# Package 'admiral'

September 5, 2022

**Type** Package

**Title** ADaM in R Asset Library

**Version** 0.8.0

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

**Language** en-US

**License** Apache License (>= 2)

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

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.0

**Depends** R (>= 3.5)

**Imports** admiraldev, assertthat (>= 0.2.1), 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), tidyr (>= 1.0.2), tidyselect (>= 1.0.0)

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

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Thomas Neitmann [aut, cre], Stefan Bundfuss [aut], Ben Straub [aut], Samia Kabi [aut],

Gordon Miller [aut],
Teckla Akinyi [aut],
Andrew Smith [aut],
Konstantina Koukourikou [aut],
Ross Farrugia [aut],
Eric Simms [aut],
Annie Yang [aut],
Robin Koeger [aut],
Sophie Shapcott [aut],
Ojesh Upadhyay [aut],
Jack McGavigan [aut],
Kamila Duniec [aut],
Gayatri G [aut],
Alana Harris [aut],
Mahdi About [aut],
Pooja Kumari [aut],
Claudia Carlucci [aut],
Daniil Stefonishin [aut],
Michael Thorpe [ctb],
Pavan Kumar [ctb],
Hamza Rahal [ctb],
Alice Ehmann [ctb],
Tom Ratford [ctb],
Vignesh Thanikachalam [ctb],
Ondrej Slama [ctb],
Shimeng Huang [ctb],
James Kim [ctb],
Shan Lee [ctb],
Bill Denney [ctb],
Syed Mubasheer [ctb],
Wenyi Liu [ctb],
Dinakar Kulkarni [ctb],
Tamara Senior [ctb],
Jordanna Morrish [ctb],
Anthony Howard [ctb],
Barbara O'Reilly [ctb],
John Kirkpatrick [ctb],
James Black [ctb],
Leena Khatri [ctb],
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline LLC [cph, fnd]

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

**Repository** CRAN

**Date/Publication** 2022-09-05 10:20:13 UTC

# R topics documented:

---

`admiral_adsl` *Subject Level Analysis Dataset*

---

## Description

An example subject level analysis dataset

## Usage

```
admiral_adsl
```

## Format

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

## Source

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

## See Also

Other datasets: `atoxgr_criteria_ctcv4`, `ex_single`, `queries_mh`, `queries`

---

`assert_db_requirements` *Check required parameters for SMQ/SDG*

---

## Description

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

## Usage

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

### Arguments

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

### Value

An error is issued if version or fun is null.

### Author(s)

Stefan Bundfuss

### See Also

Source Specifications: assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

---

## assert_terms *Asserts Requirements for Terms for Queries*

---

### Description

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

### Usage

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

## Arguments

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

## Value

An error is issued if

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

## Author(s)

Stefan Bundfuss

## See Also

create_query_data(), query()

Source Specifications: assert_db_requirements(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## 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

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

## Value

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

## Author(s)

Shimeng Huang, Ondrej Slama

## See Also

Source Specifications: assert_db_requirements(), assert_terms(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

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

---

# atoxgr_criteria_ctcv4 *Metadata Holding Grading Criteria for NCI-CTCAEv4*

---

## Description

Metadata Holding Grading Criteria for NCI-CTCAEv4

## Usage

```
atoxgr_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")`. 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. From these variables only 'TERM' is used in the admiral code, the rest are for information and tracability only.

### Author(s)

Gordon Miller

### See Also

Other datasets: admiral_adsl, ex_single, queries_mh, queries

---

## call_derivation *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 |
| 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.

### Author(s)

Thomas Neitmann, Stefan Bundfuss, Tracey Wang

### See Also

params()

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

## Examples

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

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

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

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

---

`call_user_fun` *Calls a Function Provided by the User*

---

## Description

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

## Usage

```
call_user_fun(call)
```

## Arguments

| | |
|---|---|
| call | Call to be executed |

## Value

The return value of the function call

## Author(s)

Stefan Bundfuss

## See Also

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

## Examples

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

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

---

# censor_source *Create a* censor_source *Object*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset_name | The name of the source dataset<br>The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). |
| filter | An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. |
| date | A variable providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol is expected.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. |
| censor | Censoring value<br>CDISC strongly recommends using 0 for events and positive integers for censoring. |
| set_values_to | A named list returned by vars() defining the variables to be set for the event or censoring, e.g. vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, or NA. |

## Value

An object of class censor_source, inheriting from class tte_source

## Author(s)

Stefan Bundfuss

## See Also

derive_param_tte(), event_source()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

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

compute_bmi *Compute Body Mass Index (BMI)*

## Description

Computes BMI from height and weight

## Usage

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

## Arguments

| | |
|---|---|
| height | HEIGHT value<br>It is expected that HEIGHT is in cm.<br>Permitted Values: numeric vector |
| weight | WEIGHT value<br>It is expected that WEIGHT is in kg.<br>Permitted Values: numeric vector |

## Details

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

## Value

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

## Author(s)

Pavan Kumar

## See Also

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

## Examples

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

`compute_bsa` *Compute Body Surface Area (BSA)*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `height` | HEIGHT value<br>It is expected that HEIGHT is in cm.<br>Permitted Values: numeric vector |
| `weight` | WEIGHT value<br>It is expected that WEIGHT is in kg.<br>Permitted Values: numeric vector |
| `method` | Derivation method to use:<br>Mosteller: sqrt(height * weight / 3600)<br>DuBois-DuBois: 0.20247 * (height/100) ^ 0.725 * weight ^ 0.425<br>Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378<br>Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456<br>Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight)))<br>Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444<br>Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425<br>Permitted Values: character value |

## Details

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

## Value

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

## Author(s)

Eric Simms

## See Also

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

## 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'`).<br>A character date is expected in a format like `yyyy-mm-ddThh:mm:ss` (partial or complete). |
| `dt` | The Date vector to compare.<br>A date object is expected. |

## Details

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

## Value

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

## Author(s)

Samia Kabi

## See Also

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

## Examples

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

---

# compute_duration *Compute Duration*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| start_date | The start date<br>A date or date-time object is expected.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.<br>Refer to convert_dtc_to_dt() to obtain a vector of imputed dates. |
| end_date | The end date<br>A date or date-time object is expected.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.<br>Refer to convert_dtc_to_dt() to obtain a vector of imputed dates. |
| in_unit | Input unit<br>See floor_in and add_one parameter for details.<br>Default: 'days'<br>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds' |

| | |
|---|---|
| out_unit | Output unit<br>The duration is derived in the specified unit<br>Default: 'days'<br>Permitted Values: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds' |
| floor_in | Round down input dates?<br>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.<br>Default: 'TRUE"<br>Permitted Values: TRUE, FALSE |
| add_one | Add one input unit?<br>If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.<br>Default: TRUE<br>Permitted Values: TRUE, FALSE |
| trunc_out | Return integer part<br>The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.<br>Default: FALSE<br>Permitted Values: TRUE, FALSE |

## Details

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

## Value

The duration between the two date in the specified unit

## Author(s)

Stefan Bundfuss

## See Also

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

## Examples

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

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

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

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

---

`compute_framingham` *Compute Framingham Heart Study Cardiovascular Disease 10-Year Risk Score*

---

## 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<br>A numeric vector is expected. |
| `chol` | Total serum cholesterol (mg/dL)<br>A numeric vector is expected. |
| `cholhdl` | HDL serum cholesterol (mg/dL)<br>A numeric vector is expected. |

| | |
|---|---|
| age | Age (years)<br>A numeric vector is expected. |
| sex | Gender<br>A character vector is expected. Expected Values: 'M' 'F' |
| smokefl | Smoking Status<br>A character vector is expected. Expected Values: 'Y' 'N' |
| diabetfl | Diabetic Status<br>A character vector is expected. Expected Values: 'Y' 'N' |
| trthypfl | Treated for hypertension status<br>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 D'Agostino, 2008 is: # nolint

**For Women:**

| Factor | Amount |
|---|---|
| Age | 2.32888 |
| Total Chol | 1.20904 |
| HDL Chol | -0.70833 |
| Sys BP | 2.76157 |
| Sys BP + Hypertension Meds | 2.82263 |
| Smoker | 0.52873 |
| Non-Smoker | 0 |
| Diabetic | 0.69154 |
| Not Diabetic | 0 |
| Average Risk | 26.1931 |
| Risk Period | 0.95012 |

**For Men:**

| Factor | Amount |
|---|---|
| Age | 3.06117 |
| Total Chol | 1.12370 |
| HDL Chol | -0.93263 |
| Sys BP | 1.93303 |
| Sys BP + Hypertension Meds | 2.99881 |
| Smoker | .65451 |
| Non-Smoker | 0 |
| Diabetic | 0.57367 |
| Not Diabetic | 0 |
| Average Risk | 23.9802 |
| Risk Period | 0.88936 |

**The equation for calculating risk:**

$$RiskFactors = (log(Age) * AgeFactor) + (log(TotalChol) * TotalCholFactor) + (log(CholHDL) * CholHDLFactor)$$

$$Risk = 100 * (1 - RiskPeriodFactor^{e}xp(RiskFactors))$$

## Value

A numeric vector of Framingham values

## Author(s)

Alice Ehmann

## See Also

`derive_param_framingham()`

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

## 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)
```

## Arguments

| | |
|---|---|
| diabp | Diastolic blood pressure<br>A numeric vector is expected. |
| sysbp | Systolic blood pressure<br>A numeric vector is expected. |
| hr | Heart rate<br>A numeric vector or NULL is expected. |

## Details

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

if it is based on diastolic and systolic blood pressure and

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

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

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

## Value

A numeric vector of MAP values

## Author(s)

Stefan Bundfuss

## See Also

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

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

## Details

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

*Bazett*:

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

*Fridericia*:

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

*Sagie*:

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

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

## Value

QT interval in msec

## Author(s)

Stefan Bundfuss

## See Also

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

## 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 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

## Author(s)

Alice Ehmann Ojesh Upadhyay

## See Also

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

## 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^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

```
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

## Author(s)

Alice Ehmann Ojesh Upadhyay

## See Also

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

## Examples

```
compute_qual_imputation_dec("<40.1")
```

compute_rr *Compute RR Interval From Heart Rate*

## Description

Computes RR interval from heart rate.

## Usage

```
compute_rr(hr)
```

## Arguments

| | |
|---|---|
| hr | Heart rate<br>A numeric vector is expected. It is expected that heart rate is measured in beats/min. |

## Details

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

## Value

RR interval in msec:

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

## Author(s)

Stefan Bundfuss

## 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()

## Examples

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

compute_tmf *Derive the Time Imputation Flag*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dtc` | The date character vector (`'--DTC'`).<br>A character date is expected in a format like `yyyy-mm-ddThh:mm:ss` (partial or complete). |
| `dtm` | The Date vector to compare (`'--DTM'`).<br>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'`).<br>A logical value<br>Default: `FALSE` |

## Details

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

## Value

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

## Author(s)

Samia Kabi, Stefan Bundfuss

## See Also

Date/Time Computation Functions that returns a vector: `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 = "2019-07-18T15:25", dtm = as.POSIXct("2019-07-18T15:25:00"))
compute_tmf(dtc = "2019-07-18T15", dtm = as.POSIXct("2019-07-18T15:25:00"))
compute_tmf(dtc = "2019-07-18", dtm = as.POSIXct("2019-07-18"))
```

---

# convert_blanks_to_na *Convert Blank Strings Into NAs*

---

## Description

Turn SAS blank strings into proper R NAs.

## Usage

```
convert_blanks_to_na(x)

## Default S3 method:
convert_blanks_to_na(x)

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

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

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

## Arguments

| | |
|---|---|
| x | Any R object |

## Details

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

## Value

An object of the same class as the input

## Author(s)

Thomas Neitmann

## See Also

Utilities for Formatting Observations: format_eoxxstt_default(), format_reason_default(), yn_to_numeric()

## Examples

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

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

---

# convert_date_to_dtm *Convert a Date into a Datetime Object*

---

## Description

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

## Usage

```
convert_date_to_dtm(
  dt,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

## Arguments

| | |
|---|---|
| dt | The date to convert.<br>A date or character date is expected in a format like yyyy-mm-ddThh:mm:ss. |
| highest_imputation | Highest imputation level<br>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.<br>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

```
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 datetime object

## Author(s)

Samia Kabi

## See Also

Date/Time Computation Functions that returns a vector: 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_dtc_to_dt` *Convert a Date Character Vector into a Date Object*

## Description

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

## Usage

```
convert_dtc_to_dt(
  dtc,
  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.

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

```
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

## Author(s)

Samia Kabi

## See Also

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

## Examples

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

---

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

---

## Description

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

## Usage

```
convert_dtc_to_dtm(
  dtc,
  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

```
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<br>For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").<br>Permitted Values: TRUE, FALSE<br>*Default*: FALSE |

## Details

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

## Value

A datetime object

## Author(s)

Samia Kabi, Stefan Bundfuss

## See Also

Date/Time Computation Functions that returns a vector: 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")
```

---

# 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 |

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Filtering Observations: filter_confirmation(), filter_extreme(), filter_relative(), max_cond(), min_cond()

## Examples

```
library(tibble)
library(dplyr)
library(admiral)
data <- tibble::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"))
```

---

# create_query_data *Creates a queries dataset as input dataset to the* dataset_queries *argument in* derive_vars_query()

---

## Description

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

## Usage

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

## Arguments

`queries` List of queries

A list of `query()` objects is expected.

`meddra_version` MedDRA version

The MedDRA version used for coding the terms in the AE dataset should be specified. If any of the queries is a SMQ or a customized query including a SMQ, the parameter needs to be specified.

*Permitted Values*: A character string (the expected format is company-specific)

`whodd_version` WHO Drug Dictionary version

The version of the WHO Drug Dictionary used for coding the terms in the CM dataset should be specified. If any of the queries is a SDG, the parameter needs to be specified.

*Permitted Values*: A character string (the expected format is company-specific)

`get_smq_fun` Function which returns the terms of an SMQ

For each query specified for the `queries` parameter which refers to an SMQ (i.e., those where the `definition` field is set to a `smq_select()` object or a list which contains at least one `smq_select()` object) the specified function is called to retrieve the terms defining the query. This function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level.

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

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

The function must provide the following parameters

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

`get_sdg_fun` Function which returns the terms of an SDG

For each query specified for the `queries` parameter which refers to an SDG the specified function is called to retrieve the terms defining the query. This

function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level.

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

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

The function must provide the following parameters

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

## Details

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

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

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

- VAR_PREFIX: Prefix of the variables to be created by derive_vars_query() as specified by the prefix element.
- QUERY_NAME: Name of the query as specified by the name element.

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

derive_vars_query(), query(), smq_select(), sdg_select(), Queries Dataset Documentation

OCCDS Functions: create_single_dose_dataset(), derive_vars_atc(), derive_vars_query(), get_terms_from_db()

## Examples

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

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

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

```
create_query_data(queries = list(cq))

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

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

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

# create a query dataset for SDGs
sdg <- query(
  prefix = "SDG01",
  id = auto,
  definition = sdg_select(
    name = "5-aminosalicylates for ulcerative colitis"
  )
)

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

# creating a query dataset for a customized query including SMQs
# The get_smq_terms function from admiral.test is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(
    query(
      prefix = "CQ03",
```

```
      name = "Special issues of interest",
      definition = list(
        smq_select(
          name = "Pregnancy and neonatal topics (SMQ)",
          scope = "NARROW"
        ),
        cqterms
      )
    )
  ),
  get_smq_fun = admiral.test:::get_smq_terms,
  meddra_version = "20.1"
)
```

---

`create_single_dose_dataset` *Create dataset of single doses*

---

## Description

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

## Usage

```
create_single_dose_dataset(
  dataset,
  dose_freq = EXDOSFRQ,
  start_date = ASTDT,
  start_datetime = ASTDTM,
  end_date = AENDT,
  end_datetime = AENDTM,
  lookup_table = dose_freq_lookup,
  lookup_column = CDISC_VALUE,
  keep_source_vars = vars(USUBJID, EXDOSFRQ, ASTDT, ASTDTM, AENDT, AENDTM)
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The columns specified by `dose_freq`, `start_date` and the `end_date` parameters are expected. |
| `dose_freq` | The dose frequency<br>The aggregate dosing frequency used for multiple doses in a row.<br>Default: `EXDOSFRQ`<br>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.
Default: ASTDT

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.
Default: ASTDTM

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.
Default: AENDT

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.
Default: AENDTM

lookup_table
: The dose frequency value lookup table
The table used to look up dose_freq values and determine the appropriate multiplier to be used for row generation. If a lookup table other than the default is used, it must have columns DOSE_WINDOW, DOSE_COUNT, and CONVERSION_FACTOR. The default table dose_freq_lookup is described in detail here.
Default: dose_freq_lookup
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.
Default: CDISC_VALUE (column of dose_freq_lookup)

keep_source_vars
: List of variables to be retained from source dataset
Default: vars(USUBJID, EXDOSFRQ, ASTDT, ASTDTM, AENDT, AENDTM)

## 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.

## Value

The input dataset with a single dose per row.

## Author(s)

Michael Thorpe, Andrew Smith

## See Also

OCCDS Functions: create_query_data(), derive_vars_atc(), derive_vars_query(), get_terms_from_db()

## Examples

```
# Example with default lookup

library(lubridate)
library(stringr)

data <- tibble::tribble(
  ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "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 <- tibble::tribble(
  ~Value, ~DOSE_COUNT, ~DOSE_WINDOW, ~CONVERSION_FACTOR,
  "Q30MIN", (1 / 30), "MINUTE", 1,
  "Q90MIN", (1 / 90), "MINUTE", 1
)

data <- tibble::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
)
```

date_source *Create a* date_source *object*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset_name | The name of the dataset, i.e. a string, used to search for the date. |
| filter | An unquoted condition for filtering dataset. |
| date | A variable providing a date. A date or a datetime can be specified. An unquoted symbol is expected. |
| date_imputation | *Deprecated*, please use derive_vars_dtm() to convert DTC variables to datetime variables in the dataset. |
| time_imputation | *Deprecated*, please use derive_vars_dtm() to convert DTC variables to datetime variables in the dataset. |
| preserve | *Deprecated*, please use derive_vars_dtm() to convert DTC variables to datetime variables in the dataset. |
| traceability_vars | A named list returned by vars() defining the traceability variables, e.g. vars(LALVDOM = "AE", LALVSEQ = AESEQ, LALVVAR = "AESTDTC"). The values must be a symbol, a character string, a numeric, or NA. |

## Value

An object of class date_source.

## Author(s)

Stefan Bundfuss

## See Also

derive_var_extreme_dtm(), derive_var_extreme_dt()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

---

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

---

## Description

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

## Usage

```
death_event

lastalive_censor

ae_event

ae_ser_event

ae_gr1_event

ae_gr2_event

ae_gr3_event

ae_gr4_event

ae_gr5_event

ae_gr35_event

ae_sev_event

ae_wd_event
```

## 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 derive_param_tte().

## See Also

derive_param_tte(), tte_source(), event_source(), censor_source()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

```
# 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")
}
```

---

# default_qtc_paramcd *Get Default Parameter Code for Corrected QT*

---

## Description

Get Default Parameter Code for Corrected QT

## Usage

```
default_qtc_paramcd(method)
```

## Arguments

| | |
|---|---|
| method | Method used to QT correction<br>Permitted Values: "Bazett", "Fridericia", "Sagie" |

## Value

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

## Author(s)

Thomas Neitmann

## See Also

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

## Examples

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

---

`derivation_slice` *Create a* `derivation_slice` *Object*

---

## Description

Create a `derivation_slice` object as input for `slice_derivation()`.

## Usage

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

## Arguments

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

## Value

An object of class `derivation_slice`

An object of class `slice`.

## Author(s)

Stefan Bundfuss

## See Also

`slice_derivation()`, `params()`

Higher Order Functions: `call_derivation()`, `print.derivation_slice()`, `restrict_derivation()`, `slice_derivation()`

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

## Description

**[Deprecated]**

This function is deprecated. Please use `derive_param-computed()` instead.

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected.<br>The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `parameters`. |
| `by_vars` | Grouping variables<br>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.<br>*Permitted Values:* list of variables |
| `parameters` | Required parameter codes<br>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.<br>*Permitted Values:* A character vector of `PARAMCD` values |
| `analysis_value` | Definition of the analysis value<br>An expression defining the analysis value (`AVAL`) of the new parameter is expected. The analysis values of the parameters specified by `parameters` can be accessed using `AVAL.<parameter code>`, e.g., `AVAL.SYSBP`.<br>*Permitted Values:* An unquoted expression |

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. For example `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter.

*Permitted Values:* List of variable-value pairs

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

*Permitted Values:* a condition

`constant_by_vars`

By variables for constant parameters

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

*Permitted Values:* list of variables

`constant_parameters`

Required constant parameter codes

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

*Permitted Values:* A character vector of `PARAMCD` values

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

Other deprecated: `derive_var_aendy()`

---

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

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

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

## Examples

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

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

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

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

---

`derive_param_bmi` *Adds a Parameter for BMI*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected.<br>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<br>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.<br>*Permitted Values:* list of variables |
| set_values_to | Variables to be set<br>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.<br>*Permitted Values:* List of variable-value pairs |
| weight_code | WEIGHT parameter code<br>The observations where PARAMCD equals the specified value are considered as the WEIGHT. It is expected that WEIGHT is measured in kg<br>Permitted Values: character value |
| height_code | HEIGHT parameter code<br>The observations where PARAMCD equals the specified value are considered as the HEIGHT. It is expected that HEIGHT is measured in cm<br>Permitted Values: character value |
| get_unit_expr | An expression providing the unit of the parameter<br>The result is used to check the units of the input parameters.<br>Permitted Values: A variable of the input dataset or a function call |
| filter | Filter condition<br>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.<br>*Permitted Values:* a condition |

## Details

The analysis value of the new parameter is derived as

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

## Value

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

## Author(s)

Pavan Kumar

## See Also

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

## Examples

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

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

---

# derive_param_bsa *Adds a Parameter for BSA (Body Surface Area) Using the Specified Method*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected.<br>The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by HEIGHT and WEIGHT. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values:* list of variables |
| method | Derivation method to use. Note that HEIGHT is expected in cm and WEIGHT is expected in kg:<br>Mosteller: sqrt(height * weight / 3600)<br>DuBois-DuBois: 0.20247 * (height/100) ^ 0.725 * weight ^ 0.425<br>Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378<br>Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456<br>Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight)))<br>Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444<br>Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425<br>Permitted Values: character value |
| set_values_to | Variables to be set<br>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.<br>*Permitted Values:* List of variable-value pairs |
| height_code | HEIGHT parameter code<br>The observations where PARAMCD equals the specified value are considered as the HEIGHT assessments. It is expected that HEIGHT is measured in cm.<br>Permitted Values: character value |

| | |
|---|---|
| weight_code | WEIGHT parameter code<br>The observations where PARAMCD equals the specified value are considered as the WEIGHT assessments. It is expected that WEIGHT is measured in kg.<br>Permitted Values: character value |
| get_unit_expr | An expression providing the unit of the parameter<br>The result is used to check the units of the input parameters.<br>Permitted Values: A variable of the input dataset or a function call |
| filter | Filter condition<br>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.<br>*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.

## Author(s)

Eric Simms

## See Also

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

## Examples

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

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

```
  get_unit_expr = extract_unit(PARAM)
)

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

---

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

---

## Description

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

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

## Usage

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

## Arguments

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

by_vars
: Grouping variables

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

  *Permitted Values:* A character vector of PARAMCD values

analysis_value
: Definition of the analysis value

  An expression defining the analysis value (AVAL) of the new parameter is expected. The analysis values of the parameters specified by parameters can be accessed using AVAL.<parameter code>, e.g., AVAL.SYSBP.

  *Permitted Values:* An unquoted expression

set_values_to
: Variables to be set

  The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.

  *Permitted Values:* List of variable-value pairs

filter
: Filter condition

  The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

  *Permitted Values:* a condition

constant_by_vars
: By variables for constant parameters

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

  *Permitted Values:* list of variables

constant_parameters
: Required constant parameter codes

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

  *Permitted Values:* A character vector of PARAMCD values

## Details

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

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

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

## Examples

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

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

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

```
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.3, "kg", "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.7, "kg", "WEEK 2"
)

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

---

`derive_param_doseint` *Adds a Parameter for Dose Intensity*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter, `PARAMCD`, and `AVAL` are expected.<br>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

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. For example `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter.

*Permitted Values:* List of variable-value pairs

`tadm_code` Total Doses Administered parameter code

The observations where `PARAMCD` equals the specified value are considered as the total dose administered. The `AVAL` associated with this `PARAMCD` will be the numerator of the dose intensity calculation.

Permitted Values: character value

`tpadm_code` Total Doses Planned parameter code

The observations where `PARAMCD` equals the specified value are considered as the total planned dose. The `AVAL` associated with this `PARAMCD` will be the denominator of the dose intensity calculation.

Permitted Values: character value

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

Default: `Inf`

Permitted Values: `Inf`, `100`

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

If `zero_doses = Inf`:

1. If the planned dose (`tpadm_code`) is 0 and administered dose (`tadm_code`) is 0, `NaN` is returned.
2. If the planned dose (`tpadm_code`) is 0 and the administered dose (`tadm_code`) is > 0, `Inf` is returned.

If `zero_doses = 100` :

1. If the planned dose (`tpadm_code`) is 0 and administered dose (`tadm_code`) is 0, 0 is returned.
2. If the planned dose (`tpadm_code`) is 0 and the administered dose (`tadm_code`) is > 0, 100 is returned.

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

*Permitted Values:* a condition

## Details

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

## Value

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

## Author(s)

Alice Ehmann

## See Also

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

## Examples

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

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

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

derive_param_doseint(
  adex,
  by_vars = vars(USUBJID, VISIT),
  set_values_to = vars(PARAMCD = "TDOSINT2"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
  zero_doses = "100"
)
```

derive_param_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 measureable disease at baseline.

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified for subject_keys and the PARAMCD variable are expected. |
| dataset_adsl | ADSL input dataset<br>The variables specified for subject_keys are expected. For each subject (as defined by subject_keys) from the specified dataset (dataset_adsl), the existence flag is calculated and added as a new observation to the input datasets (dataset) |
| dataset_add | Additional dataset<br>The variables specified by the subject_keys parameter are expected.<br>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<br>The condition is evaluated at the additional dataset (dataset_add).<br>For all subjects where it evaluates as TRUE at least once AVALC is set to the true value (true_value) for the new observations. |

For all subjects where it evaluates as FALSE or NA for all observations AVALC is set to the false value (false_value).

For all subjects not present in the additional dataset AVALC is set to the missing value (missing_value).

true_value — True value

For all subjects with at least one observations in the additional dataset (dataset_add) fulfilling the event condition (condition), AVALC is set to the specified value (true_value).

*Default*: "Y"

*Permitted Value*: A character scalar

false_value — False value

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

*Default*: NA_character_

*Permitted Value*: A character scalar

missing_value — Values used for missing information

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

*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)

The (first) argument of the function must expect a character vector and the function must return a numeric vector.

*Default:* yn_to_numeric (see yn_to_numeric() for details)

subject_keys — Variables to uniquely identify a subject

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

set_values_to — Variables to set

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

## Details

1. The additional dataset (dataset_add) is restricted to the observations matching the filter_add condition.
2. For each subject in dataset_adsl a new observation is created.

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

3. The AVAL variable is added and set to aval_fun(AVALC).
4. The variables specified by the set_values_to parameter are added to the new observations.
5. The new observations are added to input dataset.

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

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

## Examples

```
library(dplyr)
library(lubridate)

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

tu <- tibble::tribble(
  ~USUBJID, ~VISIT,      ~TUSTRESC,
  "1",      "SCREENING", "TARGET",
  "1",      "WEEK 1",    "TARGET",
  "1",      "WEEK 5",    "TARGET",
  "1",      "WEEK 9",    "NON-TARGET",
  "2",      "SCREENING", "NON-TARGET",
  "2",      "SCREENING", "NON-TARGET"
) %>%
```

```
  mutate(
    STUDYID = "XX1234",
    TUTESTCD = "TUMIDENT"
  )

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

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>• The variables specified by the `by_vars`,`analysis_var` parameters and `PARAMCD` are expected,<br>• 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 or NA. (e.g. vars(PARAMCD = "TDOSE",PARCAT1 = "OVERALL")). More general expression are not allowed.

*Permitted Values:* List of variable-value pairs

## Details

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

## Value

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

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

If the input datasets contains

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

## Author(s)

Samia Kabi

## See Also

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

## Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(stringr, warn.conflicts = FALSE)
adex <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~AVALC, ~VISIT, ~ASTDT, ~AENDT,
  "1015", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "DOSE", 85, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "DOSE", 82, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1015", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "ADJ", NA, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1017", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
  "1017", "DOSE", 50, NA_character_, "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
  "1017", "DOSE", 65, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02"),
  "1017", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
  "1017", "ADJ", NA, "ADVERSE EVENT", "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
  "1017", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02")
) %>%
 mutate(ASTDTM = ymd_hms(paste(ASTDT, "00:00:00")), AENDTM = ymd_hms(paste(AENDT, "00:00:00")))

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

# average dose in w2-24
adex %>%
  derive_param_exposure(
    by_vars = vars(USUBJID),
    filter = VISIT %in% c("WEEK 2", "WEEK 24"),
    set_values_to = vars(PARAMCD = "AVDW224", PARCAT1 = "WEEK2-24"),
    input_code = "DOSE",
    analysis_var = AVAL,
```

```
    summary_fun = function(x) mean(x, na.rm = TRUE)
  ) %>%
  select(-ASTDTM, -AENDTM)

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

---

`derive_param_first_event` *Add a First Event Parameter*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The `PARAMCD` variable is expected. |
| `dataset_adsl` | ADSL input dataset<br>The variables specified for `subject_keys` are expected. For each observation of the specified dataset a new observation is added to the input dataset. |

dataset_source — Source dataset

All observations in the specified dataset fulfilling the condition specified by filter_source are considered as event.

The variables specified by the subject_keys and date_var parameter are expected.

filter_source — Source filter

All observations in dataset_source fulfilling the specified condition are considered as event.

For subjects with at least one event AVALC is set to "Y", AVAL to 1, and ADT to the first date where the condition is fulfilled.

For all other subjects AVALC is set to "N", AVAL to 0, and ADT to NA.

date_var — Date variable

Date variable in the source dataset (dataset_source). The variable is used to sort the source dataset. ADT is set to the specified variable for events.

subject_keys — Variables to uniquely identify a subject

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

set_values_to — Variables to set

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

check_type — Check uniqueness?

If "warning" or "error" is specified, a message is issued if the observations of the input dataset restricted to the source parameter (source_param) are not unique with respect to the subject keys (subject_key parameter) and ADT.

*Default*: "warning"

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

## Details

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_extreme_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

```
library(dplyr)
library(lubridate)

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

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

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

```
    ANL01FL = "Y"
  )
)

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

---

derive_param_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

```
derive_param_framingham(
  dataset,
  by_vars,
  set_values_to = vars(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

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected.<br>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, chol_code and hdl_code. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values:* list of variables |
| set_values_to | Variables to be set<br>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.<br>*Permitted Values:* List of variable-value pairs |
| sysbp_code | Systolic blood pressure parameter code<br>The observations where PARAMCD equals the specified value are considered as the systolic blood pressure assessments.<br>*Permitted Values:* character value |
| chol_code | Total serum cholesterol code<br>The observations where PARAMCD equals the specified value are considered as the total cholesterol assessments. This must be measured in mg/dL.<br>*Permitted Values:* character value |
| cholhdl_code | HDL serum cholesterol code<br>The observations where PARAMCD equals the specified value are considered as the HDL cholesterol assessments. This must be measured in mg/dL.<br>*Permitted Values:* character value |
| age | Subject age<br>A variable containing the subject's age.<br>*Permitted Values:* A numeric variable name that refers to a subject age column of the input dataset |
| sex | Subject sex<br>A variable containing the subject's sex.<br>*Permitted Values:* A character variable name that refers to a subject sex column of the input dataset |
| smokefl | Smoking status flag<br>A flag indicating smoking status.<br>*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 column 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 indicates 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 D'Agostino, 2008 is: # nolint **For Women:**

| Factor | Amount |
|---|---|
| Age | 2.32888 |
| Total Chol | 1.20904 |
| HDL Chol | -0.70833 |
| Sys BP | 2.76157 |
| Sys BP + Hypertension Meds | 2.82263 |
| Smoker | 0.52873 |
| Non-Smoker | 0 |
| Diabetic | 0.69154 |
| Not Diabetic | 0 |
| Average Risk | 26.1931 |
| Risk Period | 0.95012 |

**For Men:**

| Factor | Amount |
|---|---|
| Age | 3.06117 |
| Total Chol | 1.12370 |
| HDL Chol | -0.93263 |
| Sys BP | 1.93303 |
| Sys BP + Hypertension Meds | 2.99881 |
| Smoker | .65451 |
| Non-Smoker | 0 |
| Diabetic | 0.57367 |

| | |
|---|---|
| Not Diabetic | 0 |
| Average Risk | 23.9802 |
| Risk Period | 0.88936 |

**The equation for calculating risk:**

$$RiskFactors = (log(Age) * AgeFactor) + (log(TotalChol) * TotalCholFactor) + (log(CholHDL) * CholHDLFactor)$$

$$Risk = 100 * (1 - RiskPeriodFactor^{RiskFactors})$$

## Value

The input dataset with the new parameter added

## Author(s)

Alice Ehmann Jack McGavigan Ben Straub

## See Also

compute_framingham()

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

## Examples

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

adcvrisk <- tibble::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",
  "01-701-1015", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 54.91,
  "mg/dL", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 26.72,
  "mg/dL", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 119,
  "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)", 65.55,
  "mg/dL", "BASELINE", 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 = vars(USUBJID, VISIT),
    set_values_to = vars(
      PARAMCD = "FCVD101",
      PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
    ),
    get_unit_expr = AVALU
  )

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

---

`derive_param_map` *Adds a Parameter for Mean Arterial Pressure*

---

## Description

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

## Usage

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

```
  get_unit_expr,
  filter = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected.<br>The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by sysbp_code, diabp_code and hr_code. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values:* list of variables |
| set_values_to | Variables to be set<br>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.<br>*Permitted Values:* List of variable-value pairs |
| sysbp_code | Systolic blood pressure parameter code<br>The observations where PARAMCD equals the specified value are considered as the systolic blood pressure assessments.<br>Permitted Values: character value |
| diabp_code | Diastolic blood pressure parameter code<br>The observations where PARAMCD equals the specified value are considered as the diastolic blood pressure assessments.<br>Permitted Values: character value |
| hr_code | Heart rate parameter code<br>The observations where PARAMCD equals the specified value are considered as the heart rate assessments.<br>Permitted Values: character value |
| get_unit_expr | An expression providing the unit of the parameter<br>The result is used to check the units of the input parameters.<br>Permitted Values: A variable of the input dataset or a function call |
| filter | Filter condition<br>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.<br>*Permitted Values:* a condition |

## Details

The analysis value of the new parameter is derived as

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

if it is based on diastolic and systolic blood pressure and

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

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_extreme_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_first_event(), derive_param_framinghar derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

## Examples

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

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

# Derive MAP based on diastolic and systolic blood pressure
advs %>%
  derive_param_map(
    by_vars = vars(USUBJID, VISIT),
```

```
    set_values_to = vars(
      PARAMCD = "MAP",
      PARAM = "Mean Arterial Pressure (mmHg)"
    ),
    get_unit_expr = extract_unit(PARAM)
  ) %>%
  filter(PARAMCD != "PULSE")

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

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` and the `unit_var` parameter, `PARAMCD`, and `AVAL` are expected.<br>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 `vars(PARAMCD = "MAP")` defines the parameter code for the new parameter.

*Permitted Values:* List of variable-value pairs

`qt_code` QT parameter code

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

Permitted Values: character value

`rr_code` RR parameter code

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

Permitted Values: character value

`get_unit_expr` An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted Values: A variable of the input dataset or a function call

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

*Permitted Values:* a condition

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

`compute_qtc()`

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

## Examples

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

derive_param_qtc(
  adeg,
  by_vars = vars(USUBJID, VISIT),
  method = "Bazett",
  set_values_to = vars(
    PARAMCD = "QTCBR",
    PARAM = "QTcB - Bazett's Correction Formula Rederived (msec)",
    AVALU = "msec"
  ),
  get_unit_expr = AVALU
)

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

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

derive_param_rr *Adds a Parameter for Derived RR (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.

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter, PARAMCD, and AVAL are expected.<br>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<br>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.<br>*Permitted Values:* list of variables |
| set_values_to | Variables to be set<br>The specified variables are set to the specified values for the new observations. For example vars(PARAMCD = "MAP") defines the parameter code for the new parameter.<br>*Permitted Values:* List of variable-value pairs |
| hr_code | HR parameter code<br>The observations where PARAMCD equals the specified value are considered as the heart rate assessments.<br>Permitted Values: character value |
| get_unit_expr | An expression providing the unit of the parameter<br>The result is used to check the units of the input parameters.<br>Permitted Values: A variable of the input dataset or a function call |

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

*Permitted Values:* a condition

## Details

The analysis value of the new parameter is derived as

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_extreme_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_first_event()`, `derive_param_framingha`
`derive_param_map()`, `derive_param_qtc()`, `derive_param_wbc_abs()`, `derive_summary_records()`

## Examples

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

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

```
)
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The `PARAMCD` variable is expected. |
| `dataset_adsl` | ADSL input dataset<br>The variables specified for `start_date`, `start_imputation_flag`, and `subject_keys` are expected. |
| `source_datasets` | Source datasets<br>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<br>If the parameter is specified, separate time to event parameters are derived for each by group.<br>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.<br>The by variables are not included in the output dataset. |
| `start_date` | Time to event origin date<br>The variable `STARTDT` is set to the specified date. The value is taken from the ADSL dataset. |

If the event or censoring date is before the origin date, ADT is set to the origin date.

If the specified variable is imputed, the corresponding date imputation flag must specified for start_imputation_flag.

event_conditions

Sources and conditions defining events

A list of event_source() objects is expected.

censor_conditions

Sources and conditions defining censorings

A list of censor_source() objects is expected.

create_datetime

Create datetime variables?

If set to TRUE, variables ADTM and STARTDTM are created. Otherwise, variables ADT and STARTDT are created.

set_values_to Variables to set

A named list returned by vars() defining the variables to be set for the new parameter, e.g. vars(PARAMCD = "OS", PARAM = "Overall Survival") is expected. The values must be symbols, character strings, numeric values, expressions, or NA.

subject_keys Variables to uniquely identify a subject

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

## Details

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

**Deriving the events:**

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

**Deriving the censoring observations:**

1. For each censoring source dataset the observations as specified by the filter element are selected. Then for each patient the last observation (with respect to date) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied.
3. The CNSR variable is added and set to the censor element.

4. The variables specified by the set_values_to element are added.
5. The selected observations of all censoring source datasets are combined into a single dataset.
6. For each patient the last observation (with respect to the ADT variable) from the single dataset is selected.

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

Finally

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

## Value

The input dataset with the new parameter added

## Author(s)

Stefan Bundfuss

## Examples

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

adsl <- admiral_adsl

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

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

```
    SRCVAR = "LSTALVDT"
  )
)

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

# derive time to adverse event for each preferred term #
adsl <- 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",     "2021-01-03T10:56", 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 = vars(
    EVNTDESC = "AE",
    SRCDOM = "AE",
    SRCVAR = "AESTDTC",
    SRCSEQ = AESEQ
  )
)

eos <- censor_source(
```

```
  dataset_name = "adsl",
  date = EOSDT,
  set_values_to = vars(
    EVNTDESC = "END OF STUDY",
    SRCDOM = "ADSL",
    SRCVAR = "EOSDT"
  )
)

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

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | 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 vars() defining the variables to be set for the new parameter, e.g. vars(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

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

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

$$WhiteBloodCellCount * FractionValue$$

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

## Value

The input dataset with the new parameter added

## Author(s)

Annie Yang

## See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_extreme_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_first_event(), derive_param_framingha
derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_summary_records()

## Examples

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

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

---

# derive_summary_records

*Add New Records Within By Groups Using Aggregation Functions*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | A data frame. |
| by_vars | Variables to consider for generation of groupwise summary records. Providing the names of variables in vars() will create a groupwise summary and generate summary records for the specified groups. |
| filter | Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved.<br>For example,<br>• filter = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars.<br>• 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<br>The specified variables are set to the specified values for the new observations.<br>A list of variable name-value pairs is expected.<br>• LHS refers to a variable.<br>• RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or NA, e.g., vars(PARAMCD = "TDOSE", PARCAT1 = "OVERALL"). More general expression are not allowed. |

## Details

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

## Value

A data frame with derived records appended to original dataset.

## Author(s)

Vignesh Thanikachalam, Ondrej Slama

## See Also

get_summary_records()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_extreme_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_first_event(), derive_param_framingha
derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs()

## Examples

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

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

advs <- tibble::tribble(
  ~USUBJID, ~VSSEQ, ~PARAM, ~AVAL, ~VSSTRESU, ~VISIT, ~VSDTC,
  "XYZ-001-001", 1164, "Weight", 99, "kg", "Screening", "2018-03-19",
  "XYZ-001-001", 1165, "Weight", 101, "kg", "Run-In", "2018-03-26",
  "XYZ-001-001", 1166, "Weight", 100, "kg", "Baseline", "2018-04-16",
  "XYZ-001-001", 1167, "Weight", 94, "kg", "Week 24", "2018-09-30",
  "XYZ-001-001", 1168, "Weight", 92, "kg", "Week 48", "2019-03-17",
  "XYZ-001-001", 1169, "Weight", 95, "kg", "Week 52", "2019-04-14",
)
```

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

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

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

---

`derive_vars_aage` *Derive Analysis Age*

---

## Description

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

## Usage

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

## Arguments

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

## Details

The age is derived as the integer part of the duration from start to end date in the specified unit. When 'years' or 'months' are specified in the out_unit parameter, because of the underlying lubridate::time_length() function that is used here, results are calculated based on the actual calendar length of months or years rather than assuming equal days every month (30.4375 days) or every year (365.25 days).

## Value

The input dataset with AAGE and AAGEU added

## Author(s)

Stefan Bundfuss

## See Also

derive_vars_duration()

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

## Examples

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

derive_vars_aage(data)
```

---

# derive_vars_atc *Derive ATC Class Variables*

---

## Description

Add Anatomical Therapeutic Chemical class variables from FACM to ADCM

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter are required |
| dataset_facm | FACM dataset<br>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<br>*Permitted Values:* list of variables |
| value_var | The variable of dataset_facm containing the values of the transposed variables<br>Default: FASTRESC |

## Value

The input dataset with ATC variables added

## Author(s)

Thomas Neitmann

## See Also

OCCDS Functions: create_query_data(), create_single_dose_dataset(), derive_vars_query(), get_terms_from_db()

## Examples

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

derive_vars_atc(cm, facm)
```

---

# derive_vars_disposition_reason

*Derive a Disposition Reason at a Specific Timepoint*

---

## Description

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

## Usage

```
derive_vars_disposition_reason(
  dataset,
  dataset_ds,
  new_var,
  reason_var,
  new_var_spe = NULL,
  reason_var_spe = NULL,
  format_new_vars = format_reason_default,
  filter_ds,
  subject_keys = vars(STUDYID, USUBJID)
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset |
| dataset_ds | Dataset containing the disposition information (e.g. ds)<br>The dataset must contain:<br>• STUDYID, USUBJID,<br>• The variable(s) specified in the reason_var (and reason_var_spe, if required)<br>• The variables used in filter_ds. |
| new_var | Name of the disposition reason variable<br>A variable name is expected (e.g. DCSREAS). |
| reason_var | The variable used to derive the disposition reason<br>A variable name is expected (e.g. DSDECOD). |
| new_var_spe | Name of the disposition reason detail variable<br>A variable name is expected (e.g. DCSREASP). If new_var_spe is specified, it is expected that reason_var_spe is also specified, otherwise an error is issued.<br>Default: NULL |
| reason_var_spe | The variable used to derive the disposition reason detail<br>A variable name is expected (e.g. DSTERM). If new_var_spe is specified, it is expected that reason_var_spe is also specified, otherwise an error is issued.<br>Default: NULL |
| format_new_vars | The function used to derive the reason(s)<br>This function is used to derive the disposition reason(s) and must follow the below conventions<br>• If only the main reason for discontinuation needs to be derived (i.e. new_var_spe is NULL), the function must have at least one character vector argument, e.g. format_reason <- function(reason) and new_var will be derived as new_var = format_reason(reason_var). Typically, the content of the function would return reason_var or NA depending on the value (e.g. if_else ( reason != "COMPLETED" & !is.na(reason), reason, NA_character_)). DCSREAS = format_reason(DSDECOD) returns DCSREAS = DSDECOD when DSDECOD is not 'COMPLETED' nor NA, NA otherwise. |

- If both the main reason and the details needs to be derived (new_var_spe is specified) the function must have two character vectors argument, e.g. format_reason2 <- function(reason, reason_spe) and new_var will be derived as new_var = format_reason(reason_var), new_var_spe will be derived as new_var_spe = format_reason(reason_var, reason_var_spe). Typically, the content of the function would return reason_var_spe or NA depending on the reason_var value (e.g. if_else ( reason == "OTHER", reason_spe, NA_character_)). DCSREASP = format_reason(DSDECOD, DSTERM) returns DCSREASP = DSTERM when DSDECOD is equal to 'OTHER'.

Default: format_reason_default, see format_reason_default() for details.

filter_ds
: Filter condition for the disposition data.

  Filter used to select the relevant disposition data. It is expected that the filter restricts dataset_ds such that there is at most one observation per patient. An error is issued otherwise.

  Permitted Values: logical expression.

subject_keys
: Variables to uniquely identify a subject

  A list of quosures where the expressions are symbols as returned by vars() is expected.

## Details

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

## Value

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

## Author(s)

Samia Kabi

## See Also

format_reason_default()

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

## Examples

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

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

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

---

`derive_vars_dt` *Derive/Impute a Date from a Date Character Vector*

---

## Description

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

## Usage

```
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.

`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*: `"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 `dtc` value are considered. The possible dates are defined by the missing parts of the `dtc` date (see example below). This ensures that the non-missing parts of the `dtc` date are not changed. A date or date-time object is expected. For example

```
impute_dtc_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.

## Author(s)

Samia Kabi

## 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",
  "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"
)

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

derive_vars_dt(
  adae,
  dtc = AESTDTC,
  new_vars_prefix = "AST",
  highest_imputation = "M",
  min_dates = vars(TRTSDTM)
)

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

derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "mid",
  preserve = TRUE
)
```

---

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

---

## Description

Derive a datetime object (`'--DTM'`) from a date character vector (`'--DTC'`). The date and time can be imputed (see `date_imputation`/`time_imputation` 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",
  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 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".

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

```
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`

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

## Value

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

## Author(s)

Samia Kabi

## 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

```
library(tibble)
library(lubridate)

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

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  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 = vars(DTHDT, DCUTDT)
)

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

```
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  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(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "mid",
  preserve = TRUE
)
```

---

`derive_vars_dtm_to_dt` *Derive Date Variables from Datetime Variables*

---

## Description

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

## Usage

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

## Arguments

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

## Value

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

### Author(s)

Teckla Akinyi

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

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

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

---

## derive_vars_dtm_to_tm *Derive Time Variables from Datetime Variables*

---

### Description

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

### Usage

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

### Arguments

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

## Details

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

## Value

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

## Author(s)

Teckla Akinyi

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

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

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

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

derive_vars_duration *Derive Duration*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>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<br>A date or date-time variable is expected. This variable must be present in specified input dataset.<br>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<br>A date or date-time variable is expected. This variable must be present in specified input dataset.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. |
| in_unit | Input unit<br>See floor_in and add_one parameter for details.<br>Default: 'days'<br>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds' |

| | |
|---|---|
| out_unit | Output unit<br>The duration is derived in the specified unit<br>Default: 'days'<br>Permitted Values: 'years', 'months', 'days', 'hours', 'minutes', 'seconds' |
| floor_in | Round down input dates?<br>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.<br>Default: 'TRUE"<br>Permitted Values: TRUE, FALSE |
| add_one | Add one input unit?<br>If the duration is non-negative, one input unit is added. I.e., the duration can not be zero.<br>Default: TRUE Permitted Values: TRUE, FALSE |
| trunc_out | Return integer part<br>The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.<br>Default: FALSE<br>Permitted Values: TRUE, FALSE |

## Details

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

## Value

The input dataset with the duration and unit variable added

## Author(s)

Stefan Bundfuss

## See Also

compute_duration()

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_dy()

## Examples

```
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("2019-11-11T04:00:00"),
  "P04", 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_vars_dy *Derive Relative Day Variables*

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

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

## Author(s)

Teckla Akinyi

## 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(lubridate)
library(dplyr)

datain <- tibble::tribble(
  ~TRTSDTM, ~ASTDTM, ~AENDT,
  "2014-01-17T23:59:59", "2014-01-18T13:09:O9", "2014-01-20"
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDT = ymd(AENDT)
  )

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

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

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

---

# derive_vars_last_dose *Derive Last Dose*

---

## Description

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

## Usage

```
derive_vars_last_dose(
  dataset,
  dataset_ex,
  filter_ex = NULL,
```

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

## Arguments

| | |
|---|---|
| dataset | Input dataset. The variables specified by the by_vars and analysis_date parameters are expected. |
| dataset_ex | Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars parameter are expected. |
| filter_ex | Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL. |
| by_vars | Variables to join by (created by dplyr::vars). |
| dose_id | Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars(). |
| dose_date | The EX dose date variable. 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 dplyr::vars (e.g. vars(VISIT)), or named list returned by vars() (e.g. vars(LSTEXVIS = VISIT)). If set to NULL, then all variables from dataset_ex are kept without renaming. Defaults to NULL. |
| traceability_vars | A named list returned by vars() listing the traceability variables, e.g. vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ). The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables. |

## Details

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
- analysis_date: time is imputed to 23:59:59 if the variable is a date variable

The last dose records are identified as follows:

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

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

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

## Value

Input dataset with EX source variables from last dose added.

## Author(s)

Ondrej Slama, Annie Yang

## See Also

derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_last_dose_grp(), create_single_dose_dataset()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

# create datetime variables in input datasets
ex_single <- derive_vars_dtm(
  head(ex_single, 100),
  dtc = EXENDTC,
```

```
  new_vars_prefix = "EXEN",
  flag_imputation = "none"
)

adae <- admiral_ae %>%
  head(100) %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AST",
    highest_imputation = "M"
  )

# add last dose vars
adae %>%
  derive_vars_last_dose(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXENDTM),
    new_vars = vars(EXDOSE, EXTRT, EXSEQ, EXENDTC, VISIT),
    dose_date = EXENDTM,
    analysis_date = ASTDTM
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, EXDOSE, EXTRT, EXENDTC, EXSEQ, VISIT)

# or with traceability variables
adae %>%
  derive_vars_last_dose(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXENDTM),
    new_vars = vars(EXDOSE, EXTRT, EXSEQ, EXENDTC, VISIT),
    dose_date = EXENDTM,
    analysis_date = ASTDTM,
   traceability_vars = dplyr::vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXENDTC")
  ) %>%
 select(STUDYID, USUBJID, AESEQ, AESTDTC, EXDOSE, EXTRT, EXENDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR)
```

---

`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).

## Usage

```
derive_vars_merged(
```

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter are expected. |
| dataset_add | Additional dataset<br>The variables specified by the by_vars, the new_vars, and the order parameter are expected. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by vars() |
| order | Sort order<br>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.<br>*Default*: NULL<br>*Permitted Values*: list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) or NULL |
| new_vars | Variables to add<br>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 = vars(<new name> = <old name>).<br>For example new_vars = vars(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset.<br>And new_vars = vars(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.<br>If the parameter is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added.<br>*Default*: NULL<br>*Permitted Values*: list of variables created by vars() |
| mode | Selection mode<br>Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null. |

If the order parameter is not specified, the mode parameter is ignored.

*Default*: NULL

*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 parameter is not specified, all observations are considered.

*Default*: NULL

*Permitted Values*: a condition

match_flag — Match flag

If the parameter is specified (e.g., match_flag = FLAG), the specified variable (e.g., FLAG) is added to the input dataset. This variable will be TRUE for all selected records from dataset_add which are merged into the input dataset, and NA otherwise.

*Default*: NULL

*Permitted Values*: Variable name

check_type — Check uniqueness?

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

*Default*: "warning"

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

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_transposed(), get_summary_records()

## Examples

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

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

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

# Derive treatment start datetime (TRTSDTM)
data(admiral_ex)

## Impute exposure start date to first date/time
ex_ext <- derive_vars_dtm(
  admiral_ex,
  dtc = EXSTDTC,
  new_vars_prefix = "EXST",
  highest_imputation = "M",
)

## Add first exposure datetime and imputation flags to adsl
```

```
derive_vars_merged(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = ex_ext,
  by_vars = vars(STUDYID, USUBJID),
  new_vars = vars(TRTSDTM = EXSTDTM, TRTSDTF = EXSTDTF, TRTSTMF = EXSTTMF),
  order = vars(EXSTDTM),
  mode = "first"
)

# Derive treatment start datetime (TRTSDTM)
data(admiral_ex)

## Impute exposure start date to first date/time
ex_ext <- derive_vars_dtm(
  admiral_ex,
  dtc = EXSTDTC,
  new_vars_prefix = "EXST",
  highest_imputation = "M",
)

## Add first exposure datetime and imputation flags to adsl
derive_vars_merged(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = ex_ext,
  filter_add = !is.na(EXSTDTM),
  by_vars = vars(STUDYID, USUBJID),
  new_vars = vars(TRTSDTM = EXSTDTM, TRTSDTF = EXSTDTF, TRTSTMF = EXSTTMF),
  order = vars(EXSTDTM),
  mode = "first"
)

# Derive treatment end datetime (TRTEDTM)
## Impute exposure end datetime to last time, no date imputation
ex_ext <- derive_vars_dtm(
  admiral_ex,
  dtc = EXENDTC,
  new_vars_prefix = "EXEN",
  time_imputation = "last",
)

## Add last exposure datetime and imputation flag to adsl
derive_vars_merged(
  select(admiral_dm, STUDYID, USUBJID),
  dataset_add = ex_ext,
  filter_add = !is.na(EXENDTM),
  by_vars = vars(STUDYID, USUBJID),
  new_vars = vars(TRTEDTM = EXENDTM, TRTETMF = EXENTMF),
  order = vars(EXENDTM),
  mode = "last"
)
```

`derive_vars_merged_dt` *Merge a (Imputed) Date Variable*

## Description

**[Deprecated]**

This function is *deprecated*, please use `derive_vars_dt()` and `derive_vars_merged()` instead.

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

## Usage

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

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter are expected. |
| `dataset_add` | Additional dataset<br>The variables specified by the `by_vars`, the `dtc`, and the `order` parameter are expected. |
| `by_vars` | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by `vars()` |

`order` Sort order

If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. The imputed date variable can be specified as well (see examples below).

Please note that `NA` is considered as the last value. I.e., if a order variable is `NA` and `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'.

*Default*: `NULL`

*Permitted Values*: list of variables or `desc(<variable>)` function calls created by `vars()`, e.g., `vars(ADT, desc(AVAL)` or `NULL`

`new_vars_prefix`

Prefix used for the output variable(s).

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

`filter_add` Filter for additional dataset (`dataset_add`)

Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered.

*Default*: `NULL`

*Permitted Values*: a condition

`mode` Selection mode

Determines if the first or last observation is selected. If the `order` parameter is specified, `mode` must be non-null.

If the `order` parameter is not specified, the `mode` parameter is ignored.

*Default*: `NULL`

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

`dtc` The `'--DTC'` date to impute

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

`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 dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_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

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.

*Default*: "warning"

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

## Value

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

## Author(s)

Stefan Bundfuss

---

`derive_vars_merged_dtm`

*Merge a (Imputed) Datetime Variable*

---

## Description

**[Deprecated]**

This function is *deprecated*, please use `derive_vars_dtm()` and `derive_vars_merged()` instead.

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

## Usage

```
derive_vars_merged_dtm(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars_prefix,
  filter_add = NULL,
  mode = NULL,
  dtc,
  date_imputation = NULL,
  time_imputation = "00:00:00",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
```

```
  preserve = FALSE,
  check_type = "warning",
  duplicate_msg = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter are expected. |
| dataset_add | Additional dataset<br>The variables specified by the by_vars, the dtc, and the order parameter are expected. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by vars() |
| order | Sort order<br>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. The imputed datetime variable can be specified as well (see examples below).<br>*Default*: NULL<br>*Permitted Values*: list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) or NULL |
| new_vars_prefix | Prefix used for the output variable(s).<br>A character scalar is expected. For the date variable "DT" is appended to the specified prefix, for the date imputation flag "DTF", and for the time imputation flag "TMF". I.e., for new_vars_prefix = "AST" the variables ASTDT, ASTDTF, and ASTTMF are created. |
| filter_add | Filter for additional dataset (dataset_add)<br>Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered.<br>*Default*: NULL<br>*Permitted Values*: a condition |
| mode | Selection mode<br>Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null.<br>If the order parameter is not specified, the mode parameter is ignored.<br>*Default*: NULL<br>*Permitted Values*: "first", "last", NULL |
| dtc | The '--DTC' date to impute<br>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. |

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".

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

```
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

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.

*Default*: "warning"

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

## Value

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

## Author(s)

Stefan Bundfuss

---

`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

```
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<br>The variables specified by the `by_vars` parameter are expected. |
| `dataset_add` | Lookup table<br>The variables specified by the `by_vars` parameter are expected. |
| `by_vars` | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by `vars()` |
| `order` | Sort order<br>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.<br>*Default*: `NULL`<br>*Permitted Values*: list of variables or `desc(<variable>)` function calls created by `vars()`, e.g., `vars(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 = vars(<new name> = <old name>).

  For example new_vars = vars(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset.

  And new_vars = vars(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.

  If the parameter is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added.

  *Default*: NULL

  *Permitted Values*: list of variables created by vars()

mode
: Selection mode

  Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null.

  If the order parameter is not specified, the mode parameter is ignored.

  *Default*: NULL

  *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 parameter is not specified, all observations are considered.

  *Default*: NULL

  *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.

  *Default*: "warning"

  *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).

## Author(s)

Annie Yang

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_vs")
param_lookup <- tibble::tribble(
  ~VSTESTCD, ~VSTEST, ~PARAMCD, ~PARAM,
  "SYSBP", "Systolic Blood Pressure", "SYSBP", "Systolic Blood Pressure (mmHg)",
  "WEIGHT", "Weight", "WEIGHT", "Weight (kg)",
  "HEIGHT", "Height", "HEIGHT", "Height (cm)",
  "TEMP", "Temperature", "TEMP", "Temperature (C)",
  "MAP", "Mean Arterial Pressure", "MAP", "Mean Arterial 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 = admiral_vs,
  dataset_add = param_lookup,
  by_vars = vars(VSTESTCD),
  new_vars = vars(PARAMCD),
  print_not_mapped = TRUE
)
```

---

# derive_vars_query *Derive Query Variables*

---

## Description

Derive Query Variables

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset. |
| dataset_queries | A dataset containing required columns VAR_PREFIX, QUERY_NAME, TERM_LEVEL, TERM_NAME, TERM_ID, and optional columns QUERY_ID, QUERY_SCOPE, QUERY_SCOPE_NUM. The content of the dataset will be verified by assert_valid_queries(). create_query_data() can be used to create the dataset. |

## Details

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

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

## Value

The input dataset with query variables derived.

## Author(s)

Ondrej Slama, Shimeng Huang

## See Also

create_query_data() assert_valid_queries()

OCCDS Functions: create_query_data(), create_single_dose_dataset(), derive_vars_atc(), get_terms_from_db()

## Examples

```
data("queries")
adae <- tibble::tribble(
  ~USUBJID, ~ASTDTM, ~AETERM, ~AESEQ, ~AEDECOD, ~AELLT, ~AELLTCD,
  "01", "2020-06-02 23:59:59", "ALANINE AMINOTRANSFERASE ABNORMAL",
  3, "Alanine aminotransferase abnormal", NA_character_, NA_integer_,
  "02", "2020-06-05 23:59:59", "BASEDOW'S DISEASE",
```

```
  5, "Basedow's disease", NA_character_, 1L,
  "03", "2020-06-07 23:59:59", "SOME TERM",
  2, "Some query", "Some term", NA_integer_,
  "05", "2020-06-09 23:59:59", "ALVEOLAR PROTEINOSIS",
  7, "Alveolar proteinosis", NA_character_, NA_integer_
)
derive_vars_query(adae, queries)
```

---

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

---

## Description

**[Deprecated]**

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

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

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

## Usage

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

## Arguments

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

## Value

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

## Author(s)

Vignesh Thanikachalam

derive_vars_transposed *Derive Variables by Transposing and Merging a Second Dataset*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter are required |
| dataset_merge | Dataset to transpose and merge<br>The variables specified by the by_vars, key_var and value_var parameters are expected |
| by_vars | Keys used to merge dataset_merge with dataset |
| key_var | The variable of dataset_merge containing the names of the transposed variables |
| value_var | The variable of dataset_merge containing the values of the transposed variables |
| filter | Expression used to restrict the records of dataset_merge prior to transposing |

## Details

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

## Value

The input dataset with transposed variables from dataset_merge added

## Author(s)

Thomas Neitmann

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), get_summary_records()

## Examples

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

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

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

derive_var_ady *Derive Analysis Study Day*

## Description

**[Deprecated]**

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

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

## Usage

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

## Arguments

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

## Details

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

## Value

The input dataset with `ADY` column added

## Author(s)

Stefan Bundfuss

derive_var_aendy *Derive Analysis End Relative Day*

## Description

**[Deprecated]**

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

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

## Usage

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

## Arguments

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

## Details

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

## Value

The input dataset with `AENDY` column added

## Author(s)

Stefan Bundfuss

## See Also

Other deprecated: `derive_derived_param()`

## derive_var_agegr_fda *Derive Age Groups*

### Description

**[Deprecated]**

These functions are *deprecated*.

### Usage

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

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

### Arguments

| | |
|---|---|
| dataset | Input dataset |
| age_var | AGE variable |
| age_unit | AGE unit variable |
| new_var | New variable to create inside dataset |

### Author(s)

Ondrej Slama

## derive_var_age_years *Derive Age in Years*

### Description

Derive Age in Years

### Usage

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

### Arguments

| | |
|---|---|
| dataset | Input dataset. |
| age_var | AGE variable. |

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

## Details

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

## Value

The input dataset with new_var parameter added in years.

## Author(s)

Michael Thorpe

## See Also

ADSL Functions that returns variable appended to dataset: derive_var_disposition_status(), derive_var_dthcaus(), derive_var_extreme_dtm(), derive_var_extreme_dt(), derive_vars_aage(), derive_vars_disposition_reason()

## Examples

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

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

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

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

derive_var_analysis_ratio *Derive Ratio Variable*

## Description

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

## Usage

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

## Arguments

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

## Details

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

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

## Value

The input dataset with a ratio variable appended

## Author(s)

Ben Straub

## See Also

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

## Examples

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

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

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

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

---

`derive_var_anrind` *Derive Reference Range Indicator*

---

## Description

Derive Reference Range Indicator

## Usage

```
derive_var_anrind(dataset)
```

## Arguments

| | |
|---|---|
| `dataset` | The input dataset |

## Details

`ANRIND` is set to

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

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

## Value

The input dataset with additional column ANRIND

## Author(s)

Thomas Neitmann

## See Also

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

## Examples

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

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

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

---

# derive_var_astdy *Derive Analysis Start Relative Day*

---

## Description

**[Deprecated]**

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

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

## Usage

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

## Arguments

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

## Details

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

## Value

The input dataset with ASTDY column added

## Author(s)

Stefan Bundfuss

---

# derive_var_atirel *Derive Time Relative to Reference*

---

## Description

**[Deprecated]**

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

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

## Usage

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

### Arguments

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

### Details

ATIREL is set to:

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

### Value

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

### Author(s)

Teckla Akinyi

---

## derive_var_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<br>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 low 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

## Author(s)

Gordon Miller

## See Also

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

## Examples

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

adlb <- tibble::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)
```

---

`derive_var_atoxgr_dir` *Derive Lab Toxicity Grade 0 - 4*

---

## Description

Derives a character lab grade based on severity/toxicity criteria.

## Usage

```
derive_var_atoxgr_dir(
  dataset,
  new_var,
  tox_description_var,
  meta_criteria = atoxgr_criteria_ctcv4,
  criteria_direction,
  get_unit_expr
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input data set<br>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)<br>Default: `atoxgr_criteria_ctcv4`<br>admiral metadata data set `atoxgr_criteria_ctcv4` implements Common Terminology Criteria for Adverse Events (CTCAE) v4.0<br>The metadata should have the following variables:<br>• `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.<br>• `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.

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).

## 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

## Author(s)

Gordon Miller

## See Also

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

## Examples

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

data <- tibble::tribble(
  ~ATOXDSCL,                    ~AVAL,  ~ANRLO,   ~ANRHI, ~PARAM,
  "Hypoglycemia",               119,    4,        7,      "Glucose (mmol/L)",
  "Hypoglycemia",               120,    4,        7,      "Glucose (mmol/L)",
```

```
  "Anemia",                      129,    120,      180,    "Hemoglobin (g/L)",
  "White blood cell decreased",  10,     5,        20,     "White blood cell (10^9/L)",
  "White blood cell decreased",  15,     5,        20,     "White blood cell (10^9/L)",
  "Anemia",                      140,    120,      180,    "Hemoglobin (g/L)"
)

derive_var_atoxgr_dir(data,
  new_var = ATOXGRL,
  tox_description_var = ATOXDSCL,
  meta_criteria = atoxgr_criteria_ctcv4,
  criteria_direction = "L",
  get_unit_expr = extract_unit(PARAM)
)

data <- tibble::tribble(
  ~ATOXDSCH,                     ~AVAL,  ~ANRLO,   ~ANRHI, ~PARAM,
  "Hyperglycemia",               119,    4,        7,      "Glucose (mmol/L)",
  "Hyperglycemia",               120,    4,        7,      "Glucose  (mmol/L)",
  "GGT increased",             129,    0,       30,    "Gamma Glutamyl Transferase (U/L)",
  "Lymphocyte count increased",  4,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "Lymphocyte count increased",  2,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "GGT increased",             140,    120,     180,    "Gamma Glutamyl Transferase (U/L)"
)

derive_var_atoxgr_dir(data,
  new_var = ATOXGRH,
  tox_description_var = ATOXDSCH,
  meta_criteria = atoxgr_criteria_ctcv4,
  criteria_direction = "H",
  get_unit_expr = extract_unit(PARAM)
)
```

---

`derive_var_base` *Derive Baseline Variables*

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

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

## Author(s)

Thomas Neitmann

## 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_basetype(), derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()

## Examples

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

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

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

## Derive `BNRIND` variable from `ANRIND`
if (FALSE) {
  derive_var_base(
    dataset,
    by_vars = vars(USUBJID, PARAMCD),
    source_var = ANRIND,
    new_var = BNRIND
  )
}
```

---

`derive_var_basetype` *Derive BASETYPE Variable*

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

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

## Value

The input dataset with variable BASETYPE added

## Author(s)

Thomas Neitmann

## 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_ontrtfl(), derive_var_pchg(), derive_var_shift()

## Examples

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

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


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

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

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

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

print(bds_with_basetype, n = Inf)

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

---

`derive_var_chg` *Derive Change from Baseline*

---

## Description

Derive change from baseline (`CHG`) in a BDS dataset

## Usage

```
derive_var_chg(dataset)
```

## Arguments

| | |
|---|---|
| `dataset` | The input dataset. Required variables are `AVAL` and `BASE`. |

## Details

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

## Value

The input dataset with an additional column named `CHG`

## Author(s)

Thomas Neitmann

## See Also

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_basetype(), derive_var_base(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift()

## Examples

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

---

# derive_var_confirmation_flag *Derive Confirmation 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

```
derive_var_confirmation_flag(
  dataset,
  by_vars,
  order,
  new_var,
  join_vars,
  join_type,
  first_cond = NULL,
  filter,
  true_value = "Y",
  false_value = NA_character_,
  check_type = "warning"
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>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.

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 = vars(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

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` = `vars(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>        <dbl> <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` = `vars(AVISITN)` is restricted to

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <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.

## Author(s)

Stefan Bundfuss

## See Also

`filter_confirmation()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_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_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_worst_flag()`, `derive_vars_last_dose()`, `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_confirmation_flag(
  adae,
  new_var = ALCOVFL,
  by_vars = vars(USUBJID),
  join_vars = vars(ACOVFL, ADY),
  join_type = "all",
  order = vars(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_confirmation_flag(
  data,
  by_vars = vars(USUBJID),
  new_var = CONFFL,
  join_vars = vars(AVALC, AVISITN),
  join_type = "after",
  order = vars(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_confirmation_flag(
  data,
  by_vars = vars(USUBJID),
  join_vars = vars(AVALC),
  join_type = "after",
  order = vars(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"
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"
)

derive_var_confirmation_flag(
  data,
  by_vars = vars(USUBJID),
  join_vars = vars(AVALC, ADY),
  join_type = "after",
  order = vars(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") >
```

```
        max_cond(var = ADY.join, cond = AVALC.join == "PR") |
        count_vals(var = AVALC.join, val = "CR") == 0
    )
)
```

---

derive_var_disposition_status *Derive a Disposition Status at a Specific Timepoint*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| dataset | Input dataset. |
| dataset_ds | Dataset containing the disposition information (e.g.: ds).<br>It must contain:<br>• STUDYID, USUBJID,<br>• The variable(s) specified in the status_var<br>• The variables used in filter_ds. |
| new_var | Name of the disposition status variable.<br>A variable name is expected (e.g. EOSSTT). |
| status_var | The variable used to derive the disposition status.<br>A variable name is expected (e.g. DSDECOD). |
| format_new_var | The format used to derive the status.<br>Default: format_eoxxstt_default() defined as: |

```
format_eoxxstt_default <- function(status) {
  case_when(
   status %in% c("SCREEN FAILURE", "SCREENING NOT COMPLETED") ~ "NOT STARTED",
    status == "COMPLETED" ~ "COMPLETED",
   !status %in% c("COMPLETED", "SCREEN FAILURE", "SCREENING NOT COMPLETED")
```

```
    & !is.na(status) ~ "DISCONTINUED",
    TRUE ~ "ONGOING"
  )
}
```

where `status` is the `status_var`.

`filter_ds` Filter condition for the disposition data.

one observation per patient. An error is issued otherwise.

Permitted Values: logical expression.

`subject_keys` Variables to uniquely identify a subject

A list of quosures where the expressions are symbols as returned by `vars()` is expected.

## Value

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

## Author(s)

Samia Kabi

## See Also

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()`, `derive_vars_disposition_reason()`

## Examples

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

# Default derivation: EOSSTT =
#- NOT STARTED when status_var is SCREEN FAILURE or SCREENING NOT COMPLETED
#- COMPLETED when status_var is COMPLETED
#- DISCONTINUED when status_var is not COMPLETED nor SCREEN FAILURE nor
#  SCREENING NOT COMPLETED nor NA
#- ONGOING otherwise

admiral_dm %>%
  derive_var_disposition_status(
    dataset_ds = admiral_ds,
    new_var = EOSSTT,
    status_var = DSDECOD,
```

```
    filter_ds = DSCAT == "DISPOSITION EVENT"
  ) %>%
  select(STUDYID, USUBJID, EOSSTT)

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

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

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

---

`derive_var_dthcaus` *Derive Death Cause*

---

## Description

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

## Usage

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

dthcaus_source(
```

```
  dataset_name,
  filter,
  date,
  order = NULL,
  mode = "first",
  dthcaus,
  traceability_vars = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset.<br>The variables specified by subject_keys are required. |
| ... | Objects of class "dthcaus_source" created by dthcaus_source(). |
| source_datasets | A named list containing datasets in which to search for the death cause |
| subject_keys | Variables to uniquely identify a subject<br>A list of quosures where the expressions are symbols as returned by vars() is expected. |
| 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 to be used for sorting dataset. |
| order | Sort order<br>Additional variables to be used for sorting the dataset which is ordered by the date and order. Can be used to avoid duplicate record warning.<br>*Default*: NULL<br>*Permitted Values*: list of variables or desc(<variable>) function calls created by vars(), e.g., vars(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 or a string literal — if a variable name, e.g., AEDECOD, it is the variable in the source dataset to be used to assign values to DTHCAUS; if a string literal, e.g. "Adverse Event", it is the fixed value to be assigned to DTHCAUS. |
| traceability_vars | A named list returned by vars() listing the traceability variables, e.g. vars(DTHDOM = "DS", DTHSEQ = DSSEQ). The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables. |

## 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

`derive_var_dthcaus()` returns the input dataset with `DTHCAUS` variable added.

`dthcaus_source()` returns an object of class "dthcaus_source".

## Functions

- `dthcaus_source`: Create objects of class "dthcaus_source"

## Author(s)

Shimeng Huang, Samia Kabi, Thomas Neitmann, Tamara Senior

## See Also

ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_var_disposition_status()`, `derive_var_extreme_dtm()`, `derive_var_extreme_dt()`, `derive_vars_aage()`, `derive_vars_disposition_reason()`

Source Specifications: `assert_db_requirements()`, `assert_terms()`, `assert_valid_queries()`, `censor_source()`, `date_source()`, `death_event`, `event_source()`, `extend_source_datasets()`, `filter_date_sources()`, `format.sdg_select()`, `format.smq_select()`, `list_tte_source_objects()`, `params()`, `query()`, `sdg_select()`, `smq_select()`, `tte_source()`, `validate_query()`, `validate_sdg_select()`, `validate_smq_select()`

## Examples

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

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"
) %>%
  mutate(
    AEDTHDT = ymd(AEDTHDTC)
  )
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", "PAT02", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
  "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
) %>%
  mutate(
    DSSTDT = ymd(DSSTDTC)
```

```
  )

# Derive `DTHCAUS` only - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDT,
  mode = "first",
  dthcaus = AEDECOD
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDT,
  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 = AEDTHDT,
  mode = "first",
  dthcaus = AEDECOD,
  traceability_vars = vars(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDT,
  mode = "first",
  dthcaus = DSTERM,
  traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive `DTHCAUS` as above - now including post-study deaths with different `DTHCAUS` value
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDT,
  mode = "first",
  dthcaus = AEDECOD,
  traceability_vars = vars(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
```

```
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDT,
  mode = "first",
  dthcaus = DSTERM,
  traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

src_ds_post <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & DSTERM == "POST STUDY REPORTING OF DEATH",
  date = DSSTDT,
  mode = "first",
  dthcaus = "POST STUDY: UNKNOWN CAUSE",
  traceability_vars = vars(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(adsl, src_ae, src_ds, src_ds_post, source_datasets = list(ae = ae, ds = ds))
```

---

`derive_var_extreme_dt` *Derive First or Last Date from Multiple Sources*

---

## Description

Add the first or last date from multiple sources to the dataset, e.g., the last known alive date (LSTALVDT).

## Usage

```
derive_var_extreme_dt(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = vars(STUDYID, USUBJID)
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by `subject_keys` are required. |
| `new_var` | Name of variable to create |
| `...` | Source(s) of dates. One or more `date_source()` objects are expected. |
| `source_datasets` | A named `list` containing datasets in which to search for the first or last date |

| | |
|---|---|
| mode | Selection mode (first or last)<br>If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected.<br>Permitted Values: "first", "last" |
| subject_keys | Variables to uniquely identify a subject<br>A list of quosures where the expressions are symbols as returned by vars() is expected. |

## Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the filter element are selected. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable specified by the date element.
3. The variables specified by the traceability_vars element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.
6. The time part is removed from the new variable.

## Value

The input dataset with the new variable added.

## Author(s)

Stefan Bundfuss, Thomas Neitmann

## See Also

date_source(), derive_var_extreme_dtm(), derive_vars_merged()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_var_disposition_status(
derive_var_dthcaus(), derive_var_extreme_dtm(), derive_vars_aage(), derive_vars_disposition_reason()

## Examples

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

# derive last known alive date (LSTALVDT)
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDT
```

```
)
ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDT
)

ae_ext <- admiral_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 = LBDT,
  filter = !is.na(LBDT),
)

lb_ext <- derive_vars_dt(
  admiral_lb,
  dtc = LBDTC,
  new_vars_prefix = "LB"
)

adsl_date <- date_source(dataset_name = "adsl", date = TRTEDT)

admiral_dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT)

# derive last alive date and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDT,
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
```

```
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDT,
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)
lb_date <- date_source(
  dataset_name = "lb",
  date = LBDT,
  filter = !is.na(LBDT),
  traceability_vars = vars(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

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

admiral_dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT, LALVDOM, LALVSEQ, LALVVAR)
```

---

## derive_var_extreme_dtm

*Derive First or Last Datetime from Multiple Sources*

---

## Description

Add the first or last datetime from multiple sources to the dataset, e.g., the last known alive datetime (LSTALVDTM).

## Usage

```
derive_var_extreme_dtm(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = vars(STUDYID, USUBJID)
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by subject_keys are required. |
| new_var | Name of variable to create |
| ... | Source(s) of dates. One or more date_source() objects are expected. |
| source_datasets | A named list containing datasets in which to search for the first or last date |
| mode | Selection mode (first or last)<br>If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected.<br>Permitted Values: "first", "last" |
| subject_keys | Variables to uniquely identify a subject<br>A list of quosures where the expressions are symbols as returned by vars() is expected. |

## Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the filter element are selected. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable specified by the date element. If this is a date variable (rather than datetime), then the time is imputed as "00:00:00".
3. The variables specified by the traceability_vars element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.

## Value

The input dataset with the new variable added.

## Author(s)

Stefan Bundfuss, Thomas Neitmann

## See Also

date_source(), derive_var_extreme_dt(), derive_vars_merged()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_var_disposition_status(
derive_var_dthcaus(), derive_var_extreme_dt(), derive_vars_aage(), derive_vars_disposition_reason()

## Examples

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

# derive last known alive datetime (LSTALVDTM)
ae_start <- date_source(
  dataset_name = "ae",
  date = AESTDTM
)
ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTM
)

ae_ext <- admiral_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"
  )

lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTM,
  filter = !is.na(LBDTM)
)

lb_ext <- derive_vars_dtm(
  admiral_lb,
  dtc = LBDTC,
  new_vars_prefix = "LB"
)
```

```
adsl_date <- date_source(dataset_name = "adsl", date = TRTEDTM)

admiral_dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = 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 = AESTDTM,
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = AEENDTM,
  traceability_vars = vars(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)
lb_date <- date_source(
  dataset_name = "lb",
  date = LBDTM,
  filter = !is.na(LBDTM),
  traceability_vars = vars(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDTM,
  traceability_vars = vars(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDTM"
```

```
  )
)

admiral_dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = admiral_adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM, LALVDOM, LALVSEQ, LALVVAR)
```

---

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

---

## Description

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

## Usage

```
derive_var_extreme_flag(
  dataset,
  by_vars,
  order,
  new_var,
  mode,
  filter = deprecated(),
  check_type = "warning"
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter are expected. |
| `by_vars` | Grouping variables<br>Permitted Values: list of variables |
| `order` | Sort order<br>The first or last observation is determined with respect to the specified order.<br>Permitted Values: list of variables or functions of variables |

| | |
|---|---|
| new_var | Variable to add<br>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.<br>Permitted Values: list of name-value pairs |
| mode | Flag mode<br>Determines of the first or last observation is flagged.<br>Permitted Values: "first", "last" |
| filter | Deprecated, please use restrict_derivation() instead (see examples). |
| check_type | Check uniqueness?<br>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.<br>Default: "warning"<br>Permitted Values: "none", "warning", "error" |

## Details

For each group (with respect to the variables specified for the by_vars parameter), new_var is set to "Y" for the first or last observation (with respect to the order specified for the order parameter and the flag mode specified for the mode parameter). Only observations included by the filter parameter are considered for flagging. Otherwise, new_var is set to NA. Thus, the direction of "worst" is considered fixed for all parameters in the dataset depending on the order and the mode, i.e. for every parameter the first or last record will be flagged across the whole dataset.

## Value

The input dataset with the new flag variable added

## Author(s)

Stefan Bundfuss

## See Also

derive_var_worst_flag()

General Derivation Functions for all ADaMs that returns variable appended to dataset: 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_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

# Flag last value for each patient, test, and visit, baseline observations are ignored
```

```
admiral_vs %>%
  restrict_derivation(
    derivation = derive_var_extreme_flag,
    args = params(
      by_vars = vars(USUBJID, VSTESTCD, VISIT),
      order = vars(VSTPTNUM),
      new_var = LASTFL,
      mode = "last"
    ),
    filter = VISIT != "BASELINE"
  ) %>%
  arrange(USUBJID, VSTESTCD, VISITNUM, VSTPTNUM) %>%
  select(USUBJID, VSTESTCD, VISIT, VSTPTNUM, VSSTRESN, LASTFL)

# Baseline (ABLFL) examples:

input <- tibble::tribble(
  ~STUDYID, ~USUBJID,  ~PARAMCD,     ~AVISIT,                  ~ADT, ~AVAL,    ~DTYPE,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-27"),  15.0,        NA,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-25"),  14.0,        NA,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-23"),  15.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM01",    "WEEK 1", as.Date("2021-04-27"),  10.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM01",    "WEEK 2", as.Date("2021-04-30"),  12.0,        NA,
  "TEST01",  "PAT02", "PARAM01", "SCREENING", as.Date("2021-04-27"),  15.0, "AVERAGE",
  "TEST01",  "PAT02", "PARAM01",  "BASELINE", as.Date("2021-04-25"),  14.0, "AVERAGE",
  "TEST01",  "PAT02", "PARAM01",  "BASELINE", as.Date("2021-04-23"),  15.0, "AVERAGE",
  "TEST01",  "PAT02", "PARAM01",    "WEEK 1", as.Date("2021-04-27"),  10.0, "AVERAGE",
  "TEST01",  "PAT02", "PARAM01",    "WEEK 2", as.Date("2021-04-30"),  12.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-27"),  15.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-25"),  14.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-23"),  15.0,        NA,
  "TEST01",  "PAT01", "PARAM02",  "BASELINE", as.Date("2021-04-27"),  10.0, "AVERAGE",
  "TEST01",  "PAT01", "PARAM02",    "WEEK 2", as.Date("2021-04-30"),  12.0,        NA,
  "TEST01",  "PAT02", "PARAM02", "SCREENING", as.Date("2021-04-27"),  15.0,        NA,
  "TEST01",  "PAT02", "PARAM02",  "BASELINE", as.Date("2021-04-25"),  14.0,        NA,
  "TEST01",  "PAT02", "PARAM02",    "WEEK 1", as.Date("2021-04-23"),  15.0,        NA,
  "TEST01",  "PAT02", "PARAM02",    "WEEK 1", as.Date("2021-04-27"),  10.0,        NA,
  "TEST01",  "PAT02", "PARAM02",  "BASELINE", as.Date("2021-04-30"),  12.0,        NA
)

# Last observation
restrict_derivation(
  input,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD),
    order = vars(ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
)
```

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

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

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

# OCCURDS Examples
data("admiral_ae")

# Most severe AE first occurrence per patient
admiral_ae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = vars(USUBJID),
    order = vars(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first"
```

```
  ) %>%
  arrange(USUBJID, AESTDY, AESEQ) %>%
  select(USUBJID, AEDECOD, AESEV, AESTDY, AESEQ, AOCCIFL)

# Most severe AE first occurrence per patient per body system
admiral_ae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCSIFL,
    by_vars = vars(USUBJID, AEBODSYS),
    order = vars(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first"
  ) %>%
  arrange(USUBJID, AESTDY, AESEQ) %>%
  select(USUBJID, AEBODSYS, AESEV, AESTDY, AOCCSIFL)
```

---

`derive_var_last_dose_amt` *Derive Last Dose Amount*

---

## Description

Add a variable for dose amount from the last dose to the input dataset.

## Usage

```
derive_var_last_dose_amt(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_var,
  dose_var = EXDOSE,
  traceability_vars = NULL
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset. The variables specified by the `by_vars` and `analysis_date` parameters are expected. |

dataset_ex — Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars parameter are expected.

filter_ex — Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL.

by_vars — Variables to join by (created by dplyr::vars).

dose_id — Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars().

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

## Details

The last dose amount is derived as the dose amount where the maximum dose_date is lower to or equal to the analysis_date per by_vars for each observation in dataset.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function create_single_dose_dataset() can be used to generate single doses from aggregate dose information and satisfy single_dose_condition.

## Value

Input dataset with additional column new_var.

## Author(s)

Annie Yang

## See Also

derive_vars_last_dose(), create_single_dose_dataset()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_flag(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

ex_single <- derive_vars_dtm(
  head(ex_single, 100),
  dtc = EXENDTC,
  new_vars_prefix = "EXEN",
  flag_imputation = "none"
)

adae <- admiral_ae %>%
  head(100) %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AST",
    highest_imputation = "M"
  )

adae %>%
  derive_var_last_dose_amt(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXENDTM),
    dose_date = EXENDTM,
    analysis_date = ASTDTM,
    new_var = LDOSE,
    dose_var = EXDOSE
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSE)

# or with traceability variables
adae %>%
  derive_var_last_dose_amt(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXENDTM),
    dose_date = EXENDTM,
    analysis_date = ASTDTM,
    new_var = LDOSE,
    dose_var = EXDOSE,
    traceability_vars = vars(
      LDOSEDOM = "EX",
      LDOSESEQ = EXSEQ,
      LDOSEVAR = "EXDOSE"
    )
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR, LDOSE)
```

derive_var_last_dose_date *Derive Last Dose Date-Time*

## Description

Add a variable for the dose date or datetime of the last dose to the input dataset.

## Usage

```
derive_var_last_dose_date(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_var,
  output_datetime = TRUE,
  traceability_vars = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset. The variables specified by the by_vars and analysis_date parameters are expected. |
| dataset_ex | Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars parameter are expected. |
| filter_ex | Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL. |
| by_vars | Variables to join by (created by dplyr::vars). |
| dose_id | Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars(). |
| dose_date | The EX dose date variable. 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 vars() listing the traceability variables, e.g. vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ). The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables.

## Details

The last dose date is derived as the maximum dose date where the dose_date is lower to or equal to the analysis_date per by_vars for each observation in dataset. When output_datetime is TRUE and time is missing, then the last dose date time is imputed to 00:00:00. However, if date is missing, then no imputation is done.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function create_single_dose_dataset() can be used to generate single doses from aggregate dose information and satisfy single_dose_condition.

## Value

Input dataset with additional column new_var.

## Author(s)

Ben Straub

## See Also

derive_vars_last_dose(), create_single_dose_dataset()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_flag(), derive_var_last_dose_amt(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_character(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

ex_single <- derive_vars_dtm(
  head(ex_single, 100),
  dtc = EXENDTC,
  new_vars_prefix = "EXEN",
  flag_imputation = "none"
)

adae <- admiral_ae %>%
```

```
  head(100) %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AST",
    highest_imputation = "M"
  )

adae %>%
  derive_var_last_dose_date(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXENDTM),
    dose_date = EXENDTM,
    analysis_date = ASTDTM,
    new_var = LDOSEDTM,
    traceability_vars = vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXDOSE")
  ) %>%
  select(STUDYID, USUBJID, AESEQ, AESTDTC, LDOSEDOM, LDOSESEQ, LDOSEVAR, LDOSEDTM)
```

---

`derive_var_last_dose_grp` *Derive Last Dose with User-Defined Groupings*

---

## Description

Add a variable for user-defined dose grouping of the last dose to the input dataset.

## Usage

```
derive_var_last_dose_grp(
  dataset,
  dataset_ex,
  filter_ex = NULL,
  by_vars = vars(STUDYID, USUBJID),
  dose_id = vars(),
  dose_date,
  analysis_date,
  single_dose_condition = (EXDOSFRQ == "ONCE"),
  new_var,
  grp_brks,
  grp_lbls,
  include_lowest = TRUE,
  right = TRUE,
  dose_var = EXDOSE,
  traceability_vars = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset. The variables specified by the by_vars and analysis_date parameters are expected. |
| dataset_ex | Input EX dataset. The variables specified by the by_vars, dose_date, new_vars parameters, and source variables from traceability_vars parameter are expected. |
| filter_ex | Filtering condition applied to EX dataset. For example, it can be used to filter for valid dose. Defaults to NULL. |
| by_vars | Variables to join by (created by dplyr::vars). |
| dose_id | Variables to identify unique dose (created by dplyr::vars). Defaults to empty vars(). |
| dose_date | The EX dose date variable. 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. |
| right | Logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. Refer to right parameter in cut() for details. |
| dose_var | The source dose amount variable. Defaults to EXDOSE. |
| traceability_vars | A named list returned by vars() listing the traceability variables, e.g. vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ). The left-hand side (names of the list elements) gives the names of the traceability variables in the returned dataset. The right-hand side (values of the list elements) gives the values of the traceability variables in the returned dataset. These can be either strings or symbols referring to existing variables. |

## Details

Last dose is the dose with maximum dose_date that is lower to or equal to the analysis_date per by_vars for each observation in dataset. The last dose group is then derived by user-defined grouping, which groups dose_var as specified in grp_brks, and returns grp_lbls as the values for new_var.

If dose information is aggregated (i.e. is a dosing frequency other than "ONCE" over a period defined by a start and end date) the function create_single_dose_dataset() can be used to generate single doses from aggregate dose information and satisfy single_dose_condition.

## Value

Input dataset with additional column new_var.

## Author(s)

Ben Straub

## See Also

derive_vars_last_dose(), cut(), create_single_dose_dataset()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_flag(), derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_merged_cat(), derive_var_merged_character(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

ex_single <- derive_vars_dtm(
  head(ex_single, 100),
  dtc = EXSTDTC,
  new_vars_prefix = "EXST",
  flag_imputation = "none"
)

adae <- admiral_ae %>%
  head(100) %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AST",
    highest_imputation = "M"
  )

adae %>%
  derive_var_last_dose_grp(
    dataset_ex = ex_single,
    filter_ex = (EXDOSE > 0 | (EXDOSE == 0 & grepl("PLACEBO", EXTRT))) &
      !is.na(EXSTDTM),
    by_vars = vars(STUDYID, USUBJID),
    dose_date = EXSTDTM,
    new_var = LDGRP,
    grp_brks = c(0, 20, 40, 60),
    grp_lbls = c("Low", "Medium", "High"),
    include_lowest = TRUE,
    right = TRUE,
    dose_var = EXDOSE,
```

```
  analysis_date = ASTDTM,
  traceability_vars = vars(LDOSEDOM = "EX", LDOSESEQ = EXSEQ, LDOSEVAR = "EXENDTC")
) %>%
select(USUBJID, LDGRP, LDOSEDOM, LDOSESEQ, LDOSEVAR)
```

---

`derive_var_merged_cat` *Merge a Categorization Variable*

---

## Description

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

## Usage

```
derive_var_merged_cat(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  source_var,
  cat_fun,
  filter_add = NULL,
  mode = NULL,
  missing_value = NA_character_
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter are expected. |
| `dataset_add` | Additional dataset<br>The variables specified by the `by_vars`, the `source_var`, and the `order` parameter are expected. |
| `by_vars` | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by `vars()` |
| `order` | Sort order<br>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.<br>*Default*: `NULL`<br>*Permitted Values*: list of variables or `desc(<variable>)` function calls created by `vars()`, e.g., `vars(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 parameter 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 parameter is not specified, all observations are considered.
*Default*: `NULL`
*Permitted Values*: a condition

`mode` — Selection mode
Determines if the first or last observation is selected. If the `order` parameter is specified, `mode` must be non-null.
If the `order` parameter is not specified, the `mode` parameter is ignored.
*Default*: `NULL`
*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`).

## Author(s)

Stefan Bundfuss

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_confirmation_flag()`, `derive_var_extreme_flag()`, `derive_var_last_dose_amt()`, `derive_var_last_dose_date()`, `derive_var_last_dose_grp()`, `derive_var_merged_character()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_worst_flag()`, `derive_vars_last_dose()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `derive_vars_transposed()`, `get_summary_records()`

## Examples

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

wgt_cat <- function(wgt) {
  case_when(
    wgt < 50 ~ "low",
    wgt > 90 ~ "high",
    TRUE ~ "normal"
  )
}

derive_var_merged_cat(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  order = vars(VSDTC, VSSEQ),
  filter_add = VSTESTCD == "WEIGHT" & substr(VISIT, 1, 9) == "SCREENING",
  new_var = WGTBLCAT,
  source_var = VSSTRESN,
  cat_fun = wgt_cat,
  mode = "last"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WGTBLCAT)

# defining a value for missing VS data
derive_var_merged_cat(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  order = vars(VSDTC, VSSEQ),
  filter_add = VSTESTCD == "WEIGHT" & substr(VISIT, 1, 9) == "SCREENING",
  new_var = WGTBLCAT,
  source_var = VSSTRESN,
  cat_fun = wgt_cat,
  mode = "last",
  missing_value = "MISSING"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WGTBLCAT)
```

---

# derive_var_merged_character

*Merge a Character Variable*

---

## Description

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

## Usage

```
derive_var_merged_character(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  source_var,
  case = NULL,
  filter_add = NULL,
  mode = NULL,
  missing_value = NA_character_
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the by_vars parameter are expected. |
| dataset_add | Additional dataset<br>The variables specified by the by_vars, the source_var, and the order parameter are expected. |
| by_vars | Grouping variables<br>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.<br>*Permitted Values*: list of variables created by vars() |
| order | Sort order<br>If the parameter is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.<br>*Default*: NULL<br>*Permitted Values*: list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) or NULL |
| new_var | New variable<br>The specified variable is added to the additional dataset and set to the transformed value with respect to the case parameter. |
| source_var | Source variable |
| case | Change case<br>Changes the case of the values of the new variable.<br>*Default*: NULL<br>*Permitted Values*: NULL, "lower", "upper", "title" |
| filter_add | Filter for additional dataset (dataset_add)<br>Only observations fulfilling the specified condition are taken into account for merging. If the parameter is not specified, all observations are considered. |

*Default*: NULL

*Permitted Values*: a condition

mode — Selection mode

Determines if the first or last observation is selected. If the order parameter is specified, mode must be non-null.

If the order parameter is not specified, the mode parameter is ignored.

*Default*: NULL

*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).

## Author(s)

Stefan Bundfuss

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_flag(), derive_var_last_dose_amt(), derive_var_last_dose_date(), derive_var_last_dose_grp(), derive_var_merged_cat(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

derive_var_merged_character(
```

```
  admiral_dm,
  dataset_add = admiral_ds,
  by_vars = vars(STUDYID, USUBJID),
  new_var = DISPSTAT,
  filter_add = DSCAT == "DISPOSITION EVENT",
  source_var = DSDECOD,
  case = "title"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, DISPSTAT)
```

---

`derive_var_merged_exist_flag` *Merge an Existence Flag*

---

## Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in another dataset fulfilling a certain condition.

## Usage

```
derive_var_merged_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  new_var,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL
)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `by_vars` parameter are expected. |
| `dataset_add` | Additional dataset<br>The variables specified by the `by_vars` parameter are expected. |
| `by_vars` | Grouping variables<br>*Permitted Values*: list of variables |
| `new_var` | New variable<br>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 observations 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 parameter is not specified, all observations are considered.

*Permitted Values*: a condition

## Details

1. The additional dataset is restricted to the observations matching the `filter_add` condition.
2. The new variable is added to the input dataset and set to the true value (`true_value`) if for the by group at least one observation exists in the (restricted) additional dataset where the condition evaluates to `TRUE`. It is set to the false value (`false_value`) if for the by group at least one observation exists and for all observations the condition evaluates to `FALSE` or `NA`. Otherwise, it is set to the missing value (`missing_value`).

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_confirmation_flag()`, `derive_var_extreme_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_var_obs_number()`, `derive_var_worst_flag()`, `derive_vars_last_dose()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `derive_vars_transposed()`, `get_summary_records()`

## Examples

```
library(admiral.test)
library(dplyr, warn.conflicts = FALSE)
data("admiral_dm")
data("admiral_ae")
derive_var_merged_exist_flag(
  admiral_dm,
  dataset_add = admiral_ae,
  by_vars = vars(STUDYID, USUBJID),
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, AERELFL)

data("admiral_vs")
derive_var_merged_exist_flag(
  admiral_dm,
  dataset_add = admiral_vs,
  by_vars = vars(STUDYID, USUBJID),
  filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  false_value = "N",
  missing_value = "M"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WTBLHIFL)
```

---

# derive_var_obs_number *Adds a Variable Numbering the Observations Within Each By Group*

---

## Description

Adds a variable numbering the observations within each by group

## Usage

```
derive_var_obs_number(
  dataset,
  by_vars = NULL,
  order = NULL,
  new_var = ASEQ,
  check_type = "none"
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified by the order and the by_vars parameter are expected. |

| | |
|---|---|
| by_vars | Grouping variables<br>Permitted Values: list of variables |
| order | Sort order<br>Within each by group the observations are ordered by the specified order.<br>Permitted Values: list of variables or functions of variables |
| new_var | Name of variable to create<br>The new variable is set to the observation number for each by group. The numbering starts with 1.<br>Default: ASEQ |
| check_type | Check uniqueness?<br>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.<br>Default: "none"<br>Permitted Values: "none", "warning", "error" |

## Details

For each group (with respect to the variables specified for the by_vars parameter) the first or last observation (with respect to the order specified for the order parameter and the mode specified for the mode parameter) is included in the output dataset.

## Value

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

## Author(s)

Stefan Bundfuss

## See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_worst_flag(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

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

admiral_vs %>%
  select(USUBJID, VSTESTCD, VISITNUM, VSTPTNUM) %>%
  filter(VSTESTCD %in% c("HEIGHT", "WEIGHT")) %>%
```

```
  derive_var_obs_number(
    by_vars = vars(USUBJID, VSTESTCD),
    order = vars(VISITNUM, VSTPTNUM)
  )
```

---

derive_var_ontrtfl *Derive On-Treatment Flag Variable*

---

## Description

Derive on-treatment flag (ONTRTFL) in an ADaM dataset with a single assessment date (e.g ADT) or event start and end dates (e.g. ASTDT/AENDT).

## Usage

```
derive_var_ontrtfl(
  dataset,
  new_var = ONTRTFL,
  start_date,
  end_date = NULL,
  ref_start_date,
  ref_end_date = NULL,
  ref_end_window = 0,
  filter_pre_timepoint = NULL,
  span_period = NULL
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset.<br>Required columns are start_date, end_date, ref_start_date and ref_end_date. |
| new_var | On-treatment flag variable name to be created.<br>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.<br>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.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.<br>Optional; Default is null. If the used and date value is missing on an observation, it is assumed the medication is ongoing and ONTRTFL is set to "Y". |

`ref_start_date` The lower bound of the on-treatment period Required; A date or date-time object column is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

`ref_end_date` The upper bound of the on-treatment period A date or date-time object column is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Optional; This can be null and everything after `ref_start_date` will be considered on-treatment. Default is `NULL`.

`ref_end_window` A window to add to the upper bound `ref_end_date` measured in days (e.g. 7 if 7 days should be added to the upper bound) Optional; default is 0.

`filter_pre_timepoint`

An expression to filter observations as not on-treatment when `date = ref_start_date`. For example, if observations where `VSTPT = PRE` should not be considered on-treatment when `date = ref_start_date`, `filter_pre_timepoint` should be used to denote when the on-treatment flag should be set to null. Optional; default is `NULL`.

`span_period` A `"Y"` scalar character. If `"Y"`, events that started prior to the `ref_start_date`and are ongoing or end after the `ref_start_date` are flagged as `"Y"`. Optional; default is `NULL`.

## Details

On-Treatment is calculated by determining whether the assessment date or start/stop dates fall between 2 dates. The following logic is used to assign on-treatment = `"Y"`:

1. `start_date` is missing and `ref_start_date`is non-missing
2. No timepoint filter is provided (`filter_pre_timepoint`) and both `start_date` and `ref_start_date` are non-missing and `start_date = ref_start_date`
3. A timepoint is provided (`filter_pre_timepoint`) and both `start_date` and `ref_start_date` are non-missing and `start_date = ref_start_date` and the filter provided in `filter_pre_timepoint` is not true.
4. `ref_end_date` is not provided and `ref_start_date < start_date`
5. `ref_end_date` is provided and `ref_start_date < start_date <= ref_end_date + ref_end_window`.

If the `end_date` is provided and the `end_date` < ref_start_date then the `ONTRTFL` is set to `NULL`.This would be applicable to cases where the `start_date` is missing and `ONTRTFL` has been assigned as `"Y"` above.

If the `span_period` is specified as `"Y"`, this allows the user to assign `ONTRTFL` as `"Y"` to cases where the record started prior to the `ref_start_date` and was ongoing or ended after the `ref_start_date`.

Any date imputations needed should be done prior to calling this function.

## Value

The input dataset with an additional column named `ONTRTFL` with a value of `"Y"` or `NA`

## Author(s)

Alice Ehmann, Teckla Akinyi

## 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_basetype(), derive_var_base(), derive_var_chg(), derive_var_pchg(), derive_var_shift()

## Examples

```
library(tibble)
library(dplyr)
library(lubridate, warn.conflict = 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"),
) %>%
  mutate(TPT = c(NA, "PRE", NA))
derive_var_ontrtfl(
  advs,
  start_date = ADTM,
```

```
  ref_start_date = TRTSDTM,
  ref_end_date = TRTEDTM,
  filter_pre_timepoint = TPT == "PRE"
)

advs <- tribble(
  ~USUBJID, ~ASTDT,            ~TRTSDT,           ~TRTEDT,           ~AENDT,
  "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
  "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
  "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
)
derive_var_ontrtfl(
  advs,
  start_date = ASTDT,
  end_date = AENDT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  ref_end_window = 60,
  span_period = "Y"
)

advs <- tribble(
  ~USUBJID, ~ASTDT,            ~AP01SDT,          ~AP01EDT,          ~AENDT,
  "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
  "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
  "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
)
derive_var_ontrtfl(
  advs,
  new_var = ONTR01FL,
  start_date = ASTDT,
  end_date = AENDT,
  ref_start_date = AP01SDT,
  ref_end_date = AP01EDT,
  span_period = "Y"
)
```

---

`derive_var_pchg` *Derive Percent Change from Baseline*

---

## Description

Derive percent change from baseline (`PCHG`) in a BDS dataset

## Usage

```
derive_var_pchg(dataset)
```

## Arguments

| | |
|---|---|
| `dataset` | The input dataset. Required variables are `AVAL` and `BASE`. |

## Details

Percent change from baseline is calculated by dividing change from baseline by the absolute value of the baseline value and multiplying the result by `100`.

## Value

The input dataset with an additional column named `PCHG`

## Author(s)

Thomas Neitmann

## See Also

`derive_var_chg()`

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_basetype()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_ontrtfl()`, `derive_var_shift()`

## Examples

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

---

# derive_var_shift *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

```
derive_var_shift(
  dataset,
  new_var,
  from_var,
  to_var,
  na_val = "NULL",
  sep_val = " to "
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The columns specified by from_var and the to_var parameters are expected. |
| new_var | Name of the character shift variable to create. |
| from_var | Variable containing value to shift from. |
| to_var | Variable containing value to shift to. |
| na_val | Character string to replace missing values in from_var or to_var.<br>Default: "NULL" |
| sep_val | Character string to concatenate values of from_var and to_var.<br>Default: " to " |

## Details

new_var is derived by concatenating the values of from_var to values of to_var (e.g. "NORMAL to HIGH"). When from_var or to_var has missing value, the missing value is replaced by na_val (e.g. "NORMAL to NULL").

## Value

The input dataset with the character shift variable added

## Author(s)

Annie Yang

## 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_basetype(), derive_var_base(), derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg()

## Examples

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

data <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BNRIND, ~ANRIND,
  "P01", "ALB", 33, "Y", "LOW", "LOW",
  "P01", "ALB", 38, NA, "LOW", "NORMAL",
  "P01", "ALB", NA, NA, "LOW", NA,
  "P02", "ALB", 37, "Y", "NORMAL", "NORMAL",
  "P02", "ALB", 49, NA, "NORMAL", "HIGH",
  "P02", "SODIUM", 147, "Y", "HIGH", "HIGH"
)

data %>%
  convert_blanks_to_na() %>%
  derive_var_shift(
```

```
    new_var = SHIFT1,
    from_var = BNRIND,
    to_var = ANRIND
  )

# or only populate post-baseline records
data %>%
  convert_blanks_to_na() %>%
  restrict_derivation(
    derivation = derive_var_shift,
    args = params(
      new_var = SHIFT1,
      from_var = BNRIND,
      to_var = ANRIND
    ),
    filter = is.na(ABLFL)
  )
```

---

`derive_var_trtdurd` *Derive Total Treatment Duration (Days)*

---

## Description

Derives total treatment duration (days) (`TRTDURD`)

## Usage

```
derive_var_trtdurd(dataset, start_date = TRTSDT, end_date = TRTEDT)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The columns specified by the `start_date` and the `end_date` parameter are expected. |
| `start_date` | The start date<br>A date or date-time object is expected.<br>Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.<br>Default: `TRTSDT` |
| `end_date` | The end date<br>A date or date-time object is expected.<br>Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.<br>Default: `TRTEDT` |

## Details

The total treatment duration is derived as the number of days from start to end date plus one.

## Value

The input dataset with TRTDURD added

## Author(s)

Stefan Bundfuss

## See Also

derive_vars_duration()

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

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

derive_var_trtdurd(data)
```

---

# derive_var_worst_flag *Adds a Variable Flagging the Maximal / Minimal Value Within a Group of Observations*

---

## Description

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

## Usage

```
derive_var_worst_flag(
  dataset,
  by_vars,
  order,
  new_var,
  param_var,
  analysis_var,
  worst_high,
  worst_low,
  filter = deprecated(),
  check_type = "warning"
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset. Variables specified by by_vars, order, param_var, and analysis_var are expected. |
| by_vars | Grouping variables<br>Permitted Values: list of variables |
| order | Sort order. Used to determine maximal / minimal observation if they are not unique, see Details section for more information. |
| new_var | Variable to add to the dataset. It is set "Y" for the maximal / minimal observation of each group, see Details section for more information. |
| param_var | Variable with the parameter values for which the maximal / minimal value is calculated. |
| analysis_var | Variable with the measurement values for which the maximal / minimal value is calculated. |
| worst_high | Character with param_var values specifying the parameters referring to "high". Use character(0) if not required. |
| worst_low | Character with param_var values specifying the parameters referring to "low". Use character(0) if not required. |
| filter | Deprecated, please use restrict_derivation() instead (see examples). |
| check_type | Check uniqueness?<br>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.<br>Default: "warning"<br>Permitted Values: "none", "warning", "error" |

## Details

For each group with respect to the variables specified by the by_vars parameter, the maximal / minimal observation of analysis_var is labelled in the new_var column as "Y", if its param_var is in worst_high / worst_low. Otherwise, it is assigned NA. If there is more than one such maximal / minimal observation, the first one with respect to the order specified by the order parameter is flagged. The direction of "worst" depends on the definition of worst for a specified parameters in the arguments worst_high / worst_low, i.e. for some parameters the highest value is the worst and for others the worst is the lowest value.

## Value

The input dataset with the new flag variable added.

## Author(s)

Ondrej Slama

## See Also

derive_var_extreme_flag()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_confirmation_flag(), derive_var_extreme_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_var_merged_exist_flag(), derive_var_obs_number(), derive_vars_last_dose(), derive_vars_merged_lookup(), derive_vars_merged(), derive_vars_transposed(), get_summary_records()

## Examples

```
input <- tibble::tribble(
  ~STUDYID, ~USUBJID,  ~PARAMCD,     ~AVISIT,                  ~ADT, ~AVAL,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-27"),  15.0,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-25"),  14.0,
  "TEST01",  "PAT01", "PARAM01",  "BASELINE", as.Date("2021-04-23"),  15.0,
  "TEST01",  "PAT01", "PARAM01",    "WEEK 1", as.Date("2021-04-27"),  10.0,
  "TEST01",  "PAT01", "PARAM01",    "WEEK 2", as.Date("2021-04-30"),  12.0,
  "TEST01",  "PAT02", "PARAM01", "SCREENING", as.Date("2021-04-27"),  15.0,
  "TEST01",  "PAT02", "PARAM01",  "BASELINE", as.Date("2021-04-25"),  14.0,
  "TEST01",  "PAT02", "PARAM01",  "BASELINE", as.Date("2021-04-23"),  15.0,
  "TEST01",  "PAT02", "PARAM01",    "WEEK 1", as.Date("2021-04-27"),  10.0,
  "TEST01",  "PAT02", "PARAM01",    "WEEK 2", as.Date("2021-04-30"),  12.0,
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-27"),  15.0,
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-25"),  14.0,
  "TEST01",  "PAT01", "PARAM02", "SCREENING", as.Date("2021-04-23"),  15.0,
  "TEST01",  "PAT01", "PARAM02",  "BASELINE", as.Date("2021-04-27"),  10.0,
  "TEST01",  "PAT01", "PARAM02",    "WEEK 2", as.Date("2021-04-30"),  12.0,
  "TEST01",  "PAT02", "PARAM02", "SCREENING", as.Date("2021-04-27"),  15.0,
  "TEST01",  "PAT02", "PARAM02",  "BASELINE", as.Date("2021-04-25"),  14.0,
  "TEST01",  "PAT02", "PARAM02",    "WEEK 1", as.Date("2021-04-23"),  15.0,
  "TEST01",  "PAT02", "PARAM02",    "WEEK 1", as.Date("2021-04-27"),  10.0,
  "TEST01",  "PAT02", "PARAM02",  "BASELINE", as.Date("2021-04-30"),  12.0,
  "TEST01",  "PAT02", "PARAM03", "SCREENING", as.Date("2021-04-27"),  15.0,
  "TEST01",  "PAT02", "PARAM03",  "BASELINE", as.Date("2021-04-25"),  14.0,
  "TEST01",  "PAT02", "PARAM03",    "WEEK 1", as.Date("2021-04-23"),  15.0,
  "TEST01",  "PAT02", "PARAM03",    "WEEK 1", as.Date("2021-04-27"),  10.0,
  "TEST01",  "PAT02", "PARAM03",  "BASELINE", as.Date("2021-04-30"),  12.0
)

derive_var_worst_flag(
  input,
  by_vars = vars(USUBJID, PARAMCD, AVISIT),
  order = vars(desc(ADT)),
  new_var = WORSTFL,
  param_var = PARAMCD,
  analysis_var = AVAL,
  worst_high = c("PARAM01", "PARAM03"),
  worst_low = "PARAM02"
)
## Not run:
```

```
# example with ADVS
restrict_derivation(
  advs,
  derivation = derive_var_worst_flag,
  args = params(
    by_vars = vars(USUBJID, PARAMCD, AVISIT),
    order = vars(ADT, ATPTN),
    new_var = WORSTFL,
    param_var = PARAMCD,
    analysis_var = AVAL,
    worst_high = c("SYSBP", "DIABP"),
    worst_low = "RESP"
  ),
  filter = !is.na(AVISIT) & !is.na(AVAL)
)

## End(Not run)
```

---

`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 forstart_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()

---

dtm_level *Create a* dtm_level *object*

---

## Description

Create a dtm_level object

## Usage

```
dtm_level(level)
```

## Arguments

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

## Details

A dtm_level object is an ordered factor, i.e., two objects can be compared.

## Value

A dtm_level object

## Author(s)

Stefan Bundfuss

## See Also

Utilities used for date imputation: dt_level(), get_imputation_target_date(), get_imputation_target_time(), get_partialdatetime(), restrict_imputed_dtc_dtm(), restrict_imputed_dtc_dt()

## dt_level *Create a* dt_level *object*

### Description

Create a dt_level object

### Usage

```
dt_level(level)
```

### Arguments

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

### Details

A dt_level object is an ordered factor, i.e., two objects can be compared.

### Value

A dt_level object

### Author(s)

Stefan Bundfuss

### See Also

Utilities used for date imputation: dtm_level(), get_imputation_target_date(), get_imputation_target_time(), get_partialdatetime(), restrict_imputed_dtc_dtm(), restrict_imputed_dtc_dt()

## event_source *Create an* event_source *Object*

### Description

event_source objects are used to define events as input for the derive_param_tte() function.

### Usage

```
event_source(dataset_name, filter = NULL, date, set_values_to = NULL)
```

## Arguments

| | |
|---|---|
| dataset_name | The name of the source dataset<br>The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). |
| filter | An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. |
| date | A variable providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol is expected.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. |
| set_values_to | A named list returned by vars() defining the variables to be set for the event or censoring, e.g. vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, or NA. |

## Value

An object of class event_source, inheriting from class tte_source

## Author(s)

Stefan Bundfuss

## See Also

derive_param_tte(), censor_source()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

```
# Death event
event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = vars(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)
```

`extend_source_datasets` *Add By Groups to All Datasets if Necessary*

## Description

The function ensures that the by variables are contained in all source datasets.

## Usage

```
extend_source_datasets(source_datasets, by_vars)
```

## Arguments

| | |
|---|---|
| `source_datasets` | Source datasets<br>A named list of datasets is expected. Each dataset must contain either all by variables or none of the by variables. |
| `by_vars` | By variables |

## Details

1. The by groups are determined as the union of the by groups occurring in the source datasets.
2. For all source datasets which do not contain the by variables the source dataset is replaced by the cartesian product of the source dataset and the by groups.

## Value

The list of extended source datasets

## Author(s)

Stefan Bundfuss

## See Also

Source Specifications: `assert_db_requirements()`, `assert_terms()`, `assert_valid_queries()`, `censor_source()`, `date_source()`, `death_event`, `derive_var_dthcaus()`, `event_source()`, `filter_date_sources()`, `format.sdg_select()`, `format.smq_select()`, `list_tte_source_objects()`, `params()`, `query()`, `sdg_select()`, `smq_select()`, `tte_source()`, `validate_query()`, `validate_sdg_select()`, `validate_smq_select()`

## Examples

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

adsl <- tibble::tribble(
  ~USUBJID, ~TRTSDT,           ~EOSDT,
  "01",     ymd("2020-12-06"), ymd("2021-03-06"),
  "02",     ymd("2021-01-16"), ymd("2021-02-03")
) %>%
  mutate(STUDYID = "AB42")

ae <- tibble::tribble(
  ~USUBJID, ~AESTDTC,            ~AESEQ, ~AEDECOD,
  "01",     "2021-01-03T10:56", 1,      "Flu",
  "01",     "2021-03-04",       2,      "Cough",
  "01",     "2021",             3,      "Flu"
) %>%
  mutate(STUDYID = "AB42")

extend_source_datasets(
  source_datasets = list(adsl = adsl, ae = ae),
  by_vars = vars(AEDECOD)
)
```

---

# extract_duplicate_records

*Extract Duplicate Records*

---

## Description

Extract Duplicate Records

## Usage

```
extract_duplicate_records(dataset, by_vars)
```

## Arguments

| | |
|---|---|
| dataset | A data frame |
| by_vars | A list of variables created using `vars()` identifying groups of records in which to look for duplicates |

## Value

A `data.frame` of duplicate records within `dataset`

## Author(s)

Thomas Neitmann

## See Also

Utilities for Dataset Checking: get_duplicates_dataset(), get_many_to_one_dataset(), get_one_to_many_dataset(

## Examples

```
data(admiral_adsl)

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

extract_duplicate_records(adsl, vars(USUBJID))
```

---

# extract_unit *Extract Unit From Parameter Description*

---

## Description

Extract the unit of a parameter from a description like "Param (unit)".

## 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)

### See Also

Other datasets: admiral_adsl, atoxgr_criteria_ctcv4, queries_mh, queries

## filter_confirmation *Filter Confirmed Observations*

### 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

```
filter_confirmation(
  dataset,
  by_vars,
  join_vars,
  join_type,
  first_cond = NULL,
  order,
  filter,
  check_type = "warning"
)
```

## Arguments

| | |
|---|---|
| dataset | Input dataset<br>The variables specified for by_vars, join_vars, and order are expected. |
| by_vars | By variables<br>The specified variables are used as by variables for joining the input dataset with itself. |
| join_vars | Variables to keep from joined dataset<br>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 = vars(AVALC, AVISITN) and filter = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join could be specified.<br>The *.join variables are not included in the output dataset. |
| join_type | Observations to keep after joining<br>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.<br>*Permitted Values:* "before", "after", "all" |
| first_cond | Condition for selecting range of data<br>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<br>The observations are ordered by the specified order. |
| filter | Condition for selecting observations<br>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 = vars(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>        <dbl> <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 = vars(AVISITN)‘ is restricted to

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <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 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.

## Author(s)

Stefan Bundfuss

## See Also

`count_vals()`, `min_cond()`, `max_cond()`

Utilities for Filtering Observations: `count_vals()`, `filter_extreme()`, `filter_relative()`, `max_cond()`, `min_cond()`

## Examples

```
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_confirmation(
  adae,
  by_vars = vars(USUBJID),
  join_vars = vars(ACOVFL, ADY),
  join_type = "all",
  order = vars(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_confirmation(
  data,
  by_vars = vars(USUBJID),
  join_vars = vars(AVALC, AVISITN),
  join_type = "after",
  order = vars(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_confirmation(
  data,
  by_vars = vars(USUBJID),
  join_vars = vars(AVALC),
  join_type = "after",
  order = vars(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_confirmation(
  data,
  by_vars = vars(USUBJID),
  join_vars = vars(AVALC, ADY),
  join_type = "after",
  order = vars(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
    )
)
```

---

`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<br>A list of tte_source() objects is expected. |
| source_datasets | Source datasets<br>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<br>If the parameter is specified, for each by group the observations are selected separately. |
| create_datetime | Create datetime variable?<br>If set to TRUE, variables ADTM is created. Otherwise, variables ADT is created. |
| subject_keys | Variables to uniquely identify a subject<br>A list of symbols created using vars() is expected. |
| mode | Selection mode (first or last)<br>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.<br>Permitted Values: "first", "last" |

## Details

The following steps are performed to create the output dataset:

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

## Value

A dataset with one observation per subject as described in the "Details" section.

## Author(s)

Stefan Bundfuss

## See Also

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

```
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",     "2021-01-03", 1,      "Flu",
  "01",     "2021-03-04", 2,      "Cough",
  "01",     "2021-01-01", 3,      "Flu"
) %>%
  mutate(
    STUDYID = "AB42",
    AESTDT = ymd(AESTDTC)
  )

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

filter_date_sources(
```

```
  sources = list(ttae),
  source_datasets = list(adsl = adsl, ae = ae),
  by_vars = vars(AEDECOD),
  create_datetime = FALSE,
  subject_keys = vars(STUDYID, USUBJID),
  mode = "first"
)
```

---

`filter_extreme` *Filter the First or Last Observation for Each By Group*

---

## Description

Filters the first or last observation for each by group.

## Usage

```
filter_extreme(dataset, by_vars = NULL, order, mode, check_type = "warning")
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset<br>The variables specified by the `order` and the `by_vars` parameter are expected. |
| `by_vars` | Grouping variables<br>*Default*: `NULL`<br>*Permitted Values:* list of variables created by `vars()` |
| `order` | Sort order<br>Within each by group the observations are ordered by the specified order.<br>*Permitted Values:* list of variables or `desc(<variable>)` function calls created by `vars()`, e.g., `vars(ADT, desc(AVAL))` |
| `mode` | Selection mode (first or last)<br>If `"first"` is specified, the first observation of each by group is included in the output dataset. If `"last"` is specified, the last observation of each by group is included in the output dataset.<br>*Permitted Values:* `"first"`, `"last"` |
| `check_type` | Check uniqueness?<br>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.<br>*Default:* `"warning"`<br>*Permitted Values:* `"none"`, `"warning"`, `"error"` |

## Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is included in the output dataset.

## Value

A dataset containing the first or last observation of each by group

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Filtering Observations: count_vals(), filter_confirmation(), filter_relative(), max_cond(), min_cond()

## Examples

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

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

# Select highest dose for each patient on the active drug
admiral_ex %>%
  filter(EXTRT != "PLACEBO") %>%
  filter_extreme(
    by_vars = vars(USUBJID),
    order = vars(EXDOSE),
    mode = "last",
    check_type = "none"
  ) %>%
  select(USUBJID, EXTRT, EXDOSE)
```

---

# filter_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<br>The variables specified by the order and the by_vars parameter are expected. |
| by_vars | Grouping variables<br>*Permitted Values:* list of variables created by vars() |
| order | Sort order<br>Within each by group the observations are ordered by the specified order.<br>*Permitted Values:* list of variables or desc(<variable>) function calls created by vars(), e.g., vars(ADT, desc(AVAL)) |
| condition | Condition for Reference Observation<br>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)<br>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.<br>*Permitted Values:* "first", "last" |
| selection | Select observations before or after the reference observation?<br>*Permitted Values:* "before", "after" |
| inclusive | Include the reference observation?<br>*Permitted Values:* TRUE, FALSE |
| keep_no_ref_groups | Should by groups without reference observation be kept?<br>*Default:* TRUE<br>*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:* "none"

*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) if fulfilled the first or last time (order parameter and mode parameter) is included in the output dataset.

## Value

A dataset containing for each by group the observations before or after the observation where the condition was fulfilled the first or last time

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Filtering Observations: count_vals(), filter_confirmation(), filter_extreme(), max_cond(), min_cond()

## Examples

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

response <- tibble::tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "CR",
  "1",      4,        "SD",
  "1",      5,        "NE",
  "2",      1,        "SD",
  "2",      2,        "PD",
  "2",      3,        "PD",
  "3",      1,        "SD",
  "4",      1,        "SD",
  "4",      2,        "PR",
  "4",      3,        "PD",
  "4",      4,        "SD",
  "4",      5,        "PR"
)

# Select observations up to first PD for each patient
response %>%
```

```
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Select observations after last CR, PR, or SD for each patient
response %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC %in% c("CR", "PR", "SD"),
    mode = "last",
    selection = "after",
    inclusive = FALSE
  )

# Select observations from first response to first PD
response %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC %in% c("CR", "PR"),
    mode = "first",
    selection = "after",
    inclusive = TRUE,
    keep_no_ref_groups = FALSE
  ) %>%
  filter_relative(
    by_vars = vars(USUBJID),
    order = vars(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )
```

---

`format.sdg_select` *Returns a Character Representation of a* `sdg_select()` *Object*

---

## Description

The function returns a character representation of a `sdg_select()` object. It can be used for error messages for example.

### Usage

```
## S3 method for class 'sdg_select'
format(x, ...)
```

### Arguments

| | |
|---|---|
| x | A sdg_select() object |
| ... | Not used |

### Value

A character representation of the sdg_select() object

### Author(s)

Stefan Bundfuss

### See Also

sdg_select()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

### Examples

```
format(
  sdg_select(
    name = "5-aminosalicylates for ulcerative colitis"
  )
)
```

---

## format.smq_select *Returns a Character Representation of a* smq_select() *Object*

---

### Description

The function returns a character representation of a smq_select() object. It can be used for error messages for example.

### Usage

```
## S3 method for class 'smq_select'
format(x, ...)
```

### Arguments

| | |
|---|---|
| x | A smq_select() object |
| ... | Not used |

### Value

A character representation of the smq_select() object

### Author(s)

Stefan Bundfuss

### See Also

smq_select()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

### Examples

```
format(smq_select(id = 42, scope = "NARROW"))
```

---

## format_eoxxstt_default

*Default Format for Disposition Status*

---

### Description

Define a function to map the disposition status. To be used as an input for derive_var_disposition_status().

### Usage

```
format_eoxxstt_default(status)
```

### Arguments

| | |
|---|---|
| status | the disposition variable used for the mapping (e.g. DSDECOD). |

### Details

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

## Value

A character vector derived based on the values given in status: "NOT STARTED" if status is "SCREEN FAILURE" or "SCREENING NOT COMPLETED", "COMPLETED" if status is "COMPLETED", "DISCONTINUED" if status is not in ("COMPLETED","SCREEN FAILURE", "SCREENING NOT COMPLETED") nor NA, "ONGOING" otherwise.

## Author(s)

Samia Kabi

## See Also

derive_var_disposition_status()

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

## Examples

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

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

---

# format_reason_default *Default Format for the Disposition Reason*

---

## Description

Define a function to map the disposition reason, to be used as a parameter in derive_vars_disposition_reason().

## Usage

```
format_reason_default(reason, reason_spe = NULL)
```

## Arguments

| | |
|---|---|
| reason | the disposition variable used for the mapping (e.g. DSDECOD). |
| reason_spe | the disposition variable used for the mapping of the details if required (e.g. DSTERM). |

### Details

format_reason_default(DSDECOD) returns DSDECOD when DSDECOD is not 'COMPLETED' nor NA. format_reason_default(DSDECOD, DSTERM) returns DSTERM when DSDECOD is equal to 'OTHER'. Usually this function can not be used with %>%.

### Value

A character vector

### Author(s)

Samia Kabi

### See Also

derive_vars_disposition_reason()

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

### Examples

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

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

---

## get_duplicates_dataset

*Get Duplicate Records that Lead to a Prior Error*

---

### Description

Get Duplicate Records that Lead to a Prior Error

### Usage

```
get_duplicates_dataset()
```

## Details

Many admiral function check that the input dataset contains only one record per by_vars group and throw an error otherwise. The get_duplicates_dataset() function allows one to retrieve the duplicate records that lead to an error.

Note that the function always returns the dataset of duplicates from the last error that has been thrown in the current R session. Thus, after restarting the R sessions get_duplicates_dataset() will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

## Value

A data.frame or NULL

## Author(s)

Thomas Neitmann

## See Also

Utilities for Dataset Checking: extract_duplicate_records(), get_many_to_one_dataset(), get_one_to_many_dataset()

## Examples

```
data(admiral_adsl)

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

signal_duplicate_records(adsl, vars(USUBJID), cnd_type = "warning")

get_duplicates_dataset()
```

---

# get_imputation_target_date

*Get Date Imputation Targets*

---

## Description

Get Date Imputation Targets

## Usage

```
get_imputation_target_date(date_imputation, month)
```

### Arguments

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

`month`
: Month component of the partial date

### Details

- For `date_imputation` = `"first"` `"0000"`, `"01"`, `"01"` are returned.
- For `date_imputation` = `"mid"` `"xxxx"`, `"06"`, `"30"` if `month` is `NA` and `"15"` otherwise are returned.
- For `date_imputation` = `"last"` `"9999"`, `"12"`, `"31"` are returned.
- For `date_imputation` = `"<mm>-<dd>"` `"xxxx"`, `"<mm>"`, `"<dd>"` are returned.

`"xxxx"` indicates that the component is undefined. If an undefined component occurs in the imputed DTC value, the imputed DTC value is set to `NA_character_` in the imputation functions.

### Value

A list of character vectors. The elements of the list are named "year", "month", "day".

### Author(s)

Stefan Bundfuss

### See Also

`impute_dtc_dtm()`, `impute_dtc_dt()`

Utilities used for date imputation: `dt_level()`, `dtm_level()`, `get_imputation_target_time()`, `get_partialdatetime()`, `restrict_imputed_dtc_dtm()`, `restrict_imputed_dtc_dt()`

---

## get_imputation_target_time *Get Time Imputation Targets*

---

### Description

Get Time Imputation Targets

### Usage

```
get_imputation_target_time(time_imputation)
```

### Arguments

`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.

### Details

- For `time_imputation` = `"first"` `"00"`, `"00"`, `"00"` are returned.
- For `time_imputation` = `"last"` `"23"`, `"59"`, `"59"` are returned.
- For `time_imputation` = `"<hh>:<mm>:<ss>"` `"<hh>"`, `"<mm>"`, `"<ss>"` are returned.

### Value

A list of character vectors. The elements of the list are named "hour", "minute", "second".

### Author(s)

Stefan Bundfuss

### See Also

`impute_dtc_dtm()`

Utilities used for date imputation: `dt_level()`, `dtm_level()`, `get_imputation_target_date()`, `get_partialdatetime()`, `restrict_imputed_dtc_dtm()`, `restrict_imputed_dtc_dt()`

---

## `get_many_to_one_dataset` *Get Many to One Values that Led to a Prior Error*

---

### Description

Get Many to One Values that Led to a Prior Error

### Usage

```
get_many_to_one_dataset()
```

### Details

If `assert_one_to_one()` detects an issue, the many to one values are stored in a dataset. This dataset can be retrieved by `get_many_to_one_dataset()`.

Note that the function always returns the many to one values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_many_to_one_dataset()` will return `NULL` and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

## Value

A data.frame or NULL

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Dataset Checking: extract_duplicate_records(), get_duplicates_dataset(), get_one_to_many_dataset()

## Examples

```
data(admiral_adsl)

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

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`

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Dataset Checking: `extract_duplicate_records()`, `get_duplicates_dataset()`, `get_many_to_one_dataset()`

## Examples

```
data(admiral_adsl)

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

get_one_to_many_dataset()
```

`get_partialdatetime` *Parse DTC variable and Determine Components*

## Description

Parse DTC variable and Determine Components

## Usage

```
get_partialdatetime(dtc)
```

## Arguments

| | |
|---|---|
| `dtc` | The `'--DTC'` date to parse<br>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 value for any component. |

## Details

The function can be replaced by the parttime parser once it is available.

## Value

A list of character vectors. The elements of the list are named "year", "month", "day", "hour", "minute", and "second". Missing components are set to `NA_character_`.

## Author(s)

Stefan Bundfuss

## See Also

`impute_dtc_dtm()`, `impute_dtc_dt()`

Utilities used for date imputation: `dt_level()`, `dtm_level()`, `get_imputation_target_date()`, `get_imputation_target_time()`, `restrict_imputed_dtc_dtm()`, `restrict_imputed_dtc_dt()`

`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

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

## Arguments

| | |
|---|---|
| dataset | A data frame. |
| by_vars | Variables to consider for generation of groupwise summary records. Providing the names of variables in vars() will create a groupwise summary and generate summary records for the specified groups. |
| filter | Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved.<br>For example,<br>• filter_rows = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars.<br>• 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.<br>Set a list of variables to some specified value for the new observation(s)<br>• LHS refer to a variable.<br>• RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or NA. (e.g. vars(PARAMCD = "TDOSE",PARCAT1 = "OVERALL")). More general expression are not allowed. |

## Details

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.

## Author(s)

Pavan Kumar, updated by Alana Harris

## See Also

`derive_summary_records()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_confirmation_flag()`, `derive_var_extreme_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_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_worst_flag()`, `derive_vars_last_dose()`, `derive_vars_merged_lookup()`, `derive_vars_merged()`, `derive_vars_transposed()`

## Examples

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

# Summarize the average of the triplicate ECG interval values (AVAL)
get_summary_records(
  adeg,
  by_vars = vars(USUBJID, PARAM, AVISIT),
  analysis_var = AVAL,
```

```
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = vars(DTYPE = "AVERAGE")
)

advs <- tibble::tribble(
  ~USUBJID, ~VSSEQ, ~PARAM, ~AVAL, ~VSSTRESU, ~VISIT, ~VSDTC,
  "XYZ-001-001", 1164, "Weight", 99, "kg", "Screening", "2018-03-19",
  "XYZ-001-001", 1165, "Weight", 101, "kg", "Run-In", "2018-03-26",
  "XYZ-001-001", 1166, "Weight", 100, "kg", "Baseline", "2018-04-16",
  "XYZ-001-001", 1167, "Weight", 94, "kg", "Week 24", "2018-09-30",
  "XYZ-001-001", 1168, "Weight", 92, "kg", "Week 48", "2019-03-17",
  "XYZ-001-001", 1169, "Weight", 95, "kg", "Week 52", "2019-04-14",
)

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

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

# Compute the average of AVAL only if there are more than 2 records within the
# by group
get_summary_records(
  adeg,
```

```
  by_vars = vars(USUBJID, PARAM, AVISIT),
  filter = dplyr::n() > 2,
  analysis_var = AVAL,
  summary_fun = function(x) mean(x, na.rm = TRUE),
  set_values_to = vars(DTYPE = "AVERAGE")
)
```

---

`get_terms_from_db` *Get Terms from the Queries Database*

---

## Description

The function checks if all requirements to access the database are fulfilled (version and access function are available, see `assert_db_requirements()`), reads the terms from the database, and checks if the dataset with the terms is in the expected format (see `assert_terms()`).

## Usage

```
get_terms_from_db(
  version,
  fun,
  queries,
  definition,
  expect_query_name = FALSE,
  expect_query_id = FALSE,
  i,
  temp_env,
  type
)
```

## Arguments

| | |
|---|---|
| `version` | Version<br>The version must be non null. Otherwise, an error is issued. The value is passed to the access function (`fun`). |
| `fun` | Access function<br>The access function must be non null. Otherwise, an error is issued. The function is called to retrieve the terms. |
| `queries` | Queries<br>List of all queries passed to `create_query_data()`. It is used for error messages. |
| `definition` | Definition of the query<br>The definition is passed to the access function. It defines which terms are returned. |
| `expect_query_name` | Is `QUERY_NAME` expected in the output dataset? |

`expect_query_id`
: Is `QUERY_ID` expected in the output dataset?

`i`
: Index of `definition` in `queries`
The value is used for error messages.

`temp_env`
: Temporary environment
The value is passed to the access function.

`type`
: Type of query
*Permitted Values*: `"smq"`, `"sdg"`

## Value

Output dataset of the access function

## Author(s)

Stefan Bundfuss

## See Also

OCCDS Functions: `create_query_data()`, `create_single_dose_dataset()`, `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-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"`

`min_dates` Minimum dates

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

```
impute_dtc_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 character vector

## Author(s)

Samia Kabi, Stefan Bundfuss

## 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()

## Examples

```
library(lubridate)

dates <- c(
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18T15",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019",
  "2019---07",
  ""
```

```
)

# No date imputation (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_dtm(
  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"),
    ymd("2020-11-11")
  ),
  highest_imputation = "Y"
)
```

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

## Description

Imputation partial date/time portion of a '--DTC' variable. based on user input.

## Usage

```
impute_dtc_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 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

```
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

## Author(s)

Samia Kabi, Stefan Bundfuss

## 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_dt()

## Examples

```
library(lubridate)

dates <- c(
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18T15",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019",
  "2019---07",
  ""
)

# No date imputation (highest_imputation defaulted to "h")
# Missing time part imputed with 00:00:00 portion by default
impute_dtc_dtm(dtc = dates)

# No date imputation (highest_imputation defaulted to "h")
# Missing time part imputed with 23:59:59 portion
impute_dtc_dtm(
  dtc = dates,
  time_imputation = "23:59:59"
)

# Same as above
```

```
impute_dtc_dtm(
  dtc = dates,
  time_imputation = "last"
)

# Impute to first day/month if date is partial
# Missing time part imputed with 00:00:00 portion by default
impute_dtc_dtm(
  dtc = dates,
  highest_imputation = "M"
)
# 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"),
    ymd_hms("2020-11-11T11:11:11")
  ),
  highest_imputation = "Y"
```

```
)
```

---

`list_all_templates` *List All Available ADaM Templates*

---

## Description

List All Available ADaM Templates

## Usage

```
list_all_templates(package = "admiral")
```

## Arguments

| | |
|---|---|
| `package` | The R package in which to look for templates. By default `"admiral"`. |

## Value

A `character` vector of all available templates

## Author(s)

Shimeng Huang, Thomas Neitmann

## See Also

Utilities used for examples and template scripts: `use_ad_template()`

## Examples

```
list_all_templates()
```

---

`list_tte_source_objects` *List all* `tte_source` *Objects Available in a Package*

---

## Description

List all `tte_source` Objects Available in a Package

## Usage

```
list_tte_source_objects(package = "admiral")
```

### Arguments

| | |
|---|---|
| package | The name of the package in which to search for tte_source objects |

### Value

A data.frame where each row corresponds to one tte_source object or NULL if package does not contain any tte_source objects

### Author(s)

Thomas Neitmann

### See Also

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

### 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 |

### Author(s)

Stefan Bundfuss

### See Also

Utilities for Filtering Observations: count_vals(), filter_confirmation(), filter_extreme(), filter_relative(), min_cond()

## Examples

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

---

# 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 |

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Filtering Observations: count_vals(), filter_confirmation(), filter_extreme(), filter_relative(), max_cond()

## Examples

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

---

# params *Create a Set of Parameters*

---

## Description

Create a set of variable parameters/function arguments to be used in `call_derivation()`.

## Usage

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

## Arguments

| | |
|---|---|
| `...` | One or more named arguments |

## Value

An object of class `params`

## Author(s)

Thomas Neitmann, Tracey Wang

## See Also

call_derivation()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

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

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

## In order to derive both `ASTDT` and `AENDT` in `ADAE`, one can use `derive_vars_dt()`
adae %>%
  derive_vars_dt(
    new_vars_prefix = "AST",
    dtc = AESTDTC,
    date_imputation = "first",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = vars(TRTSDT),
    max_dates = vars(TRTEDT)
  )

## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go.
## The function arguments which are different from a variable to another (e.g. `new_vars_prefix`,
## `dtc`, and `date_imputation`) are specified as a list of `params()` in the `variable_params`
## argument of `call_derivation()`. All other arguments which are common to all variables
## (e.g. `min_dates` and `max_dates`) are specified outside of `variable_params` (i.e. in `...`).
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
  variable_params = list(
    params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
```

```
    params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
  ),
  min_dates = vars(TRTSDT),
  max_dates = vars(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the call using `derive_vars_dt()`
## to derive variables `ASTDT` and `AENDT` separately at the beginning.
```

---

`print.derivation_slice` *Print* `derivation_slice` *Objects*

---

## Description

Print `derivation_slice` Objects

## Usage

```
## S3 method for class 'derivation_slice'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `derivation_slice` object |
| `...` | Not used |

## Value

No return value, called for side effects

## See Also

`derivation_slice()`

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

## Examples

```
print(death_event)
```

## queries *Queries Dataset*

### 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_adsl`, `atoxgr_criteria_ctcv4`, `ex_single`, `queries_mh`

## 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_adsl`, `atoxgr_criteria_ctcv4`, `ex_single`, `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

| | |
|---|---|
| prefix | The value is used to populate VAR_PREFIX in the output dataset of create_query_data(), e.g., "SMQ03" |
| name | The value is used to populate QUERY_NAME 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.<br>*Permitted Values*: A character scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an smq_select() or sdg_select() object. |
| id | The value is used to populate QUERY_ID 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.<br>*Permitted Values*: A integer scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an smq_select() or sdg_select() object. |
| add_scope_num | Determines if QUERY_SCOPE_NUM in the output dataset of create_query_data() is populated<br>If the parameter is set to TRUE, the definition must be an smq_select() object.<br>*Default*: FALSE<br>*Permitted Values*: TRUE, FALSE |
| definition | Definition of terms belonging to the query<br>There are four different ways to define the terms:<br>• An smq_select() object is specified to select a query from the SMQ database.<br>• An sdg_select() object is specified to select a query from the SDG database.<br>• A data frame with columns TERM_LEVEL and TERM_NAME or TERM_ID can be specified to define the terms of a customized query. The TERM_LEVEL should be set to the name of the variable which should be used to select the terms, e.g., "AEDECOD" or "AELLTCD". TERM_LEVEL does not need to be constant within a query. For example a query can be based on AEDECOD and AELLT. |

If TERM_LEVEL refers to a character variable, TERM_NAME should be set to the value the variable. If it refers to a numeric variable, TERM_ID should be set to the value of the variable. If only character variables or only numeric variables are used, TERM_ID or TERM_NAME respectively can be omitted.

- A list of data frames and smq_select() objects can be specified to define a customized query based on custom terms and SMQs. The data frames must have the same structure as described for the previous item.

*Permitted Values*: an smq_select() object, an sdg_select() object, a data frame, or a list of data frames and smq_select() objects.

## Value

An object of class query.

## Author(s)

Stefan Bundfuss

## See Also

create_query_data(), smq_select(), sdg_select(), Queries Dataset Documentation

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

## Examples

```
# create a query for an SMQ
library(tibble)
library(magrittr, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)

query(
  prefix = "SMQ02",
  id = auto,
  definition = smq_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW"
  )
)

# create a query for an SDG
query(
  prefix = "SDG01",
  id = auto,
  definition = sdg_select(
    name = "5-aminosalicylates for ulcerative colitis"
```

```
  )
)

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

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

# creating a customized query based on SMQs and additional terms
query(
  prefix = "CQ03",
  name = "Special issues of interest",
  definition = list(
    cqterms,
    smq_select(
      name = "Pregnancy and neonatal topics (SMQ)",
      scope = "NARROW"
    ),
    smq_select(
      id = 8050L,
      scope = "BROAD"
    )
  )
)
```

---

`restrict_derivation` *Execute a Derivation on a Subset of the Input Dataset*

---

## Description

Execute a derivation on a subset of the input dataset.

## Usage

```
restrict_derivation(dataset, derivation, args = NULL, filter)
```

## Arguments

| | |
|---|---|
| `dataset` | Input dataset |
| `derivation` | Derivation |

| | |
|---|---|
| args | Arguments of the derivation<br>A params() object is expected. |
| filter | Filter condition |

## Author(s)

Stefan Bundfuss

## See Also

params() slice_derivation()

Higher Order Functions: call_derivation(), derivation_slice(), print.derivation_slice(), slice_derivation()

## Examples

```
library(magrittr)
adlb <- tibble::tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~ABLFL,
  "1",            -1,   113, NA_character_,
  "1",             0,   113, "Y",
  "1",             3,   117, NA_character_,
  "2",             0,    95, "Y",
  "3",             0,   111, "Y",
  "3",             1,   101, NA_character_,
  "3",             2,   123, NA_character_
)

# Derive BASE for post-baseline records only (derive_var_base() can not be used in this case
# as it requires the baseline observation to be in the input dataset)
restrict_derivation(
  adlb,
  derivation = derive_vars_merged,
  args = params(
    by_vars = vars(USUBJID),
    dataset_add = adlb,
    filter_add = ABLFL == "Y",
    new_vars = vars(BASE = AVAL)
  ),
  filter = AVISITN > 0
)

# Derive BASE for baseline and post-baseline records only
restrict_derivation(
  adlb,
  derivation = derive_var_base,
  args = params(
    by_vars = vars(USUBJID)
  ),
  filter = AVISITN >= 0
```

```
) %>%
  # Derive CHG for post-baseline records only
  restrict_derivation(
    derivation = derive_var_chg,
    filter = AVISITN > 0
  )
```

---

`restrict_imputed_dtc_dt` *Restrict Imputed DTC date to Minimum/Maximum Dates*

---

## Description

Restrict Imputed DTC date to Minimum/Maximum Dates

## Usage

```
restrict_imputed_dtc_dt(dtc, imputed_dtc, min_dates, max_dates)
```

## Arguments

| | |
|---|---|
| `dtc` | The `'--DTC'` date to impute<br>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. |
| `imputed_dtc` | The imputed DTC date |
| `min_dates` | Minimum dates<br>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<br>`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" )`<br>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.

## Value

- The last of the minimum dates (`min_dates`) which are in the range of the partial DTC date (`dtc`)
- The first of the maximum dates (`max_dates`) which are in the range of the partial DTC date (`dtc`)
- `imputed_dtc` if the partial DTC date (`dtc`) is not in range of any of the minimum or maximum dates.

## Author(s)

Stefan Bundfuss

## See Also

`impute_dtc_dtm()`, `impute_dtc_dt()`

Utilities used for date imputation: `dt_level()`, `dtm_level()`, `get_imputation_target_date()`, `get_imputation_target_time()`, `get_partialdatetime()`, `restrict_imputed_dtc_dtm()`

---

# `restrict_imputed_dtc_dtm` *Restrict Imputed DTC date to Minimum/Maximum Dates*

---

## Description

Restrict Imputed DTC date to Minimum/Maximum Dates

## Usage

```
restrict_imputed_dtc_dtm(dtc, imputed_dtc, min_dates, max_dates)
```

## 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.

`imputed_dtc` The imputed DTC date

`min_dates` Minimum dates

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

```
impute_dtc_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.

## Value

- The last of the minimum dates (`min_dates`) which are in the range of the partial DTC date (`dtc`)
- The first of the maximum dates (`max_dates`) which are in the range of the partial DTC date (`dtc`)
- `imputed_dtc` if the partial DTC date (`dtc`) is not in range of any of the minimum or maximum dates.

## Author(s)

Stefan Bundfuss

## See Also

impute_dtc_dtm(), impute_dtc_dt()

Utilities used for date imputation: dt_level(), dtm_level(), get_imputation_target_date(), get_imputation_target_time(), get_partialdatetime(), restrict_imputed_dtc_dt()

---

# sdg_select *Create an* sdg_select *object*

---

## Description

Create an sdg_select object

## Usage

```
sdg_select(name = NULL, id = NULL)
```

## Arguments

| | |
|---|---|
| name | Name of the query used to select the definition of the query from the company database. |
| id | Identifier of the query used to select the definition of the query from the company database. |

## Details

Exactly one name or id must be specified.

## Value

An object of class sdg_select.

## Author(s)

Stefan Bundfuss

## See Also

create_query_data(), query()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), smq_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

signal_duplicate_records *Signal Duplicate Records*

## Description

Signal Duplicate Records

## Usage

```
signal_duplicate_records(
  dataset,
  by_vars,
  msg = paste("Dataset contains duplicate records with respect to",
    enumerate(vars2chr(by_vars))),
  cnd_type = "error"
)
```

## Arguments

| | |
|---|---|
| dataset | A data frame |
| by_vars | A list of variables created using vars() identifying groups of records in which to look for duplicates |
| msg | The condition message |
| cnd_type | Type of condition to signal when detecting duplicate records. One of "message", "warning" or "error". Default is "error". |

## Value

No return value, called for side effects

## Author(s)

Thomas Neitmann

## 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, vars(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

| | |
|---|---|
| `dataset` | Input dataset |
| `derivation` | Derivation |
| `args` | Arguments of the derivation<br>A `param()` object is expected. |
| `...` | A `derivation_slice()` object is expected<br>Each slice defines a subset of the input dataset and some of the parameters for the derivation. The derivation is called on the subset with the parameters specified by the `args` parameter and the `args` field of the `derivation_slice()` object. If a parameter is specified for both, the value in `derivation_slice()` overwrites the one in `args`. |

## Details

For each slice the derivation is called on the subset defined by the `filter` field of the `derivation_slice()` object and with the parameters specified by the `args` parameter and the `args` field of the `derivation_slice()` object. If a parameter is specified for both, the value in `derivation_slice()` overwrites the one in `args`.

- Observations that match with more than one slice are only considered for the first matching slice.
- Observations with no match to any of the slices are included in the output dataset but the derivation is not called for them.

## Value

The input dataset with the variables derived by the derivation added

## Author(s)

Stefan Bundfuss

## See Also

params() restrict_derivation()

Higher Order Functions: call_derivation(), derivation_slice(), print.derivation_slice(), restrict_derivation()

## Examples

```
library(stringr)
advs <- tibble::tribble(
  ~USUBJID, ~VSDTC,       ~VSTPT,
  "1",      "2020-04-16", NA_character_,
  "1",      "2020-04-16", "BEFORE TREATMENT"
)

# For the second slice filter is set to TRUE. Thus derive_vars_dtm is called
# with time_imputation = "last" for all observations which do not match for the
# first slice.
slice_derivation(
  advs,
  derivation = derive_vars_dtm,
  args = params(
    dtc = VSDTC,
    new_vars_prefix = "A"
  ),
  derivation_slice(
    filter = str_detect(VSTPT, "PRE|BEFORE"),
    args = params(time_imputation = "first")
  ),
  derivation_slice(
    filter = TRUE,
    args = params(time_imputation = "last")
  )
)
```

---

# smq_select *Create an* smq_select *object*

---

## Description

Create an smq_select object

## Usage

```
smq_select(name = NULL, id = NULL, scope = NULL)
```

### Arguments

| | |
|---|---|
| name | Name of the query used to select the definition of the query from the company database. |
| id | Identifier of the query used to select the definition of the query from the company database. |
| scope | Scope of the query used to select the definition of the query from the company database.<br>*Permitted Values*: "BROAD", "NARROW" |

### Details

Exactly one of name or id must be specified.

### Value

An object of class smq_select.

### Author(s)

Stefan Bundfuss

### See Also

create_query_data(), query()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), tte_source(), validate_query(), validate_sdg_select(), validate_smq_select()

---

## 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<br>The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). |
| filter | An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. |
| date | A variable providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol is expected.<br>Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. |
| censor | Censoring value<br>CDISC strongly recommends using 0 for events and positive integers for censoring. |
| set_values_to | A named list returned by vars() defining the variables to be set for the event or censoring, e.g. vars(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, or NA. |

## Value

An object of class tte_source

## Author(s)

Stefan Bundfuss

## See Also

derive_param_tte(), censor_source(), event_source()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), validate_query(), validate_sdg_select(), validate_smq_select()

---

# 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, ADMH, ADPP, 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

## Author(s)

Shimeng Huang, Thomas Neitmann

## See Also

Utilities used for examples and template scripts: list_all_templates()

## Examples

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

## 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.

### Author(s)

Stefan Bundfuss

### See Also

query()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_sdg_select(), validate_smq_select()

## validate_sdg_select *Validate an object is indeed a* sdg_select *object*

### Description

Validate an object is indeed a sdg_select object

### Usage

```
validate_sdg_select(obj)
```

### Arguments

| | |
|---|---|
| obj | An object to be validated. |

## Value

The original object.

## Author(s)

Stefan Bundfuss

## See Also

sdg_select()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_smq_select()

---

# validate_smq_select *Validate an object is indeed a* smq_select *object*

---

## Description

Validate an object is indeed a smq_select object

## Usage

```
validate_smq_select(obj)
```

## Arguments

| | |
|---|---|
| obj | An object to be validated. |

## Value

The original object.

## Author(s)

Stefan Bundfuss

## See Also

smq_select()

Source Specifications: assert_db_requirements(), assert_terms(), assert_valid_queries(), censor_source(), date_source(), death_event, derive_var_dthcaus(), event_source(), extend_source_datasets(), filter_date_sources(), format.sdg_select(), format.smq_select(), list_tte_source_objects(), params(), query(), sdg_select(), smq_select(), tte_source(), validate_query(), validate_sdg_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

## Author(s)

Stefan Bundfuss

## See Also

Utilities for Formatting Observations: convert_blanks_to_na(), format_eoxxstt_default(), format_reason_default()

## Examples

```
yn_to_numeric(c("Y", "N", NA_character_))
```

# Index