created by `set_values_to` are always kept.
  - **Permitted Values**: A list of expressions where each element is a symbol or a `tidyselect` expression, e.g., `exprs(VISIT, VISITNUM, starts_with("RS"))`

- **description**
  - Description of the event
  - The description does not affect the derivations where the event is used. It is intended for documentation only.

**Value**

An object of class `event`
See Also

derive_extreme_event(), event_joined()

Source Objects: basket_select(), censor_source(), date_source(), death_event, dthcaus_source(), event_joined(), event_source(), query(), records_source(), tte_source()

---

**event_joined**

*Create a event_joined Object*

**Description**

The event_joined object is used to define events as input for the derive_extreme_event() function. This object should be used if the event does not depend on a single observation of the source dataset but on multiple observations. For example, if the event needs to be confirmed by a second observation of the source dataset.

The events are selected by calling filter_joined(). See its documentation for more details.

**Usage**

```r
event_joined(
  dataset_name = NULL,
  condition,
  order = NULL,
  join_vars,
  join_type,
  first_cond = NULL,
  set_values_to = NULL,
  keep_source_vars = NULL,
  description = NULL
)
```

**Arguments**

- **dataset_name**
  - Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for source_datasets in derive_extreme_event(). If the argument is not specified, the input dataset (dataset) of derive_extreme_event() is used.

- **condition**
  - An unquoted condition for selecting the observations, which will contribute to the extreme event.

  *Permitted Values*: an unquoted condition

- **order**
  - If specified, the specified variables or expressions are used to select the first observation.

  *Permitted Values*: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) or NULL
join_vars Variables to keep from joined dataset
The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to select all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit join_vars = exprs(AVALC, AVISITN) and filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join could be specified.
The *.join variables are not included in the output dataset.

join_type Observations to keep after joining
The argument determines which of the joined observations are kept with respect to the original observation. For example, if join_type = "after" is specified all observations after the original observations are kept.
Permitted Values: "before", "after", "all"

first_cond Condition for selecting range of data
If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the subsequent observations, all observations are removed.

set_values_to A named list returned by exprs() defining the variables to be set for the event, e.g. exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems"). The values can be a symbol, a character string, a numeric value, NA or an expression.

keep_source_vars Variables to keep from the source dataset
The specified variables are kept for the selected observations. The variables specified for by_vars (of derive_extreme_event()) and created by set_values_to are always kept.
Permitted Values: A list of expressions where each element is a symbol or a tidyselect expression, e.g., exprs(VISIT, VISITNUM, starts_with("RS")).

description Description of the event
The description does not affect the derivations where the event is used. It is intended for documentation only.

Value
An object of class event_joined

See Also
derive_extreme_event(), event()
Source Objects: basket_select(), censor_source(), date_source(), death_event, dthcaus_source(), event_source(), event(), query(), records_source(), tte_source()
event_source

Create an event_source Object

Description

event_source objects are used to define events as input for the derive_param_tte() function.

Note: This is a wrapper function for the more generic tte_source().

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 or expression providing the date of the event or censoring. A date, or
  a datetime can be specified. An unquoted symbol or expression is expected.
  Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a
  date from a date character vector to a date object.
set_values_to A named list returned by exprs() defining the variables to be set for the event
  or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR =
  "DTHDT"). The values must be a symbol, a character string, a numeric value, an
  expression, or NA.

Value

An object of class event_source, inheriting from class tte_source

See Also

derive_param_tte(), censor_source()
Source Objects: basket_select(), censor_source(), date_source(), death_event, dthcaus_source(),
event_joined(), event(), query(), records_source(), tte_source()

Examples

# Death event

event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,

"set_values_to"
example_qs

```r
set_values_to = exprs(
  EVNTDESC = "DEATH",
  SRCDOM = "ADSL",
  SRCVAR = "DTHDT"
)
)

```

---

**example_qs**  
*Example QS Dataset*

**Description**

An example QS dataset based on the examples from the CDISC ADaM Supplements Generalized Anxiety Disorder 7-Item Version 2 (GAD-7) and Geriatric Depression Scale Short Form (GDS-SF).

**Usage**

```r
example_qs
```

**Format**

An object of class `tbl_df` (inherits from `tbl, data.frame`) with 161 rows and 11 columns.

**Source**

Created by [link](https://github.com/pharmaverse/admiral/blob/main/inst/example_scripts/example_qs.R)

**See Also**

Other datasets: `admiral_adlb`, `admiral_adsl`, `ex_single`, `queries_mh`, `queries`

---

**exprs**  
*rlang exprs*

**Description**

See `rlang::exprs` for details.
**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**

```r
extend_source_datasets(source_datasets, by_vars)
```

**Arguments**

- `source_datasets`  
  Source datasets  
  A named list of datasets is expected. Each dataset must contain either all by variables or none of the by variables.

- `by_vars`  
  By variables

**Details**

1. The by groups are determined as the union of the by groups occurring in the source datasets.
2. For all source datasets which do not contain the by variables the source dataset is replaced by the cartesian product of the source dataset and the by groups.

**Value**

The list of extended source datasets

**See Also**

Other Advanced Functions: `assert_db_requirements()`, `assert_terms()`, `assert_valid_queries()`, `filter_date_sources()`, `format.basket_select()`, `list_tte_source_objects()`, `params()`, `validate_basket_select()`, `validate_query()`

**Examples**

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

adsl <- 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")
```
extract_duplicate_records

Extract Duplicate Records

**Description**

Extract Duplicate Records

**Usage**

```r
extract_duplicate_records(dataset, by_vars)
```

**Arguments**

- `dataset`  
  A data frame

- `by_vars`  
  A list of expressions created using `exprs()` identifying groups of records in which to look for duplicates

**Value**

A `data.frame` of duplicate records within dataset

**See Also**

Utilities for Dataset Checking: `get_duplicates_dataset()`, `get_many_to_one_dataset()`, `get_one_to_many_dataset()`

**Examples**

```r
data(admiral_adsl)

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

extract_duplicate_records(adsl, exprs(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

See Also
Utilities used within Derivation functions: call_user_fun(), get_not_mapped(), signal_duplicate_records()

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 exprs() 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"

See Also

Other datasets: admiral_adlb, admiral_adsl, example_qs, queries_mh, queries
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.

See Also

Other Advanced Functions: `assert_db_requirements()`, `assert_terms()`, `assert_valid_queries()`, `extend_source_datasets()`, `format.basket_select()`, `list_tte_source_objects()`, `params()`, `validate_basket_select()`, `validate_query()`

Examples

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

adsl <- 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 <- tribble(
    ~USUBJID, ~AESTDTC,  ~AESEQ, ~AEDECOD,
    "01", "2021-03-04", 1, "Cough",
    "01", "2021-03-04", 2, "Flu",
    "01", "2021-01-01", 3, "Flu"
) %>%
    mutate(
        STUDYID = "AB42",
        AESTDT = ymd(AESTDTC)
    )

ttae <- event_source(
    dataset_name = "ae",
    date = AESTDT,
    ...
)
filter_exist

Returns records that fit into existing by groups in a filtered source dataset

Description

Returns all records in the input dataset that belong to by groups that are present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return ADSL records for subjects that experienced a certain adverse event during the course of the study (as per records in ADAE).

Usage

filter_exist(dataset, dataset_add, by_vars, filter_add = NULL)

Arguments

dataset
Input dataset
The variables specified in the by_vars parameter are expected in this dataset.

dataset_add
Source dataset
The source dataset, which determines the by groups returned in the input dataset, based on the groups that exist in this dataset after being subset by filter_add.
The variables specified in the by_vars and filter_add parameters are expected in this dataset.

by_vars
Grouping variables
A list of variable names specified within exprs() is expected.

filter_add
Filter for the source dataset
The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed.
Default: NULL (i.e. no filtering will be performed)
filter_exist

Details

Returns the records in dataset which match an existing by group in dataset_add, after being filtered according to filter_add. If there are no by groups that exist in both datasets, an empty dataset will be returned.

Value

The records in the input dataset which are contained within an existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: count_vals(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

Examples

# Get demographic information about subjects who have suffered from moderate or
# severe fatigue

library(tibble)

adsl <- tribble(
  ~USUBJID, ~AGE, ~SEX,
  "01-701-1015", 63, "F",
  "01-701-1034", 77, "F",
  "01-701-1115", 84, "M",
  "01-701-1146", 75, "F",
  "01-701-1444", 63, "M"
)

adae <- tribble(
  ~USUBJID, ~AEDECOD, ~AESEV, ~AESTDTC,
  "01-701-1015", "DIARRHOEA", "MODERATE", "2014-01-09",
  "01-701-1034", "FATIGUE", "SEVERE", "2014-11-02",
  "01-701-1115", "APPLICATION SITE PRURITUS", "MODERATE", "2014-08-27",
  "01-701-1146", "FATIGUE", "MILD", "2013-01-14",
  "01-701-1146", "FATIGUE", "MODERATE", "2013-06-03"
)

filter_exist(
  dataset = adsl,
  dataset_add = adae,
  by_vars = exprs(USUBJID),
  filter_add = AEDECOD == "FATIGUE" & AESEV %in% c("MODERATE", "SEVERE")
)
**filter_extreme** | *Filter the First or Last Observation for Each By Group*

**Description**
Filters the first or last observation for each by group.

**Usage**
```r
filter_extreme(dataset, by_vars = NULL, order, mode, check_type = "warning")
```

**Arguments**
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td><strong>Input dataset</strong>&lt;br&gt;The variables specified by the order and the by_vars parameter are expected.</td>
</tr>
<tr>
<td>by_vars</td>
<td><strong>Grouping variables</strong>&lt;br&gt;<strong>Default</strong>: NULL&lt;br&gt;<strong>Permitted Values</strong>: list of variables created by <code>exprs()</code></td>
</tr>
<tr>
<td>order</td>
<td><strong>Sort order</strong>&lt;br&gt;Within each by group the observations are ordered by the specified order.&lt;br&gt;<strong>Permitted Values</strong>: list of expressions created by <code>exprs()</code>, e.g., <code>exprs(ADT, desc(AVAL))</code></td>
</tr>
<tr>
<td>mode</td>
<td><strong>Selection mode (first or last)</strong>&lt;br&gt;If &quot;first&quot; is specified, the first observation of each by group is included in the output dataset. If &quot;last&quot; is specified, the last observation of each by group is included in the output dataset.&lt;br&gt;<strong>Permitted Values</strong>: &quot;first&quot;,&quot;last&quot;</td>
</tr>
<tr>
<td>check_type</td>
<td><strong>Check uniqueness?</strong>&lt;br&gt;If &quot;warning&quot; or &quot;error&quot; 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.&lt;br&gt;<strong>Default</strong>: &quot;warning&quot;&lt;br&gt;<strong>Permitted Values</strong>: &quot;none&quot;,&quot;warning&quot;,&quot;error&quot;</td>
</tr>
</tbody>
</table>

**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
See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_joined()`, `filter_not_exist()`, `filter_relative()`, `max_cond()`, `min_cond()`

Examples

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

ex <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~EXSEQ, ~EXDOSE, ~EXTRT,
  "PILOT01", "EX", "01-1442", 1, 54, "XANO",
  "PILOT01", "EX", "01-1442", 2, 54, "XANO",
  "PILOT01", "EX", "01-1442", 3, 54, "XANO",
  "PILOT01", "EX", "01-1444", 1, 54, "XANO",
  "PILOT01", "EX", "01-1444", 2, 81, "XANO",
  "PILOT01", "EX", "05-1382", 1, 54, "XANO",
  "PILOT01", "EX", "08-1213", 1, 54, "XANO",
  "PILOT01", "EX", "10-1053", 1, 54, "XANO",
  "PILOT01", "EX", "10-1053", 2, 54, "XANO",
  "PILOT01", "EX", "10-1183", 1, 0, "PLACEBO",
  "PILOT01", "EX", "10-1183", 2, 0, "PLACEBO",
  "PILOT01", "EX", "10-1183", 3, 0, "PLACEBO",
  "PILOT01", "EX", "11-1036", 1, 0, "PLACEBO",
  "PILOT01", "EX", "11-1036", 2, 0, "PLACEBO",
  "PILOT01", "EX", "11-1036", 3, 0, "PLACEBO",
  "PILOT01", "EX", "14-1425", 1, 54, "XANO",
  "PILOT01", "EX", "15-1319", 1, 54, "XANO",
  "PILOT01", "EX", "15-1319", 2, 81, "XANO",
  "PILOT01", "EX", "16-1151", 1, 54, "XANO",
  "PILOT01", "EX", "16-1151", 2, 54, "XANO"
)

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

# Select highest dose for each patient on the active drug
ex %>%
  filter(EXPRT != "PLACEBO") %>%
  filter_extreme(
    by_vars = exprs(USUBJID),
    order = exprs(EXDOSE),
    mode = "last",
    check_type = "none"
  )
  select(USUBJID, EXTRT, EXDOSE)
```
**Description**

The function filters observation using a condition taking other observations into account. For example, it could select all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent observation. The input dataset is joined with itself to enable conditions taking variables from both the current observation and the other observations into account. The suffix ".join" is added to the variables from the subsequent observations.

An example usage might be checking if a patient received two required medications within a certain timeframe of each other.

In the oncology setting, for example, we use such processing to check if a response value can be confirmed by a subsequent assessment. This is commonly used in endpoints such as best overall response.

**Usage**

```r
filterJoined(
  dataset,
  by_vars,
  join_vars,
  join_type,
  first_cond = NULL,
  order,
  tmp_obs_nr_var = NULL,
  filter,
  check_type = "warning"
)
```

**Arguments**

- **dataset**: Input dataset
  
  The variables specified for `by_vars`, `join_vars`, and `order` are expected.

- **by_vars**: By variables
  
  The specified variables are used as by variables for joining the input dataset with itself.

- **join_vars**: Variables to keep from joined dataset
  
  The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to select all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit `join_vars = exprs(AVALC, AVISITN)` and `filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join` could be specified.
  
  The *.join variables are not included in the output dataset.
join_type Observations to keep after joining
The argument determines which of the joined observations are kept with respect to the original observation. For example, if join_type = "after" is specified all observations after the original observations are kept.
Permitted Values: "before", "after", "all"

defiltered
Condition for selecting range of data
If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the subsequent observations, all observations are removed.

order Order
The observations are ordered by the specified order.
Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

tmp_obs_nr_var Temporary observation number
The specified variable is added to the input dataset and set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. The variable can be used in the conditions (filter, first_cond). It is not included in the output dataset. It can be used to select consecutive observations or the last observation (see last example below).

filter Condition for selecting observations
The filter is applied to the joined dataset for selecting the confirmed observations. The condition can include summary functions. The joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example in the oncology setting when using this function for confirmed best overall response, filter = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1 selects observations with response "CR" and for all observations up to the confirmation observation the response is "CR" or "NE" and there is at most one "NE".

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
The following steps are performed to produce the output dataset.

Step 1:
The input dataset is joined with itself by the variables specified for by_vars. From the right hand side of the join only the variables specified for join_vars are kept. The suffix "join" is added to these variables.
For example, for by_vars = USUBJID, join_vars = exprs(AVISITN, AVALC) and input dataset
# A tibble: 2 x 4
USUBJID AVISITN AVALC AVAL  
<chr>   <dbl> <chr> <dbl>
1 Y       1 Y       1
1 N       2 N       0

the joined dataset is

A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join  
<chr>   <dbl> <chr> <dbl> <chr> <chr>
1 Y       1 Y       1 1 Y
1 Y       1 Y       1 2 N
1 N       2 N       0 1 Y
1 N       2 N       0 2 N

Step 2:
The joined dataset is restricted to observations with respect to join_type and order.
The dataset from the example in the previous step with join_type = "after" and order = exprs(AVISITN)
is restricted to

A tibble: 4 x 6
USUBJID AVISITN AVALC AVAL AVISITN.join AVALC.join  
<chr>   <dbl> <chr> <dbl> <chr> <chr>
1 Y       1 Y       1 1 Y
1 N       2 N       0 1 Y

Step 3:
If first_cond is specified, for each observation of the input dataset the joined dataset is restricted
to observations up to the first observation where first_cond is fulfilled (the observation fulfilling
the condition is included). If for an observation of the input dataset the condition is not fulfilled,
the observation is removed.

Step 4:
The joined dataset is grouped by the observations from the input dataset and restricted to the
observations fulfilling the condition specified by filter.

Step 5:
The first observation of each group is selected and the *.join variables are dropped.

Value
A subset of the observations of the input dataset. All variables of the input dataset are included in
the output dataset.

See Also
count_vals(), min_cond(), max_cond()
Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_not_exist(),
filter_relative(), max_cond(), min_cond()
Examples

```r
library(tibble)
library(admiral)

# filter observations with a duration longer than 30 and # on or after 7 days before a COVID AE (ACOVFL == "Y")
adae <- tribble(~USUBJID, ~ADY, ~ACOVFL, ~ADURN, 
                "1", 10, "N", 1, 
                "1", 21, "N", 50, 
                "1", 23, "Y", 14, 
                "1", 32, "N", 31, 
                "1", 42, "N", 20, 
                "2", 11, "Y", 13, 
                "2", 23, "N", 2, 
                "3", 13, "Y", 12, 
                "4", 14, "N", 32, 
                "4", 21, "N", 41, )

filter_joined(adae, 
              by_vars = exprs(USUBJID), 
              join_vars = exprs(ACOVFL, ADY), 
              join_type = "all", 
              order = exprs(ADY), 
              filter = ADURN > 30 & ACOVFL.join == "Y" & ADY >= ADY.join - 7 )

# filter observations with AVALC == "Y" and AVALC == "Y" at a subsequent visit
data <- tribble(~USUBJID, ~AVISITN, ~AVALC, 
                "1", 1, "Y", 
                "1", 2, "N", 
                "1", 3, "Y", 
                "1", 4, "N", 
                "2", 1, "Y", 
                "2", 2, "N", 
                "3", 1, "Y", 
                "4", 1, "N", 
                "4", 2, "N", )

filter_joined(data, 
              by_vars = exprs(USUBJID), 
              join_vars = exprs(AVALC, AVISITN), 
              join_type = "after", 
              order = exprs(AVISITN), 
              filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join )
```

# select observations with AVALC == "CR", AVALC == "CR" at a subsequent visit,  
# only "CR" or "NE" in between, and at most one "NE" in between

data <- tribble(  
  ~USUBJID, ~AVISITN, ~AVALC,  
  "1", 1, "PR",  
  "1", 2, "CR",  
  "1", 3, "NE",  
  "1", 4, "CR",  
  "1", 5, "NE",  
  "2", 1, "CR",  
  "2", 2, "PR",  
  "2", 3, "CR",  
  "3", 1, "CR",  
  "4", 1, "CR",  
  "4", 2, "NE",  
  "4", 3, "NE",  
  "4", 4, "CR",  
  "4", 5, "PR"  
)

filter_joined(  
  data,  
  by_vars = exprs(USUBJID),  
  join_vars = exprs(AVALC),  
  join_type = "after",  
  order = exprs(AVISITN),  
  first_cond = AVALC.join == "CR",  
  filter = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) &  
    count_vals(var = AVALC.join, val = "NE") <= 1  
)

# select observations with AVALC == "PR", AVALC == "CR" or AVALC == "PR"  
# at a subsequent visit at least 20 days later, only "CR", "PR", or "NE"  
# in between, at most one "NE" in between, and "CR" is not followed by "PR"

data <- tribble(  
  ~USUBJID, ~ADY, ~AVALC,  
  "1", 6, "PR",  
  "1", 12, "CR",  
  "1", 24, "NE",  
  "1", 32, "CR",  
  "1", 48, "PR",  
  "2", 3, "PR",  
  "2", 21, "CR",  
  "2", 33, "PR",  
  "3", 11, "PR",  
  "4", 7, "PR",  
  "4", 12, "NE",  
  "4", 24, "NE",  
  "4", 32, "PR",  
  "4", 55, "PR"  
)
filter_joined(
  data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC, ADY),
  join_type = "after",
  order = exprs(ADY),
  first_cond = AVALC.join %in% c("CR", "PR") & ADY.join - ADY >= 20,
  filter = AVALC == "PR" &
    all(AVALC.join %in% c("CR", "PR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1 &
    (min_cond(var = ADY.join, cond = AVALC.join == "CR") >
     max_cond(var = ADY.join, cond = AVALC.join == "PR") |
     count_vals(var = AVALC.join, val = "CR") == 0)
)

# select observations with CRIT1FL == "Y" at two consecutive visits or at the last visit
data <- tribble(
  ~USUBJID, ~AVISITN, ~CRIT1FL,
  "1", 1, "Y",
  "1", 2, "N",
  "1", 3, "Y",
  "1", 5, "N",
  "2", 1, "Y",
  "2", 3, "Y",
  "2", 5, "N",
  "3", 1, "Y",
  "4", 1, "Y",
  "4", 2, "N",
)

filter_joined(
  data,
  by_vars = exprs(USUBJID),
  tmp_obs_nr_var = tmp_obs_nr,
  join_vars = exprs(CRIT1FL),
  join_type = "all",
  order = exprs(AVISITN),
  filter = CRIT1FL == "Y" & CRIT1FL.join == "Y" &
    (tmp_obs_nr + 1 == tmp_obs_nr.join | tmp_obs_nr == max(tmp_obs_nr.join))
)

---

filter_not_exist Returns records that don't fit into existing by groups in a filtered source dataset

Description

Returns all records in the input dataset that belong to by groups that are not present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return
ADSL records for subjects that didn’t take certain concomitant medications during the course of the study (as per records in ADCM).

Usage

```
filter_not_exist(dataset, dataset_add, by_vars, filter_add = NULL)
```

Arguments

- `dataset` Input dataset
  The variables specified in the `by_vars` parameter are expected in this dataset.
- `dataset_add` Source dataset
  The source dataset, which determines the by groups returned in the input dataset, based on the groups that don’t exist in this dataset after being subset by `filter_add`. The variables specified in the `by_vars` and `filter_add` parameters are expected in this dataset.
- `by_vars` Grouping variables
  A list of variable names specified within `exprs()` is expected.
- `filter_add` Filter for the source dataset
  The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed. Default: NULL (i.e. no filtering will be performed)

Details

Returns the records in `dataset` which don’t match any existing by groups in `dataset_add`, after being filtered according to `filter_add`. If all by groups that exist in `dataset` don’t exist in `dataset_add`, an empty dataset will be returned.

Value

The records in the input dataset which are not contained within any existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_extreme()`, `filter_joined()`, `filter_relative()`, `max_cond()`, `min_cond()`

Examples

```
# Get demographic information about subjects who didn't take vitamin supplements
# during the study

library(tibble)

adsl <- tribble(
```
filter_relative

Filter the Observations Before or After a Condition is Fulfilled

Description

Filters the observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to select for each subject all observations before the first disease progression.

Usage

filter_relative(
  dataset,
  by_vars,
  order,
  condition,
  mode,
  selection,
  inclusive,
  keep_no_ref_groups = TRUE,
  check_type = "warning"
)
Arguments

dataset  Input dataset
  The variables specified by the order and the by_vars parameter are expected.

by_vars  Grouping variables
  Permitted Values: list of variables created by exprs()

order  Sort order
  Within each by group the observations are ordered by the specified order.
  Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

condition  Condition for Reference Observation
  The specified condition determines the reference observation. The output dataset contains all observations before or after (selection parameter) the reference observation.

mode  Selection mode (first or last)
  If "first" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the first time is included in the output dataset. If "last" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the last time is included in the output dataset.
  Permitted Values: "first", "last"

selection  Select observations before or after the reference observation?
  Permitted Values: "before", "after"

inclusive  Include the reference observation?
  Permitted Values: TRUE, FALSE

keep_no_ref_groups  Should by groups without reference observation be kept?
  Permitted Values: TRUE, FALSE

check_type  Check uniqueness?
  If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.
  Permitted Values: "none", "warning", "error"

Details

For each by group (by_vars parameter) the observations before or after (selection parameter) the observations where the condition (condition parameter) is fulfilled the first or last time (order parameter and mode parameter) is included in the output dataset.

Value

A dataset containing for each by group the observations before or after the observation where the condition was fulfilled the first or last time
See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_extreme()`, `filter_joined()`, `filter_not_exist()`, `max_cond()`, `min_cond()`

Examples

```r
library(tibble)

response <- tribble(~USUBJID, ~AVISITN, ~AVALC,
                      "1", 1, "PR",
                      "1", 2, "CR",
                      "1", 3, "CR",
                      "1", 4, "SD",
                      "1", 5, "NE",
                      "2", 1, "SD",
                      "2", 2, "PD",
                      "2", 3, "PD",
                      "3", 1, "SD",
                      "4", 1, "SD",
                      "4", 2, "PR",
                      "4", 3, "PD",
                      "4", 4, "SD",
                      "4", 5, "PR"
)

# Select observations up to first PD for each patient
response %>%
  filter_relative(by_vars = exprs(USUBJID),
                  order = exprs(AVISITN),
                  condition = AVALC == "PD",
                  mode = "first",
                  selection = "before",
                  inclusive = TRUE
)

# Select observations after last CR, PR, or SD for each patient
response %>%
  filter_relative(by_vars = exprs(USUBJID),
                  order = exprs(AVISITN),
                  condition = AVALC %in% c("CR", "PR", "SD"),
                  mode = "last",
                  selection = "after",
                  inclusive = FALSE
)

# Select observations from first response to first PD
response %>%
  filter_relative(by_vars = exprs(USUBJID),
                  order = exprs(AVISITN),
                  condition = AVALC == "PD",
                  mode = "first",
                  selection = "before",
                  inclusive = TRUE
)
```

format.basket_select

Returns a Character Representation of a basket_select() Object

Description

The function returns a character representation of a basket_select() object. It can be used for error messages for example.

Usage

## S3 method for class 'basket_select'
format(x, ...)

Arguments

x A basket_select() object

... Not used

Value

A character representation of the basket_select() object

See Also

basket_select()

Other Advanced Functions: assert_db_requirements(), assert_terms(), assert_valid_queries(), extend_source_datasets(), filter_date_sources(), list_tte_source_objects(), params(), validate_basket_select(), validate_query()

Examples

format(basket_select(id = 42, scope = "NARROW", type = "smq"))
get_admiral_option  Get the Value of an Admiral Option

Description
Get the Value of an Admiral Option Which Can Be Modified for Advanced Users.

Usage
get_admiral_option(option)

Arguments
option  A character scalar of commonly used admiral function inputs.
As of now, support only available for "subject_keys". See set_admiral_options() for a description of the options.

Details
This function allows flexibility for function inputs that may need to be repeated multiple times in a script, such as subject_keys.

Value
The value of the specified option.

See Also
set_admiral_options(), derive_param_exist_flag(), derive_param_tte() derive_var_dthcaus(), derive_var_extreme_dtm(), derive_vars_period(), create_period_dataset()
Other admiral_options: set_admiral_options()

Examples
library(dplyr, warn.conflicts = FALSE)
dm <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~AGE, ~AGEU,
  "PILOT01", "DM", "01-1302", 61, "YEARS",
  "PILOT01", "DM", "17-1344", 64, "YEARS"
)
vs <- tribble(
  ~STUDYID, ~DOMAIN, ~USUBJID, ~VSTESTCD, ~VISIT, ~VSTPT, ~VSSTRESN,
  "PILOT01", "VS", "01-1302", "DIABP", "BASELINE", "LYING", 76,
  "PILOT01", "VS", "01-1302", "DIABP", "BASELINE", "STANDING", 87,
  "PILOT01", "VS", "01-1302", "DIABP", "WEEK 2", "LYING", 71,
  "PILOT01", "VS", "01-1302", "DIABP", "WEEK 2", "STANDING", 79,
  "PILOT01", "VS", "17-1344", "DIABP", "BASELINE", "LYING", 88,
get_duplicates_dataset

Get Duplicate Records that Led to a Prior Error

Description

Get Duplicate Records that Led 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 session 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

See Also

Utilities for Dataset Checking: extract_duplicate_records(), get_many_to_one_dataset(), get_one_to_many_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

See Also

Utilities for Dataset Checking: extract_duplicate_records(), get_duplicates_dataset(), get_one_to_many_dataset()

Examples

library(admiraldev, warn.conflicts = FALSE)
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, exprs(SITEID), exprs(STUDYID))
)
get_not_mapped

get_many_to_one_dataset()

get_not_mapped  Get list of records not mapped from the lookup table.

Description
Get list of records not mapped from the lookup table.

Usage
get_not_mapped()

Value
A data.frame or NULL

See Also
Utilities used within Derivation functions: call_user_fun(), extract_unit(), signal_duplicate_records()

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
get_summary_records

Create Summary Records

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

```r
get_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 `exprs()` 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_rows = (AVAL > mean(AVAL, na.rm = TRUE))` will filter all AVAL values greater than mean of AVAL with in `by_vars`.
- **analysis_var** Analysis variable.
- **summary_fun** Summary function.
- **set_values_to** Values to set.
get_summary_records

- filter_rows = (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.

- 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, an expression or NA. (e.g. exprs(PARAMCD = "TDSE", PARCAT1 = "OVERALL").

Details

This function only creates derived observations and does not append them to the original dataset observations. If you would like to this instead, see the derive_summary_records() function.

Value

A data frame of derived records.

See Also

- derive_summary_records()
- General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_merged_summary(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_joined(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed()

Examples

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

adeg <- 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, "",  

...
get_summary_records

```
"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)
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = exprs(DTYPE = "AVERAGE")
)

advs <- tribble(
  ~USUBJID, ~VSSEQ, ~PARAM, ~AVAL, ~VSSTRESU, ~VISIT, ~VSDTC,
  "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.
get_summary_records(
  advs,
  by_vars = exprs(USUBJID, PARAM),
  analysis_var = AVAL,
  summary_fun = max,
  set_values_to = exprs(DTYPE = "MAXIMUM")
) %>%
get_summary_records(
  by_vars = exprs(USUBJID, PARAM),
  analysis_var = AVAL,
  summary_fun = mean,
  set_values_to = exprs(DTYPE = "AVERAGE")
)

# Sample ADEG dataset with triplicate record for only AVISIT = 'Baseline'
adeg <- 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",
)
get_terms_from_db

Get Terms from the Queries Database

Description

The function checks if all requirements to access the database are fulfilled (version and access function are available, see assert_db_requirements()), reads the terms from the database, and checks if the dataset with the terms is in the expected format (see assert_terms()).

Usage

get_terms_from_db(
  version,
  fun,
  queries,
  definition,
  expect_grpname = FALSE,
  expect_grpid = FALSE,
  i,
  temp_env
)

Arguments

version

Version

The version must be non null. Otherwise, an error is issued. The value is passed to the access function (fun).
fun  Access function
The access function must be non null. Otherwise, an error is issued. The function is called to retrieve the terms.

queries  Queries
List of all queries passed to create_query_data(). It is used for error messages.

definition  Definition of the query
The definition is passed to the access function. It defines which terms are returned.

expect_grpname  Is GRPNAME expected in the output dataset?
expect_grpid  Is GRPID expected in the output dataset?
i  Index of definition in queries
The value is used for error messages.

temp_env  Temporary environment
The value is passed to the access function.

Value
Output dataset of the access function

See Also
OCCDS Functions: derive_var_trtemfl(), derive_vars_atc(), derive_vars_query()

---

impute_dtc_dt  Impute Partial Date Portion of a '—DTC' Variable

Description
Imputation partial date portion of a '—DTC' variable based on user input.

Usage

impute_dtc_dt(
  dtc,
  highest_imputation = "n",
  date_imputation = "first",
  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-ddTh:mm:ss.
Trailing components can be omitted and - is a valid "missing" value for any component.

highest_imputation
Highest imputation level
The highest_imputation argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, \texttt{NA\_character\_} is returned. For example, for \texttt{highest\_imputation = "D"}
\texttt{"2020"} results in \texttt{NA\_character\_} because the month is missing.

If \"n\" is specified no imputation is performed, i.e., if any component is missing, \texttt{NA\_character\_} is returned.

If \"Y\" is specified, date_imputation should be \"first\" or \"last\" and min_dates or max_dates should be specified respectively. Otherwise, \texttt{NA\_character\_} is returned if the year component is missing.

Default: \"n\"

Permitted Values: \"Y\" (year, highest level), \"M\" (month), \"D\" (day), \"n\" (none, lowest level)

date_imputation
The value to impute the day/month when a datepart is missing.
A character value is expected, either a
- format with month and day specified as \"mm-dd\": e.g. \"06-15\" for the 15th of June (The year can not be specified; for imputing the year \"first\" or \"last\" together with min_dates or max_dates argument can be used (see examples)),
- or as a keyword: \"first\", \"mid\", \"last\" to impute to the first/mid/last day/month.

The argument is ignored if highest_imputation is less then \"D\".

Default: \"first\"

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

\begin{verbatim}
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
    ymd_hms("2020-12-06T12:12:12"),
    ymd_hms("2020-11-11T11:11:11")
)
returns "2020-11-11T11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

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

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

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

Value
A character vector

See Also
Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dtm()

Examples
library(lubridate)

dates <- c(
    "2019-07-18T15:25:40",
    "2019-07-18T15:25",
    "2019-07-18",
    "2019-07-18",
    "2019-02",
    "2019",
    "2019",
    "2019---07",
    ""
)

# No date imputation (highest_imputation defaulted to "n")
impute_dtc_dt(dtc = dates)

# Impute to first day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M"
)
# Same as above
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "01-01"
)

# Impute to last day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "last"
)

# Impute to mid day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "mid"
)

# Impute a date and ensure that the imputed date is not before a list of
# minimum dates
impute_dtc_dt(
  "2020-12",
  min_dates = list(
    ymd("2020-12-06"),
    ymd("2020-11-11")
  ),
  highest_imputation = "M"
)

# Impute completely missing dates (only possible if min_dates or max_dates is specified)
impute_dtc_dt(
  c("2020-12", NA_character_),
  min_dates = list(
    ymd("2020-12-06", "2020-01-01"),
    ymd("2020-11-11", NA)
  ),
  highest_imputation = "Y"
)
Description

Imputation partial date/time portion of a '--DTC' variable based on user input.

Usage

```r
impute_dtc_dtm(
  dtc,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  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`. Trailing components can be omitted and - is a valid "missing" value for any component.

- **highest_imputation**: Highest imputation level
  The `highest_imputation` argument controls which components of the DTC value are imputed if they are missing. All components up to the specified level are imputed.
  If a component at a higher level than the highest imputation level is missing, `NA_character_` is returned. For example, for `highest_imputation = "D""2020" results in `NA_character_` because the month is missing.
  If "n" is specified, no imputation is performed, i.e., if any component is missing, `NA_character_` is returned.
  If "Y" is specified, `date_imputation` should be "first" or "last" and `min_dates` or `max_dates` should be specified respectively. Otherwise, `NA_character_` is returned if the year component is missing.
  Default: "h"

- **Permitted Values**: "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

- **date_imputation**: The value to impute the day/month when a datepart is missing.
  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 (The year can not be specified; for imputing the year "first" or "last" together with `min_dates` or `max_dates` argument can be used (see examples).),
  - or as a keyword: "first", "mid", "last" to impute to the first/mid/last day/month.
The argument is ignored if highest_imputation is less than "D".
*Default*: "first".

**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.

The argument is ignored if highest_imputation = "n".
*Default*: "first".

**min_dates**

Minimum dates

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

```r
impute_dtc_dtm("2020-11",
    min_dates = list(
        ymd_hms("2020-12-06T12:12:12"),
        ymd_hms("2020-11-11T11:11:11")
    ),
    highest_imputation = "M"
)
```

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).

For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

**max_dates**

Maximum dates

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

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

**preserve**

Preserve day if month is missing and day is present

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

Permitted Values: TRUE, FALSE
*Default*: FALSE
Details

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

Value

A character vector

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dtm(), convert_dtc_to_dt(), impute_dtc_dt()
# same as above
impute_dtc_dtm(
    dtc = dates,
    highest_imputation = "M",
    date_imputation = "01-01"
)

# Impute to last day/month if date is partial
# Missing time part imputed with 23:59:59 portion
impute_dtc_dtm(
    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_dtm(
    dtc = dates,
    highest_imputation = "M",
    date_imputation = "mid"
)

# Impute a date and ensure that the imputed date is not before a list of
# minimum dates
impute_dtc_dtm(
    "2020-12",
    min_dates = list(
        ymd_hms("2020-12-06T12:12:12"),
        ymd_hms("2020-11-11T11:11:11")
    ),
    highest_imputation = "M"
)

# Impute completely missing dates (only possible if min_dates or max_dates is specified)
impute_dtc_dtm(
    c("2020-12", NA_character_),
    min_dates = list(
        ymd_hms("2020-12-06T12:12:12", "2020-01-01T01:01:01"),
        ymd_hms("2020-11-11T11:11:11", NA)
    ),
    highest_imputation = "Y"
)

---

**list_all_templates**  
*List All Available ADaM Templates*

**Description**

List All Available ADaM Templates
Usage

list_tte_source_objects

Arguments

package

Value

A character vector of all available templates

See Also

Utilities used for examples and template scripts: use_ad_template()

Examples

list_tte_source_objects()
**max_cond**

*Maximum Value on a Subset*

**Description**

The function derives the maximum value of a vector/column on a subset of entries/observations.

**Usage**

```
max_cond(var, cond)
```

**Arguments**

- **var**: A vector
- **cond**: A condition

**See Also**

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_extreme()`, `filter_joined()`, `filter_not_exist()`, `filter_relative()`, `min_cond()`

**Examples**

```r
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(~USUBJID, ~AVISITN, ~AVALC,
                "1", 1, "PR",
                "1", 2, "CR",
                "1", 3, "NE",
                "1", 4, "CR",
                "1", 5, "NE",
                "2", 1, "CR",
                "2", 2, "PR",
                "2", 3, "CR",
)
# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR

# Group by subject and find the minimum visit with a CR
first_cr_vis <- min_cond(var = AVISITN, cond = AVALC == "CR")

# Find the maximum visit with a PR after the CR visit
last_pr_vis <- max_cond(var = AVISITN, cond = AVALC == "PR")

# Indicate if PR occurred after the CR visit
pr_after_cr <- last_pr_vis > first_cr_vis

```
min_cond

Minimum Value on a Subset

Description

The function derives the minimum value of a vector/column on a subset of entries/observations.

Usage

min_cond(var, cond)

Arguments

var       A vector
cond      A condition

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond()

Examples

library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
   ~USUBJID, ~AVISITN, ~AVALC,
   "1", 1, "PR",
   "1", 2, "CR",
   "1", 3, "NE",
   "1", 4, "CR",
   "1", 5, "NE",
   "2", 1, "CR",
   "2", 2, "PR",
   "2", 3, "CR",
)

# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR

group_by(data, USUBJID) %>% mutate(
   first_cr_vis = min_cond(var = AVISITN, cond = AVALC == "CR"),
   last_pr_vis = max_cond(var = AVISITN, cond = AVALC == "PR"),
   pr_after_cr = last_pr_vis > first_cr_vis
)
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 exprs()

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 expressions

See Also

Other utils_quo: chr2vars()

Examples

negate_vars(exprs(USUBJID, STUDYID))

params  

Create a Set of Parameters

Description

Create a set of variable parameters/function arguments to be used in call_derivation().

Usage

params(...)

Arguments

... One or more named arguments
Value

An object of class \texttt{params}

See Also

call\_derivation()

Other Advanced Functions: assert\_db\_requirements(), assert\_terms(), assert\_valid\_queries(), extend\_source\_datasets(), filter\_date\_sources(), format.basket\_select(), list\_tte\_source\_objects(), validate\_basket\_select(), validate\_query()

Examples

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

adsl <- tribble(~STUDYID, ~USUBJID, ~TRTSDT, ~TRTEDT,
"PILOT01", "01-1307", NA, NA,
"PILOT01", "06-1384", "2012-09-15", "2012-09-24",
"PILOT01", "15-1085", "2013-02-16", "2013-08-18",
"PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
  mutate(across(TRTSDT:TRTEDT, as.Date))

ae <- tribble(~STUDYID, ~DOMAIN, ~USUBJID, ~AESTDTC, ~AEENDTC,
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-23", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-23", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "06-1384", "2012-09-15", "2012-09-29",
"PILOT01", "AE", "16-1298", "2013-04-22", "2013-07-06"
)

adae <- ae %>%
  select(USUBJID, AESTDTC, AEENDTC) %>%
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = exprs(TRTSDT, TRTEDT),
  )
```
In order to derive both \`ASTDT\` and \`AENDT\` in `adae`, one can use `derive_vars_dt()`

```r
adae %>%
derive_vars_dt(
  new_vars_prefix = "AST",
  dtc = AESTDTC,
  date_imputation = "first",
  min_dates = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
) %>%
derive_vars_dt(
  new_vars_prefix = "AEN",
  dtc = AEENDTC,
  date_imputation = "last",
  min_dates = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)
```

While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
one can add multiple variables in one go.

```r
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 = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)
```

The above call using `call_derivation()` is equivalent to the call using `derive_vars_dt()`
to derive variables `ASTDT` and `AENDT` separately at the beginning.

---

### Description

Queries Dataset

### Usage

queries
Format

An object of class tbl_df (inherits from tbl, data.frame) with 15 rows and 8 columns.

Source

An example of standard query dataset to be used in deriving variables in ADAE and ADCM

See Also

Other datasets: admiral_adlb, admiral_adsl, ex_single, example_qs, queries_mh

Queries MH Dataset

Description

Queries MH Dataset

Usage

queries_mh

Format

An object of class tbl_df (inherits from tbl, data.frame) with 14 rows and 8 columns.

Source

An example of standard query MH dataset to be used in deriving variables in ADMH

See Also

Other datasets: admiral_adlb, admiral_adsl, ex_single, example_qs, queries
query

Create an query object

**Description**

A query object defines a query, e.g., a Standard MedDRA Query (SMQ), a Standardized Drug Grouping (SDG), or a customized query (CQ). It is used as input to `create_query_data()`.

**Usage**

```
query(prefix, name = auto, id = NULL, add_scope_num = FALSE, definition = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>The value is used to populate PREFIX in the output dataset of <code>create_query_data()</code>, e.g., &quot;SMQ03&quot;</td>
</tr>
</tbody>
</table>
| name         | The value is used to populate GRPNAME in the output dataset of `create_query_data()`. If the auto keyword is specified, the variable is set to the name of the query in the SMQ/SDG database.  

*Permitted Values:* A character scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an `basket_select()` object.

| id           | The value is used to populate GRPID in the output dataset of `create_query_data()`. If the auto keyword is specified, the variable is set to the id of the query in the SMQ/SDG database.  

*Permitted Values:* A integer scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an `basket_select()` object.

| add_scope_num | Determines if SCOPEN in the output dataset of `create_query_data()` is populated.  

*If the parameter is set to TRUE, the definition must be an `basket_select()` object.*  

*Default:* FALSE  

*Permitted Values:* TRUE, FALSE

| definition    | Definition of terms belonging to the query  

There are three different ways to define the terms:

- An `basket_select()` object is specified to select a query from the SMQ database.

- A data frame with columns SRCVAR and TERMINAME or TERMINID can be specified to define the terms of a customized query. The SRCVAR should be set to the name of the variable which should be used to select the terms, e.g., "AEDECOD" or "AELTCD". SRCVAR does not need to be constant within a query. For example a query can be based on AEDECOD and AELT.

  If SRCVAR refers to a character variable, TERMINAME should be set to the value of the variable. If it refers to a numeric variable, TERMINID should be set to the value of the variable. If only character variables or only numeric variables are used, TERMINID or TERMINAME respectively can be omitted.
• A list of data frames and \texttt{basket\_select()} objects can be specified to define a customized query based on custom terms and SMQs. The data frames must have the same structure as described for the previous item.

\textit{Permitted Values}: an \texttt{basket\_select()} object, a data frame, or a list of data frames and \texttt{basket\_select()} objects.

\textbf{Value}

An object of class \texttt{query}.

\textbf{See Also}

\texttt{create\_query\_data()}, \texttt{basket\_select()}, \texttt{Queries Dataset Documentation}

Source Objects: \texttt{basket\_select()}, \texttt{censor\_source()}, \texttt{date\_source()}, \texttt{death\_event}, \texttt{dthcaus\_source()}, \texttt{event\_joined()}, \texttt{event\_source()}, \texttt{event()}, \texttt{records\_source()}, \texttt{tte\_source()}

\textbf{Examples}

```r
# create a query for an SMQ
library(tibble)
library(dplyr, warn.conflicts = FALSE)

# create a query for a SMQ
query(
  prefix = "SMQ02",
  id = auto,
  definition = basket_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW",
    type = "smq"
  )
)

# create a query for an SDG
query(
  prefix = "SDG01",
  id = auto,
  definition = basket_select(
    name = "5-aminosalicylates for ulcerative colitis",
    scope = NA_character_,
    type = "sdg"
  )
)

# creating a query for a customized query
cqterms <- tribble(
  "~TERMNAME", ~TERMID,
  "APPLICATION SITE ERYTHEMA", 10003041L,
  "APPLICATION SITE PRURITUS", 10003053L
) %>%
```
mutate(SRCVAR = "AEDECOD")

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

# creating a customized query based on SMQs and additional terms
query(
  prefix = "CQ03",
  name = "Special issues of interest",
  definition = list(
    cqterms,
    basket_select(
      name = "Pregnancy and neonatal topics (SMQ)",
      scope = "NARROW",
      type = "smq"
    ),
    basket_select(
      id = 8050L,
      scope = "BROAD",
      type = "smq"
    )
  )
)

---

records_source Create a records_source Object

Description

The records_source object is used to find extreme records of interest.

Usage

records_source(dataset_name, filter = NULL, new_vars)

Arguments

dataset_name The name of the source dataset
The name refers to the dataset provided by the source_datasets argument of
derive_param_extreme_record().

filter An unquoted condition for selecting the observations from dataset.

new_vars Variables to add
The specified variables from the source datasets are added to the output dataset.
Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>).
For example new_vars = exprs(var1, var2) adds variables var1 and var2
from to the input dataset.
And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from the source dataset and adds them to the input dataset renaming old_var2 to new_var2. Expressions can be used to create new variables (see for example new_vars argument in derive_vars_merged()).

Permitted Values: list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

Value

An object of class records_source

See Also

derive_param_extreme_record()

Source Objects: basket_select(), censor_source(), date_source(), death_event, dthcaus_source(), event_joined(), event_source(), event(), query(), tte_source()

restrict_derivation

Execute a Derivation on a Subset of the Input Dataset

Description

Execute a derivation on a subset of the input dataset.

Usage

restrict_derivation(dataset, derivation, args = NULL, filter)

Arguments

dataset Input dataset
derivation Derivation

A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. The function must provide the dataset argument and all arguments specified in the params() objects passed to the arg argument.

Please note that it is not possible to specify {dplyr} functions like mutate() or summarize().

args Arguments of the derivation

A params() object is expected.

filter Filter condition

See Also

params() slice_derivation()

Higher Order Functions: call_derivation(), derivation_slice(), slice_derivation()
Examples

```r
library(tibble)

adlb <- tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~ABLFL,
  "1",    -1, 113, NA_character_,
  "2",    0, 113, "Y",
  "3",    3, 117, NA_character_,
  "3",    0, 111, "Y",
  "3",    1, 101, NA_character_,
  "3",    2, 123, NA_character_,
)

# Derive BASE for post-baseline records only (derive_var_base() can not be used in this case
# as it requires the baseline observation to be in the input dataset)
restrict_derivation(
adlb,
derivation = derive_vars_merged,
args = params(
  by_vars = exprs(USUBJID),
dataset_add = adlb,
  filter_add = ABLFL == "Y",
  new_vars = exprs(BASE = AVAL)
),
filter = AVISITN > 0
)

# Derive BASE for baseline and post-baseline records only
restrict_derivation(
adlb,
derivation = derive_var_base,
args = params(
  by_vars = exprs(USUBJID)
),
filter = AVISITN >= 0
)

# Derive CHG for post-baseline records only
restrict_derivation(
derivation = derive_var_chg,
filter = AVISITN > 0
)
```

set_admiral_options  

Set the Value of Admiral Options

Description

Set the Values of Admiral Options That Can Be Modified for Advanced Users.
Usage

set_admiral_options(subject_keys)

Arguments

subject_keys  Variables to uniquely identify a subject, defaults to `exprs(STUDYID, USUBJID)`. This option is used as default value for the subject_keys argument in all admiral functions.

Details

Modify an admiral option, e.g subject_keys, such that it automatically affects downstream function inputs where `get_admiral_option()` is called such as `derive_param_exist_flag()`.

Value

No return value, called for side effects.

See Also

`get_admiral_option()`, `derive_param_exist_flag()`, `derive_param_tte()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_vars_period()`, `create_period_dataset()`

Other admiral_options: `get_admiral_option()`

Examples

```r
library(lubridate)
library(dplyr, warn.conflicts = FALSE)
library(tibble)
set_admiral_options(subject_keys = exprs(STUDYID, USUBJID2))

# Derive a new parameter for measurable disease at baseline
ads1 <- tribble(~USUBJID2, ~STUDYID, ~VISIT, ~TUSTRESC, 
                 "1", "XX1234", "SCREENING", "TARGET", 
                 "1", "XX1234", "WEEK 1", "TARGET", 
                 "1", "XX1234", "WEEK 5", "TARGET", 
                 "1", "XX1234", "WEEK 9", "NON-TARGET", 
                 "2", "XX1234", "SCREENING", "NON-TARGET", 
                 "2", "XX1234", "SCREENING", "NON-TARGET"
) %>%
  mutate(STUDYID = "XX1234")

tu <- tribble(~USUBJID2, ~VISIT, ~TUSTRESC, 
              "1", "SCREENING", "TARGET", 
              "1", "WEEK 1", "TARGET", 
              "1", "WEEK 5", "TARGET", 
              "1", "WEEK 9", "NON-TARGET", 
              "2", "SCREENING", "NON-TARGET", 
              "2", "SCREENING", "NON-TARGET"
) %>%
  mutate(STUDYID = "XX1234")
```
TUTESTCD = "TUMIDENT"
)

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

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(replace_values_by_names(by_vars))),
    cnd_type = "error"
)

Arguments

dataset A data frame
by_vars A list of expressions created using exprs() 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
See Also

Utilities used within Derivation functions: `call_user_fun()`, `extract_unit()`, `get_not_mapped()`

Examples

data(admiral_adsl)

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

signal_duplicate_records(adsl, exprs(USUBJID), cnd_type = "message")

---

`slice_derivation`  
*Execute a Derivation with Different Arguments for Subsets of the Input Dataset*

Description

The input dataset is split into slices (subsets) and for each slice the derivation is called separately. Some or all arguments of the derivation may vary depending on the slice.

Usage

```
slice_derivation(dataset, derivation, args = NULL, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dataset</code></td>
<td>Input dataset</td>
</tr>
</tbody>
</table>
| `derivation` | Derivation  
A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. The function must provide the dataset argument and all arguments specified in the `params()` objects passed to the `arg` argument.  
Please note that it is not possible to specify `{dplyr}` functions like `mutate()` or `summarize()`.
| `args` | Arguments of the derivation  
A `params()` object is expected. |
| `...` | A `derivation_slice()` object is expected  
Each slice defines a subset of the input dataset and some of the parameters for the derivation. The derivation is called on the subset with the parameters specified by the `args` parameter and the `args` field of the `derivation_slice()` object. If a parameter is specified for both, the value in `derivation_slice()` overwrites the one in `args`. |
**slice_derivation**

### Details

For each slice, the derivation is called on the subset defined by the filter field of the `derivation_slice()` object and with the parameters specified by the `args` parameter and the `args` field of the `derivation_slice()` object. If a parameter is specified for both, the value in `derivation_slice()` overwrites the one in `args`.

- Observations that match with more than one slice are only considered for the first matching slice.
- Observations with no match to any of the slices are included in the output dataset but the derivation is not called for them.

### Value

The input dataset with the variables derived by the derivation added

### See Also

- `params()` `restrict_derivation()`
- Higher Order Functions: `call_derivation()`, `derivation_slice()`, `restrict_derivation()`

### Examples

```r
library(tibble)
library(stringr)
advs <- tribble(
  ~USUBJID, ~VSDTC, ~VSTPT,
  "1", "2020-04-16", NA_character_,
  "1", "2020-04-16", "BEFORE TREATMENT"
)

# For the second slice filter is set to TRUE. Thus derive_vars_dtm is called # with time_imputation = "last" for all observations which do not match for the # first slice.
slice_derivation(
advs,
derivation = derive_vars_dtm,
args = params(
dtc = VSDTC,
new_vars_prefix = "A"
),
derivation_slice(
  filter = str_detect(VSTPT, "PRE\|BEFORE"),
  args = params(time_imputation = "first")
),
derivation_slice(
  filter = TRUE,
  args = params(time_imputation = "last")
)
)
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 or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object.

censor  Censoring value
  CDISC strongly recommends using 0 for events and positive integers for censoring.

set_values_to  A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.

Value

An object of class tte_source

See Also

derive_param_tte(), censor_source(), event_source()

Source Objects: basket_select(), censor_source(), date_source(), death_event, dthcaus_source(), event_joined(), event_source(), event(), query(), records_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"),
    package = "admiral",
    overwrite = FALSE,
    open = interactive()
)

Arguments

adam_name  An ADaM dataset name. You can use any of the available dataset name ADAE, ADCM, ADEG, ADEX, ADLB, ADLBHY, ADMH, ADPC, ADPP, ADPPK, ADSL, ADVS, and the dataset name is case-insensitive. The default dataset name is ADSL.
save_path  Path to save the script.
package  The R package in which to look for templates. By default "admiral".
overwrite  Whether to overwrite an existing file named save_path.
open  Whether to open the script right away.

Details

Running without any arguments such as use_ad_template() auto-generates adsl.R in the current path. Use list_all_templates() to discover which templates are available.

Value

No return values, called for side effects

See Also

Utilities used for examples and template scripts: list_alltemplates()

Examples

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

Validate an object is indeed a basket_select object

Description
Validate an object is indeed a basket_select object

Usage
validate_basket_select(obj)

Arguments
obj An object to be validated.

Value
The original object.

See Also
basket_select()
Other Advanced Functions: assert_db_requirements(), assert_terms(), assert_valid_queries(),
extend_source_datasets(), filter_date_sources(), format.basket_select(), list_tte_source_objects(),
params(), validate_query()

validate_query

Validate an object is indeed a query object

Description
Validate an object is indeed a query object

Usage
validate_query(obj)

Arguments
obj An object to be validated.

Value
The original object.
yn_to_numeric

See Also

query()

Other Advanced Functions: assert_db_requirements(), assert_terms(), assert_valid_queries(), extend_source_datasets(), filter_date_sources(), format.basket_select(), list_tte_source_objects(), params(), validate_basket_select()

---

yn_to_numeric  Map "Y" and "N" to Numeric Values

Description

Map "Y" and "N" to numeric values.

Usage

yn_to_numeric(arg)

Arguments

arg  Character vector

Value

1 if arg equals "Y", 0 if arg equals "N", NA_real_ otherwise

See Also

Utilities for Formatting Observations: convert_blanks_to_na(), convert_na_to_blanks()

Examples

yn_to_numeric(c("Y", "N", NA_character_))
Pipe operator

Description
See magrittr::%>% for more details.

Usage
lhs %>% rhs

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lhs</td>
<td>A value or the magrittr placeholder.</td>
</tr>
<tr>
<td>rhs</td>
<td>A function call using the magrittr semantics.</td>
</tr>
</tbody>
</table>
Index

* admiral_options
  get_admiral_option, 272
  set_admiral_options, 299
* com_bds_findings
  compute_bmi, 20
  compute_bsa, 21
  compute_egfr, 25
  compute_framingham, 28
  compute_map, 30
  compute_qtc, 32
  compute_qual_imputation, 33
  compute_qual_imputation_dec, 34
  compute_rr, 35
  compute_scale, 36
* com_date_time
  compute_age_years, 19
  compute_dtf, 22
  compute_duration, 23
  compute_tmf, 37
  convert_date_to_dtm, 41
  convert_dtc_to_dt, 44
  convert_dtc_to_dtm, 46
  impute_dtc_dt, 280
  impute_dtc_dtm, 283
* create_aux
  consolidate_metadata, 38
  create_period_dataset, 50
  create_query_data, 53
  create_single_dose_dataset, 56
* datasets
  admiral_adlb, 5
  admiral_adsl, 6
  ex_single, 254
  example_qs, 251
  queries, 293
  queries_mh, 294
* deprecated
  derive_param_extreme_event, 105
  derive_var_basetype, 185
  derive_var_confirmation_flag, 187
  derive_var_last_dose_amt, 213
  derive_var_last_dose_date, 215
  derive_var_last_dose_grp, 216
  derive_var_merged_cat, 218
  derive_var_merged_character, 220
  derive_vars_last_dose, 158
* der_adsl
  derive_var_age_years, 175
  derive_var_dthcaus, 190
  derive_var_extreme_dt, 193
  derive_var_extreme_dtm, 198
  derive_vars_aage, 133
  derive_vars_period, 169
* der_bds_findings
  derive_basetype_records, 65
  derive_var_analysis_ratio, 176
  derive_var_anrind, 178
  derive_var_atoxgr, 179
  derive_var_atoxgr_dir, 181
  derive_var_base, 183
  derive_var_chg, 186
  derive_var_ontrtfl, 230
  derive_var_pchg, 233
  derive_var_shift, 238
* der_date_time
  derive_var_trtdurd, 239
  derive_vars_dt, 136
  derive_vars_dtm, 140
  derive_vars_dtm_to_dt, 145
  derive_vars_dtm_to_tm, 146
  derive_vars_duration, 147
  derive_vars_dy, 151
* der_gen
  derive_var_extreme_flag, 202
  derive_var_joined_exist_flag, 207
  derive_var_merged_exist_flag, 222
  derive_var_merged_summary, 225
  derive_var_obs_number, 228
derive_var_relative_flag, 234
derive_var_trtdurd, 239
derive_vars_dt, 136
derive_vars_dtm, 140
derive_vars_dtm_to_dt, 145
derive_vars_dtm_to_tm, 146
derive_vars_duration, 147
derive_vars_dy, 151
derive_vars_joined, 152
derive_vars_merged, 160
derive_vars_merged_lookup, 166
derive_vars_transposed, 173
get_summary_records, 276
* der_occds
derive_var_trtemfl, 241
derive_vars_atc, 134
derive_vars_query, 171
get_terms_from_db, 279
* der_prm_bds_findings
default_qtc_paramcd, 64
derive_expected_records, 67
derive_extreme_event, 69
derive_extreme_records, 76
derive_locf_records, 81
derive_param_bmi, 84
derive_param_bsa, 87
derive_param_computed, 90
derive_param_doseint, 96
derive_param_exist_flag, 98
derive_param_exposure, 102
derive_param_extreme_record, 107
derive_param_framingham, 110
derive_param_map, 114
derive_param_qtc, 117
derive_param_rr, 119
derive_param_wbc_abs, 127
derive_summary_records, 129
* der_prm_tte
derive_param_tte, 121
* high_order_function
call_derivation, 14
derivation_slice, 65
restrict_derivation, 298
slice_derivation, 302
* metadata
atoxgr_criteria_ctcv4, 10
atoxgr_criteria_ctcv5, 11
atoxgr_criteria_daids, 12
dose_freq_lookup, 244
* other_advanced
assert_db_requirements, 7
assert_terms, 8
assert_valid_queries, 9
extend_source_datasets, 252
filter_date_sources, 255
format.basket_select, 271
list_tte_source_objects, 288
params, 291
validate_basket_select, 306
validate_query, 306
* reexport
%%, 308
desc, 244
exprs, 251
* source_specifications
basket_select, 13
censor_source, 17
date_source, 61
death_event, 63
dthcaus_source, 245
event, 246
event_joined, 248
event_source, 250
query, 295
records_source, 297
tte_source, 304
* utils_ds_chk
extract_duplicated_records, 253
get_duplicates_dataset, 273
get_many_to_one_dataset, 274
get_one_to_many_dataset, 275
* utils_examples
list_all_templates, 287
use_ad_template, 305
* utils_fil
count_vals, 49
filter_exist, 257
filter_extreme, 259
filter_joined, 261
filter_not_exist, 266
filter_relative, 268
max_cond, 289
min_cond, 290
* utils_fmt
convert_blanks_to_na, 40
convert_na_to_blanks, 48
yn_to_numeric, 307
* utils_help
call_user_fun, 16
extract_unit, 254
get_not_mapped, 275
signal_duplicate_records, 301
* utils_quo
chr2vars, 18
negate_vars, 291
%>%, 308, 309

admiral_adlb, 5, 6, 251, 255, 294
admiral_adsl, 6, 6, 251, 255, 294
ae_event (death_event), 63
ae_gr1_event (death_event), 63
ae_gr2_event (death_event), 63
ae_gr35_event (death_event), 63
ae_gr3_event (death_event), 63
ae_gr4_event (death_event), 63
ae_gr5_event (death_event), 63
ae_ser_event (death_event), 63
ae_sev_event (death_event), 63
ae_wd_event (death_event), 63
assert_db_requirements, 7, 8, 9, 252, 256, 271, 288, 292, 306, 307
assert_terms, 7, 8, 9, 252, 256, 271, 288, 292, 306, 307
assert_valid_queries, 7, 8, 9, 252, 256, 271, 288, 292, 306, 307
assert_valid_queries(), 172
atoxgr_criteria_ctcv4, 10, 12, 13, 245
atoxgr_criteria_ctcv5, 11, 11, 13, 245
atoxgr_criteria_daidsl, 11, 12, 12, 245

basket_select, 13, 17, 62, 64, 246, 248–250, 296, 298, 305
basket_select(), 54, 271, 296, 306

call_derivation, 14, 65, 298, 303
call_derivation(), 291, 292
call_user_fun, 16, 254, 275, 302
censor_source, 13, 17, 62, 64, 246, 248–250, 296, 298, 305
censor_source(), 64, 124, 250, 305
chr2vars, 18, 291
compute_age_years, 19, 22, 25, 38, 43, 45, 48, 282, 286
compute_bmi, 20, 21, 27, 30–35, 37
compute_bmi(), 85
compute_bsa, 20, 21, 27, 30–35, 37
compute_bsa(), 89
compute_dtf, 19, 22, 25, 38, 43, 45, 48, 282, 286
compute_duration, 19, 22, 23, 38, 43, 45, 48, 282, 286
compute_duration(), 149
compute_egfr, 20, 21, 25, 30–35, 37
compute_framingham, 20, 21, 27, 28, 31–35, 37
compute_framingham(), 113
compute_map, 20, 21, 27, 30, 32–35, 37
compute_map(), 116
compute_qtc, 20, 21, 27, 30, 31, 32, 33–35, 37
compute_qtc(), 118
compute_qual_imputation, 20, 21, 27, 30–32, 33, 34, 35, 37
compute_qual_imputation_dec, 20, 21, 27, 30–33, 34, 35, 37
calculate_rri(), 121
calculate_scale, 20, 21, 27, 30–35, 36
calculate_tmf, 19, 22, 25, 37, 43, 45, 48, 282, 286
consolidate_metadata, 38, 51, 54, 58
convert_blanks_to_na, 40, 49, 307
convert_date_to_dtm, 19, 22, 25, 38, 41, 45, 48, 282, 286
convert_dtc_to_dtm, 19, 22, 25, 38, 43, 44, 48, 282, 286
convert_dtc_to_dtm(), 170
create_query_data, 39, 51, 54, 56
create_single_dose_dataset, 39, 51, 54, 56
create_single_dose_dataset(), 160, 245
date_source, 13, 17, 61, 64, 246, 248–250, 296, 298, 304
date_source(), 194, 199
dead_event, 13, 17, 62, 63, 246, 248–250, 296, 298, 304
default_qtc_paramcd, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derivation_slice, 14, 65, 298, 303
derive_basetype_records, 65, 177, 178, 180, 182, 184, 186, 232, 234, 239
derive_expected_records, 64, 67, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_extreme_event(), 248, 249
derive_extreme_records, 64, 68, 72, 76, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_locf_records, 64, 68, 72, 79, 81, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_bmi, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_bmi(), 20
derive_param_bsa, 64, 68, 72, 79, 83, 85, 87, 93, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_bsa(), 21
derive_param_computed, 64, 68, 72, 79, 83, 85, 89, 90, 98, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_doseint, 64, 68, 72, 79, 83, 85, 89, 93, 96, 101, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_exist_flag, 64, 68, 72, 79, 83, 85, 89, 93, 98, 103, 108, 113, 116, 118, 121, 129, 131
derive_param_exist_flag(), 272, 300
derive_param_exposure, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 102, 108, 113, 116, 118, 121, 129, 131
derive_param_extreme_event, 104, 160, 186, 190, 215, 216, 218, 220, 222
derive_param_extreme_record, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 107, 113, 116, 118, 121, 129, 131
derive_param_extreme_record(), 298
derive_param_framingham, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 109, 116, 118, 121, 129, 131
derive_param_framingham(), 30
derive_param_map, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 114, 118, 121, 129, 131
derive_param_map(), 31
derive_param_qtc, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 117, 121, 129, 131
derive_param_qtc(), 32, 64
derive_param_rr(), 35
derive_param_tte, 121
derive_param_tte(), 17, 63, 64, 250, 272, 300, 304
derive_param_wbc_abs, 64, 68, 72, 79, 83, 85, 89, 93, 98, 101, 103, 108, 113, 116, 118, 121, 127, 131
derive_summary_records(), 227
derive_var_age_years, 134, 170, 175, 191, 194, 199
derive_var_analysis_ratio, 66, 176, 178, 180, 182, 184, 186, 232, 234, 239
derive_var_anrind, 66, 177, 178, 180, 182, 184, 186, 232, 234, 239
derive_var_atoxgr, 66, 177, 178, 179, 182, 184, 186, 232, 234, 239
derive_var_atoxgr_dir, 66, 177, 178, 180, 181, 184, 186, 232, 234, 239
derive_var_base, 66, 177, 178, 180, 182, 183, 186, 232, 234, 239
derive_var_basetype, 107, 160, 185, 190, 215, 216, 218, 220, 222
derive_var_chg, 66, 177, 178, 180, 182, 184, 186, 232, 234, 239
derive_var_chg(), 234
derive_var_confirmation_flag, 107, 160, 186, 187, 215, 216, 220, 222
derive_var_dthcaus, 134, 170, 175, 190, 194, 199
derive_var_dthcaus(), 246, 272, 300
derive_var_extreme_dt, 134, 170, 175, 191, 193, 199
derive_var_last_dose_amt
derive_var_joined_exist_flag
INDEX
derive_var_trtemfl
derive_vars_dt
derive_vars_aage
derive_var_trtdurd
derive_var_shift
derive_var_relative_flag
derive_var_ontrtfl
derive_var_obs_number
derive_var_merged_summary
derive_var_merged_character
derive_var_merged_flag
derive_var_merged_lookup
deriveVars_dtm_to_tm,
deriveVars_dtm_to_dt,
derive_vars_duration,
derive_vars_duration(),
derive_vars_dy,
derive_vars_joined,
dthcaus_source
derive_vars_period,
dthcaus_source()
dose_freq_lookup
derivative
event
ex_single
exprs
extend_source_datasets
11–13
148