Package 'admiral'

May 28, 2026

Type Package

Title ADaM in R Asset Library

Version 1.4.2

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

License Apache License (>= 2)

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

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

Depends R (>= 4.1)

Imports admiraldev (>= 1.4.0), cli (>= 3.6.2), dplyr (>= 1.1.1), 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 (>= 1.1.0)

Suggests diffdf, DT, here, htmltools, knitr, methods, pharmaversesdtm (>= 1.0.0), reactable, readxl, rmarkdown, testthat (>= 3.0.0), tibble, withr

VignetteBuilder knitr

Config/Needs/website gert, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Edoardo Mancini [aut, cre] (ORCID:
<https://orcid.org/0009-0006-4899-8641>),
Stefan Bundfuss [aut] (ORCID: <https://orcid.org/0009-0005-0027-1198>),
Matt Bearham [aut],
Arianna Cascone [aut] (ORCID: <https://orcid.org/0000-0001-5948-2831>),
Kristin Dahnert [aut],
Jeffrey Dickinson [aut],
Ross Farrugia [aut],
Fanny Gautier [aut],
Gordon Miller [aut],
Lina Patil [aut],
Jim Rothstein [aut] (ORCID: <https://orcid.org/0009-0009-8659-6071>),
Ben Straub [aut],
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline LLC [cph, fnd]

Maintainer Edoardo Mancini <edoardo.mancini@roche.com>

Repository CRAN

Date/Publication 2026-05-28 05:11:30 UTC

Contents

`admiral_adlb` *Lab Analysis Dataset*

Description

An example of lab analysis dataset

Usage

```
admiral_adlb
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 330 rows and 115 columns.

Source

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

See Also

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

---

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 54 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: admiral_adlb, ex_single, example_qs, queries, queries_mh

---

atoxgr_criteria_ctcv4 *Metadata Holding Grading Criteria for NCI-CTCAEv4 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv4 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv4
```

Format

An object of class `data.frame` with 42 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev4.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains 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.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

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

See Also

Other metadata: `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `country_code_lookup`, `dose_freq_lookup`

`atoxgr_criteria_ctcv4_uscv` *Metadata Holding Grading Criteria for NCI-CTCAEv4 using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv4 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv4_uscv
```

Format

An object of class `data.frame` with 48 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev4_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains 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.
- `UNIT_CHECK`: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

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

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv5 *Metadata Holding Grading Criteria for NCI-CTCAEv5 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv5 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv5
```

Format

An object of class `data.frame` with 39 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev5.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains 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.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.

- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

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

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv5_uscv

Metadata Holding Grading Criteria for NCI-CTCAEv5 using USCV unit where applicable

---

Description

Metadata Holding Grading Criteria for NCI-CTCAEv5 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv5_uscv
```

Format

An object of class data.frame with 45 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file data-raw/adlb_grading/ncictcaev5_uscv.json. The variables GRADE_NA_CODE, GRADE_4_CODE, GRADE_3_CODE, GRADE_2_CODE and GRADE_1_CODE in the json file are combined to create GRADE_CRITERIA_CODE, and then dropped from metadata. The dataset contains 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.
- UNIT_CHECK: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- VAR_CHECK: List of variables required to implement lab grade criteria. Use to check against input data.
- DIRECTION: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

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

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv6 *Metadata Holding Grading Criteria for NCI-CTCAEv6 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv6 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv6
```

Format

An object of class data.frame with 43 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file data-raw/adlb_grading/ncictcaev6.json. The variables GRADE_NA_CODE, GRADE_4_CODE, GRADE_3_CODE, GRADE_2_CODE and GRADE_1_CODE in the json file are combined to create GRADE_CRITERIA_CODE, and then dropped from metadata. The dataset contains 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.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v6.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_us` `country_code_lookup`, `dose_freq_lookup`

`atoxgr_criteria_ctcv6_uscv` *Metadata Holding Grading Criteria for NCI-CTCAEv6 using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv6 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv6_uscv
```

Format

An object of class `data.frame` with 48 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev6_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains 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.
- `UNIT_CHECK`: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v6.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `country_code_lookup`, `dose_freq_lookup`

---

atoxgr_criteria_daids *Metadata Holding Grading Criteria for DAIDs using SI unit where applicable*

Description

Metadata Holding Grading Criteria for DAIDs using SI unit where applicable

Usage

```
atoxgr_criteria_daids
```

Format

An object of class `data.frame` with 63 rows and 14 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/DAIDS.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `SUBGROUP` : Description of sub-group of subjects were grading will be applied (i.e. >= 18 years)
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `FILTER` : `admiral` code to apply the filter based on SUBGROUP column.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold SI 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`, `SUBGROUP`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on DAIDS website defining the grading criteria, (Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events). From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids_us country_code_lookup, dose_freq_lookup

atoxgr_criteria_daids_uscv *Metadata Holding Grading Criteria for DAIDs using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for DAIDs using USCV unit where applicable

Usage

```
atoxgr_criteria_daids_uscv
```

Format

An object of class `data.frame` with 71 rows and 14 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/DAIDS_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `SUBGROUP` : Description of sub-group of subjects were grading will be applied (i.e. >= 18 years)
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `FILTER` : `admiral` code to apply the filter based on SUBGROUP column.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold USCV 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, SUBGROUP, Grade 1, Grade 2,Grade 3,Grade 4,Grade 5, Definition are from the source document on DAIDS website defining the grading criteria. [Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events From these variables only 'TERM' is used in the {admiral} code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, country_code_lookup, dose_freq_lookup

---

basket_select *Create a* basket_select *object*

Description

Create a basket_select object

Usage

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

Arguments

name	Name of the query used to select the definition of the query from the company database.
id	Identifier of the query used to select the definition of the query from the company database.
scope	Scope of the query used to select the definition of the query from the company database.
type	The type argument expects a character scalar. It is passed to the company specific get_terms() function such that the function can determine which sort of basket is requested
...	Any number of *named* function arguments. Can be used to pass in company specific conditions or flags that will then be used in user-defined function that is passed into argument get_terms_fun for function create_query_data().

Details

Exactly one of name or id must be specified.

Value

An object of class basket_select.

See Also

create_query_data(), query()

Source Objects: censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

---

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	Input dataset
derivation	The derivation function to call A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the params() objects passed to the variable_params and ... arguments.
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 .... @details It is also possible to pass functions from outside the {admiral} package to call_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

Value

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

See Also

params() restrict_derivation() call_derivation()

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

Examples

```
library(dplyr, warn.conflicts = FALSE)
adsl <- tribble(
  ~STUDYID,   ~USUBJID,      ~TRTSDT,      ~TRTEDT,
  "PILOT01", "01-1307",           NA,           NA,
  "PILOT01", "05-1377", "2014-01-04", "2014-01-25",
  "PILOT01", "06-1384", "2012-09-15", "2012-09-24",
  "PILOT01", "15-1085", "2013-02-16", "2013-08-18",
  "PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
  mutate(
    across(TRTSDT:TRTEDT, as.Date)
  )

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

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

## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
  variable_params = list(
```

```
    params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
    params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
  ),
  min_dates = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)

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

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

Description

[Deprecated]

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

See Also

Other deprecated: `date_source()`, `derive_param_extreme_record()`, `derive_var_dthcaus()`, `derive_var_extreme_dt()`, `derive_var_extreme_dtm()`, `derive_var_merged_summary()`, `dthcaus_source()`, `get_summary_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.

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

Usage

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

Arguments

`dataset_name`	The name of the source dataset The name refers to the dataset provided by the `source_datasets` parameter of `derive_param_tte()`.
`filter`	An unquoted condition for selecting the observations from `dataset` which are events or possible censoring time points.
`date`	A variable or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to `derive_vars_dt()` or `convert_dtc_to_dt()` to impute and derive a date from a date character vector to a date object.
`censor`	Censoring value CDISC strongly recommends using `0` for events and positive integers for censoring.

set_values_to	A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.
order	Sort order An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date.

Value

An object of class censor_source, inheriting from class tte_source

See Also

derive_param_tte(), event_source()

Source Objects: basket_select(), death_event, event(), event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

Examples

```
# Last study date known alive censor

censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = exprs(
    EVNTDESC = "ALIVE",
    SRCDOM = "ADSL",
    SRCVAR = "LSTALVDT"
  )
)
```

---

chr2vars *Turn a Character Vector into a List of Expressions*

Description

Turn a character vector into a list of expressions

Usage

```
chr2vars(chr)
```

Arguments

chr	A character vector

Value

A list of expressions as returned by exprs()

See Also

Utilities for working with quosures/list of expressions: negate_vars()

Examples

```
chr2vars(c("USUBJID", "AVAL"))
```

---

compute_age_years *Compute Age in Years*

Description

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

Usage

```
compute_age_years(age, age_unit)
```

Arguments

age	The ages to convert. A numeric vector is expected.
age_unit	Age unit. Either a string containing the time unit of all ages in age or a character vector containing the time units of each age in age is expected. Note that permitted values are cases insensitive (e.g. "YEARS" is treated the same as "years" and "Years").

Details

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

Value

The ages contained in age converted to years.

See Also

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

Examples

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

compute_age_years(
  age = c(10, 520, 3650, 1000),
  age_unit = c("YEARS", "WEEKS", "DAYS", NA_character_)
)
```

---

compute_bmi *Compute Body Mass Index (BMI)*

Description

Computes BMI from height and weight

Usage

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

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm.
weight	WEIGHT value It is expected that WEIGHT is in kg.

Details

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

Value

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

See Also

derive_param_bmi()

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

Examples

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

compute_bsa *Compute Body Surface Area (BSA)*

Description

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

Usage

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

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm.
weight	WEIGHT value It is expected that WEIGHT is in kg.
method	Derivation method to use: Mosteller: sqrt(height * weight / 3600) DuBois-DuBois: 0.007184 * height ^ 0.725 * weight ^ 0.425 Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378 Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456 Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight))) Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444 Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425

Details

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

Value

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

See Also

derive_param_bsa()

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

Examples

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

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

---

compute_dtf *Derive the Date Imputation Flag*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

See Also

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

Examples

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

---

compute_duration *Compute Duration*

Description

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

Usage

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

Arguments

start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Refer to convert_dtc_to_dt() to obtain a vector of imputed dates.
end_date	The end date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Refer to convert_dtc_to_dt() to obtain a vector of imputed dates.
in_unit	Input unit See floor_in and add_one parameter for details. Permitted Values (case-insensitive): For years: "year", "years", "yr", "yrs", "y" For months: "month", "months", "mo", "mos"

For days: "day", "days", "d"

For hours: "hour", "hours", "hr", "hrs", "h"

For minutes: "minute", "minutes", "min", "mins"

For seconds: "second", "seconds", "sec", "secs", "s"

`out_unit` Output unit

The duration is derived in the specified unit

Permitted Values (case-insensitive):

For years: "year", "years", "yr", "yrs", "y"

For months: "month", "months", "mo", "mos"

For weeks: "week", "weeks", "wk", "wks", "w"

For days: "day", "days", "d"

For hours: "hour", "hours", "hr", "hrs", "h"

For minutes: "minute", "minutes", "min", "mins"

For seconds: "second", "seconds", "sec", "secs", "s"

`floor_in` Round down input dates?

The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.

`add_one` Add one input unit?

If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.

`trunc_out` Return integer part

The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.

`type` lubridate duration type.

See below for details.

Details

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

Value

The duration between the two date in the specified unit

Duration Type

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

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

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

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

See Also

`derive_vars_duration()`

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

Examples

```
library(lubridate)

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

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

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

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

compute_egfr *Compute Estimated Glomerular Filtration Rate (eGFR) for Kidney Function*

Description

Compute Kidney Function Tests:

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

Usage

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

Arguments

creat	Creatinine A numeric vector is expected.
creatu	Creatinine Units A character vector is expected. Expected Values: "SI", "CV", "umol/L", "mg/dL"
age	Age (years) A numeric vector is expected.
weight	Weight (kg) A numeric vector is expected if method = "CRCL"
sex	Gender A character vector is expected. Expected Values: "M", "F"
race	Race A character vector is expected if method = "MDRD" Expected Values: "BLACK OR AFRICAN AMERICAN" and others
method	Method A character vector is expected. Expected Values: "CRCL", "CKD-EPI", "MDRD"

Details

Calculates an estimate of Glomerular Filtration Rate (eGFR)

CRCL Creatinine Clearance (Cockcroft-Gault)

For Creatinine in umol/L:

$$\frac{(140 - age) \times weight(kg) \times constant}{Serum\ Creatinine(\mu mol/L)}$$

$$Constant = 1.04\ for\ females, 1.23\ for\ males$$

For Creatinine in mg/dL:

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

units = mL/min

CKD-EPI Chronic Kidney Disease Epidemiology Collaboration formula

$$eGFR = 142 \times min(SCr/\kappa, 1)^{\alpha} \times max(SCr/\kappa, 1)^{-1.200} \times 0.9938^{Age} \times 1.012[if\ female]$$

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

$$\kappa$$

= 0.7 (females) or 0.9 (males)

$$\alpha$$

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

MDRD Modification of Diet in Renal Disease formula

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

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

units = mL/min/1.73 m2

Value

A numeric vector of egfr values

See Also

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

Examples

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

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

compute_egfr(
```

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

compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, sex = "M", method = "CKD-EPI"
)


base <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~AGE, ~SEX, ~RACE, ~WTBL, ~CREATBL, ~CREATBLU,
  "P01", "P01-1001", 55, "M", "WHITE", 90.7, 96.3, "umol/L",
  "P01", "P01-1002", 52, "F", "BLACK OR AFRICAN AMERICAN", 68.5, 70, "umol/L",
  "P01", "P01-1003", 67, "M", "BLACK OR AFRICAN AMERICAN", 85.0, 77, "umol/L",
  "P01", "P01-1004", 76, "F", "ASIAN", 60.7, 65, "umol/L",
)

base %>%
  dplyr::mutate(
    CRCL_CG = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      method = "CRCL"
    ),
    EGFR_EPI = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      method = "CKD-EPI"
    ),
    EGFR_MDRD = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      race = RACE, method = "MDRD"
    ),
  )
```

`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 A numeric vector is expected.
chol	Total serum cholesterol (mg/dL) A numeric vector is expected.
cholhdl	HDL serum cholesterol (mg/dL) A numeric vector is expected.
age	Age (years) A numeric vector is expected.
sex	Gender A character vector is expected. Expected Values: 'M' 'F'
smokefl	Smoking Status A character vector is expected. Expected Values: 'Y' 'N'
diabetfl	Diabetic Status A character vector is expected. Expected Values: 'Y' 'N'
trthypfl	Treated for hypertension status A character vector is expected. Expected Values: 'Y' 'N'

Details

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

For Women:

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

See Also

`derive_param_framingham()`

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

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 A numeric vector is expected.
sysbp	Systolic blood pressure A numeric vector is expected.
hr	Heart rate A numeric vector or NULL is expected.

Details

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

if it is based on diastolic and systolic blood pressure and

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

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

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

Value

A numeric vector of MAP values

See Also

derive_param_map()

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

Examples

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

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

---

compute_qtc *Compute Corrected QT*

Description

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

Usage

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

Arguments

qt	QT interval A numeric vector is expected. It is expected that QT is measured in ms or msec.
rr	RR interval A numeric vector is expected. It is expected that RR is measured in ms or msec.
method	Method used to QT correction

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 ms

See Also

derive_param_qtc()

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

Examples

```
compute_qtc(qt = 350, rr = 857, method = "Bazett")

compute_qtc(qt = 350, rr = 857, method = "Fridericia")

compute_qtc(qt = 350, rr = 857, 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

See Also

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

Examples

```
compute_qual_imputation("<40")
compute_qual_imputation(c("3", ">30.2"))
```

`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

See Also

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

Examples

```
compute_qual_imputation_dec("<40.1")
compute_qual_imputation_dec(c("0.35", "1"))
```

compute_rr *Compute RR Interval From Heart Rate*

Description

Computes RR interval from heart rate.

Usage

```
compute_rr(hr)
```

Arguments

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

Details

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

Value

RR interval in ms:

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

See Also

derive_param_rr()

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

Examples

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

compute_scale *Compute Scale Parameters*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

See Also

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

Examples

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

---

compute_tmf *Derive the Time Imputation Flag*

Description

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

Usage

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

Arguments

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

ignore_seconds_flag

ADaM IG states that given SDTM (--DTC) variable, if only hours and minutes are ever collected, and seconds are imputed in (*DTM) as 00, then it is not necessary to set (*TMF) to "S".

By default it is assumed that no seconds are collected and *TMF shouldn't be set to "S". A user can set this to FALSE if seconds are collected.

The default value of ignore_seconds_flag is set to TRUE in admiral 1.4.0 and later.

Details

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

Value

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

See Also

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

Examples

```
library(lubridate)

compute_tmf(dtc = "2019-07-18T15:25", dtm = ymd_hm("2019-07-18T15:25"))
compute_tmf(dtc = "2019-07-18T15", dtm = ymd_hm("2019-07-18T15:25"))
compute_tmf(dtc = "2019-07-18", dtm = ymd("2019-07-18"))
compute_tmf(dtc = "2022-05--T00:00", dtm = ymd_hm("2022-05-15T23:59"))
compute_tmf(dtc = "2022-05--T23:00", dtm = ymd_hm("2022-05-15T23:59"))
compute_tmf(
  dtc = "2022-05--T23:59:00",
  dtm = ymd_hms("2022-05-15T23:59:59"),
  ignore_seconds_flag = FALSE
)
```

---

consolidate_metadata *Consolidate Multiple Meta Datasets Into a Single One*

Description

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

Usage

```
consolidate_metadata(
  datasets,
  key_vars,
  source_var = SOURCE,
  check_vars = "warning",
  check_type = "error"
)
```

Arguments

datasets	List of datasets to consolidate
key_vars	Key variables The specified variables must be a unique of all input datasets.
source_var	Source variable The specified variable is added to the output dataset. It is set the name of the dataset the observation is originating from.
check_vars	Check variables? If "message", "warning", or "error" is specified, a message is issued if the variable names differ across the input datasets (datasets).
check_type	Check uniqueness? If "warning" or "error" is specified, a message is issued if the key variables (key_vars) are not a unique key in all of the input datasets (datasets).

Details

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

Value

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

See Also

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

Examples

```
library(tibble)
glob_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "PULSE",      60,    100,
  "SYSBP",      90,    130,
  "DIABP",      60,     80
)
proj_ranges <- tribble(
```

```
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "SYSBP",     100,    140,
  "DIABP",      70,     90
)
stud_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "BMI",        18,     25
)

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

---

convert_blanks_to_na *Convert Blank Strings Into NAs*

Description

Turn SAS blank strings into proper R NAs.

Usage

```
convert_blanks_to_na(x)

## 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

See Also

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

Examples

```
library(tibble)

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

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

`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'`).

Note: This is a wrapper function for the function `convert_dtc_to_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.

A date or character date is expected in a format like `yyyy-mm-ddThh:mm:ss`.

`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.

`date_imputation`

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

A character value is expected.

- If `highest_imputation` is `"M"`, month and day can be specified as `"mm-dd"`: e.g. `"06-15"` for the 15th of June
- When `highest_imputation` is `"M"` or `"D"`, the following keywords are available: `"first"`, `"mid"`, `"last"` to impute to the first/mid/last day/month. If `"mid"` is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as `"06-30"` (middle of the year).
  - If only day is missing, it is imputed as `"15"` (middle of the month).

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).

`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"`.

`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_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  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 lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

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

Details

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

Value

A datetime object

See Also

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

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" (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, NA_character_ is returned.

If "Y" (year, highest level) is specified, date_imputation must be "first" or "last" and min_dates or max_dates must be specified respectively. Otherwise, an error is thrown.

date_imputation

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

A character value is expected.

- If highest_imputation is "M", month and day can be specified as "mm-dd": e.g. "06-15" for the 15th of June
- When highest_imputation is "M" or "D", the following keywords are available: "first", "mid", "last" to impute to the first/mid/last day/month. If "mid" is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).

- If only day is missing, it is imputed as "15" (middle of the month).

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).

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").

Details

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

Value

a date object

See Also

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

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.

`date_imputation`

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

A character value is expected.

- If `highest_imputation` is `"M"`, month and day can be specified as `"mm-dd"`: e.g. `"06-15"` for the 15th of June

- When highest_imputation is "M" or "D", the following keywords are available: "first", "mid", "last" to impute to the first/mid/last day/month. If "mid" is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).

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).

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".

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_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  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 lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.
For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").

Details

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

Value

A datetime object

See Also

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

Examples

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

---

convert_na_to_blanks *Convert NAs Into Blank Strings*

Description

Turn NAs to blank strings .

Usage

```
convert_na_to_blanks(x)

## Default S3 method:
convert_na_to_blanks(x)

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

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

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

Arguments

x	Any R object

Details

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

Value

An object of the same class as the input

See Also

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

Examples

```
library(tibble)

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

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

---

convert_xxtpt_to_hours

Convert XXTPT *Strings to Hours*

Description

[Experimental]

Converts CDISC timepoint strings (e.g., `PCTPT`, `VSTPT`, `EGTPT`, `ISTPT`, `LBTPT`) into numeric hours for analysis. The function handles common dose-centric formats including pre-dose, post-dose (hours/minutes), days, time ranges, and treatment-related time markers.

Usage

```
convert_xxtpt_to_hours(
  xxtpt,
  treatment_duration = 0,
  range_method = "midpoint"
)
```

Arguments

xxtpt	A character vector of timepoint descriptions from SDTM --TPT variables (e.g., PCTPT, VSTPT, EGTPT, ISTPT, LBTPT). Can contain NA values.
treatment_duration	Numeric value(s) specifying the duration of treatment in hours. Used to convert "EOI/EOT" (End of Infusion/Treatment) patterns and patterns describing time after end of treatment. Must be non-negative. Can be either: • A single value (used for all timepoints), or • A vector of the same length as xxtpt (one value per timepoint) Default is 0 hours (for instantaneous treatments like oral medications).
range_method	Method for converting time ranges to single values. Options are "midpoint" (default), "start", or "end". For example, "0-6h" with midpoint returns 3, with start returns 0, with end returns 6.

Details

The function recognizes the following patterns (all case-insensitive):

Special Cases:

- "Screening" -> 0
- "Pre-dose", "Predose", "Pre-treatment", "Pre-infusion", "Pre-inf", "Before", "Infusion", "0H" -> 0
- "EOI", "EOT", "End of Infusion", "End of Treatment", "After End of Infusion", "After End of Treatment" -> treatment_duration (default: 0)
- "Morning", "Evening" -> NA_real_
- Unrecognized values -> NA_real_

Time Ranges: Time ranges are converted based on the range_method parameter:

- "0-6h Post-dose" with range_method = "midpoint" (default) -> 3
- "0-6h Post-dose" with range_method = "start" -> 0
- "0-6h Post-dose" with range_method = "end" -> 6
- "0-4H PRIOR START OF INFUSION" with midpoint -> -2 (negative for prior)
- "8-16H POST START OF INFUSION" with midpoint -> 12
- "0-4H AFTER EOI" with midpoint and treatment_duration=1 -> 3 (1 + 2)
- "0-4H EOT" with midpoint and treatment_duration=0 -> 2

- "4-8H AFTER END OF INFUSION" with midpoint and treatment_duration=1 -> 7 (1 + 6)
- "4-8H POST INFUSION" with midpoint and treatment_duration=1 -> 7 (1 + 6)
- "4-8H POST-INF" with midpoint and treatment_duration=1 -> 7 (1 + 6)

Time-based Conversions:

- **Days**: "Day 1" -> 24, "2D" -> 48, "30 DAYS AFTER LAST" -> 720 (requires unit indicator; bare numbers like "2" return NA)
- **Hours + Minutes**: "1H30M" -> 1.5
- **Hours**: "2 hours" -> 2, "1 HOUR POST" -> 1
- **Minutes**: "30M" -> 0.5, "30 MIN POST" -> 0.5
- **Predose**: "5 MIN PREDOSE" -> -0.0833, "5 MIN PRE-DOSE" -> -0.0833
- **Before treatment**: "5 MIN BEFORE" -> -0.0833
- **Post EOI/EOT**: "1 HOUR POST EOI" -> treatment_duration + 1, "24 HR POST INF" -> treatment_duration + 24, "24 HR POST-INF" -> treatment_duration + 24, "1 HOUR AFTER EOT" -> treatment_duration + 1
- **After end**: "30MIN AFTER END OF INFUSION" -> treatment_duration + 0.5
- **Start of infusion/treatment**: "8H PRIOR START OF INFUSION" -> -8, "8H BEFORE START OF TREATMENT" -> -8
- **Pre EOI/EOT**: "10MIN PRE EOI" -> treatment_duration - 1/6, "10MIN BEFORE EOT" -> treatment_duration - 1/6

Supported Unit Formats:

- Hours: H, h, HR, hr, HOUR, hour (with optional plurals)
- Minutes: M, m, MIN, min, MINUTE, minute (with optional plurals)
- Days: D, d, DAY, day (with optional plurals)
- Flexible whitespace and optional "Post-dose", "POST", "After last" suffixes
- Hyphens in compound terms: "PRE-DOSE", "POST-INF", "POST-INFUSION"

Understanding POST/AFTER Patterns:

It's important to distinguish between patterns relative to treatment **start** versus treatment **end**:

- **Relative to START** (treatment_duration NOT added):
  - "1H POST", "1H AFTER", "30M POST" -> Time from dose/treatment start
  - These patterns assume treatment starts at time 0
  - Example: "1H POST" -> 1 hour (regardless of treatment_duration)
- **Relative to END** (treatment_duration IS added):
  - "1H POST EOI", "1H AFTER EOT", "1H POST INFUSION" -> Time from treatment end
  - These patterns account for when treatment ends (start + duration)
  - Example: "1H POST EOI" with treatment_duration=2 -> 3 hours (2 + 1)

This distinction follows standard pharmacokinetic conventions where "post-dose" refers to time from treatment initiation, while "post end of infusion" refers to time from treatment completion.

Vectorized Treatment Duration:

When `treatment_duration` is a vector, each timepoint uses its corresponding treatment duration value. This is useful when different records have different treatment durations (e.g., different infusion lengths).

Value

A numeric vector of timepoints in hours. Returns `NA_real_` for:

- Input `NA` values
- Unrecognized timepoint formats
- Non-time descriptors (e.g., "Morning", "Evening")

Returns `numeric(0)` for empty input.

See Also

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

`country_code_lookup` *Country Code Lookup*

Description

These pre-defined country codes are sourced from ISO 3166 Standards. See also Wikipedia.

Usage

```
country_code_lookup
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 249 rows and 3 columns.

Details

`country_code` is the 3-letter ISO 3166-1 county code commonly found in the ADSL `COUNTRY` variable. `country_name` is the country long name corresponding to to the 3-letter code. `country_number` is the numeric code corresponding to an alphabetic sorting of the 3-letter codes.

To see the entire table in the console, run `print(country_code_lookup)`.

See Also

dose_freq_lookup

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `dose_freq_lookup`

Examples

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

# Create reference dataset for periods
adsl <- tribble(
  ~USUBJID, ~SEX, ~COUNTRY,
  "ST01-01", "F", "AUT",
  "ST01-02", "M", "MWI",
  "ST01-03", "F", "GBR",
  "ST01-04", "M", "CHE",
  "ST01-05", "M", "NOR",
  "ST01-06", "F", "JPN",
  "ST01-07", "F", "USA"
)

adsl %>%
  derive_vars_merged(
    dataset_add = country_code_lookup,
    new_vars = exprs(COUNTRYN = country_number, COUNTRYL = country_name),
    by_vars = exprs(COUNTRY = country_code)
  )
```

---

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

See Also

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

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
  "2",      2,        "PR",
  "2",      3,        "CR",
  "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_period_dataset *Create a Reference Dataset for Subperiods, Periods, or Phases*

Description

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

Usage

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

Arguments

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

`new_vars` New variables

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

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

Only one type must be used, e.g., all right hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables. If period *and* subperiod variables are required, separate reference datasets must be created.

`subject_keys` Variables to uniquely identify a subject

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

Details

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

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

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

Value

A period reference dataset (see "Details" section)

See Also

`derive_vars_period()`

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

Examples

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

# Create reference dataset for periods
adsl <- tribble(
  ~USUBJID, ~AP01SDT,     ~AP01EDT,     ~AP02SDT,     ~AP02EDT,     ~TRT01A, ~TRT02A,
  "1",      "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07", "A",     "B",
  "2",      "2021-02-02", "2021-03-02", "2021-03-03", "2021-04-01", "B",     "A",
) %>%
  mutate(
    across(matches("AP\\d\\d[ES]DT"), ymd)
  ) %>%
```

```
  mutate(
    STUDYID = "xyz"
  )

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

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

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

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

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

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

Description

Creates a queries dataset as input dataset to the dataset_queries argument in the derive_vars_query() function as defined in the vignette("queries_dataset").

Usage

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

Arguments

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

Details

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

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

The following variables (as described in vignette("queries_dataset") are created:

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

Value

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

See Also

derive_vars_query(), query(), basket_select(), vignette("queries_dataset")

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

Examples

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

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

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

create_query_data(queries = list(cq))

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

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

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

# create a query dataset for SDGs
sdg <- query(
```

```
  prefix = "SDG01",
  id = auto,
  definition = basket_select(
    name = "5-aminosalicylates for ulcerative colitis",
    scope = NA_character_,
    type = "sdg"
  )
)

# The get_terms function from pharmaversesdtm is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(sdg),
  get_terms_fun = pharmaversesdtm:::get_terms,
  version = "2019-09"
)

# creating a query dataset for a customized query including SMQs
# The get_terms function from pharmaversesdtm is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(
    query(
      prefix = "CQ03",
      name = "Special issues of interest",
      definition = list(
        basket_select(
          name = "Pregnancy and neonatal topics (SMQ)",
          scope = "NARROW",
          type = "smq"
        ),
        cqterms
      )
    )
  ),
  get_terms_fun = pharmaversesdtm:::get_terms,
  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 = NULL,
  end_date = AENDT,
  end_datetime = NULL,
  lookup_table = dose_freq_lookup,
  lookup_column = CDISC_VALUE,
  nominal_time = NULL,
 keep_source_vars = expr_c(get_admiral_option("subject_keys"), dose_freq, start_date,
    start_datetime, end_date, end_datetime)
)
```

Arguments

dataset	Input dataset The variables specified by the dose_freq, start_date, and end_date arguments are expected to be in the dataset.
dose_freq	The dose frequency The aggregate dosing frequency used for multiple doses in a row.
start_date	The start date A date object is expected. This object cannot contain NA values. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
start_datetime	The start date-time A date-time object is expected. This object cannot contain NA values. Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object. If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.
end_date	The end date A date or date-time object is expected. This object cannot contain NA values. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
end_datetime	The end date-time A date-time object is expected. This object cannot contain NA values. Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object. If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.
lookup_table	The dose frequency value lookup table The table used to look up dose_freq values and determine the appropriate 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.

Permitted Values for DOSE_WINDOW: "MINUTE", "HOUR", "DAY", "WEEK", "MONTH", "YEAR"

lookup_column — The dose frequency value column in the lookup table

The column of lookup_table.

nominal_time — The nominal relative time from first dose (NFRLT)

Used for PK analysis, this will be in hours and should be 0 for the first dose. It can be derived as (VISITDY - 1) * 24 for example. This will be expanded as the single dose dataset is created. For example an EXDOFRQ of "QD" will result in the nominal_time being incremented by 24 hours for each expanded record.

The value can be NULL if not needed.

keep_source_vars

List of variables to be retained from source dataset

This parameter can be specified if additional information is required in the output dataset. For example EXTRT for studies with more than one drug.

Details

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

Observations with dose frequency "ONCE" are copied to the output dataset unchanged.

Value

The input dataset with a single dose per row.

See Also

Creating auxiliary datasets: consolidate_metadata(), create_period_dataset(), create_query_data()

Examples

```
# Example with default lookup

library(lubridate)
library(stringr)
library(tibble)
library(dplyr)

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "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"),
  "STUDY01", "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"),
```

```
  "STUDY01", "P01", "EVERY 2 WEEKS", ymd("2021-01-15"), ymd_hms("2021-01-15 09:57:00"),
  ymd("2021-01-29"), ymd_hms("2021-01-29 10:57:00")
)

create_single_dose_dataset(data)

# Example with custom lookup

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

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "P01", "Q30MIN", ymd("2021-01-01"), ymd_hms("2021-01-01T06:00:00"),
  ymd("2021-01-01"), ymd_hms("2021-01-01T07:00:00"),
  "STUDY02", "P02", "Q90MIN", ymd("2021-01-01"), ymd_hms("2021-01-01T06:00:00"),
  ymd("2021-01-01"), ymd_hms("2021-01-01T09:00:00")
)

create_single_dose_dataset(data,
  lookup_table = custom_lookup,
  lookup_column = Value,
  start_datetime = ASTDTM,
  end_datetime = AENDTM
)
# Example with nominal time

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~NFRLT, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "P01", "BID", 0, ymd("2021-01-01"), ymd_hms("2021-01-01 08:00:00"),
  ymd("2021-01-07"), ymd_hms("2021-01-07 20:00:00"),
  "STUDY01", "P01", "BID", 168, ymd("2021-01-08"), ymd_hms("2021-01-08 08:00:00"),
  ymd("2021-01-14"), ymd_hms("2021-01-14 20:00:00"),
  "STUDY01", "P01", "BID", 336, ymd("2021-01-15"), ymd_hms("2021-01-15 08:00:00"),
  ymd("2021-01-29"), ymd_hms("2021-01-29 20:00:00")
)

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

```
# Example - derive a single dose dataset with imputations

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

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

ex <- tribble(
  ~STUDYID, ~USUBJID, ~EXSEQ, ~EXTRT, ~EXDOSE, ~EXDOSU, ~EXDOSFRQ, ~EXSTDTC, ~EXENDTC,
  "01", "1015", 1, "PLAC", 0, "mg", "QD", "2014-01-02", "2014-01-16",
  "01", "1015", 2, "PLAC", 0, "mg", "QD", "2014-06-17", "2014-06-18",
  "01", "1015", 3, "PLAC", 0, "mg", "QD", "2014-06-19", NA_character_,
  "01", "1023", 1, "PLAC", 0, "mg", "QD", "2012-08-05", "2012-08-27",
  "01", "1023", 2, "PLAC", 0, "mg", "QD", "2012-08-28", "2012-09-01",
  "01", "1211", 1, "XANO", 54, "mg", "QD", "2012-11-15", "2012-11-28",
  "01", "1211", 2, "XANO", 54, "mg", "QD", "2012-11-29", NA_character_,
  "01", "1445", 1, "PLAC", 0, "mg", "QD", "2014-05-11", "2014-05-25",
  "01", "1445", 2, "PLAC", 0, "mg", "QD", "2014-05-26", "2014-11-01",
  "01", "1083", 1, "PLAC", 0, "mg", "QD", "2013-07-22", "2013-08-01"
)

adsl_death <- adsl %>%
  mutate(
    DTHDTM = convert_date_to_dtm(DTHDT),
    # Remove `DCUT` setup line below if ADSL `DCUTDT` is populated.
    DCUTDT = convert_dtc_to_dt("2015-03-06"), # Example only, enter date.
    DCUTDTM = convert_date_to_dtm(DCUTDT)
  )

# Select valid dose records, non-missing `EXSTDTC` and `EXDOSE`.
ex_mod <- ex %>%
  filter(!is.na(EXSTDTC) & !is.na(EXDOSE)) %>%
  derive_vars_merged(adsl_death, by_vars = get_admiral_option("subject_keys")) %>%
```

```
  # Example, set up missing `EXDOSFRQ` as QD daily dosing regime.
  # Replace with study dosing regime per trial treatment.
  mutate(EXDOSFRQ = if_else(is.na(EXDOSFRQ), "QD", EXDOSFRQ)) %>%
  # Create EXxxDTM variables and replace missing `EXENDTM`.
  derive_vars_dtm(
    dtc = EXSTDTC,
    new_vars_prefix = "EXST",
    date_imputation = "first",
    time_imputation = "first",
    flag_imputation = "none",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXSTDTM)) %>%
  derive_vars_dtm(
    dtc = EXENDTC,
    new_vars_prefix = "EXEN",
    # Maximum imputed treatment end date must not be not greater than
    # date of death or after the datacut date.
    max_dates = exprs(DTHDTM, DCUTDTM),
    date_imputation = "last",
    time_imputation = "last",
    flag_imputation = "none",
    highest_imputation = "Y",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXENDTM)) %>%
  # Select only unique values.
  # Removes duplicated records before final step.
  distinct(
    STUDYID, USUBJID, EXTRT, EXDOSE, EXDOSFRQ, DCUTDT, DTHDT, EXSTDT,
    EXSTDTM, EXENDT, EXENDTM, EXSTDTC, EXENDTC
  )

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

`date_source` *Create a* `date_source` *object*

Description

[Deprecated] The `date_source()` function has been deprecated in favor of `event()`.

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, set_values_to = NULL)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the date.
filter	An unquoted condition for filtering dataset.
date	A variable or an expression providing a date. A date or a datetime can be specified. An unquoted symbol or expression is expected.
set_values_to	Variables to be set

Value

An object of class date_source.

See Also

derive_var_extreme_dtm(), derive_var_extreme_dt()

Other deprecated: call_user_fun(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

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

# lab date from LB where assessment was taken, i.e. not "NOT DONE"
lb_date <- date_source(
  dataset_name = "lb",
  filter = LBSTAT != "NOT DONE" | is.na(LBSTAT),
  date = convert_dtc_to_dt(LBDTC)
)

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

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

Description

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

Usage

```
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 Objects: basket_select(), censor_source(), event(), event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

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

Value

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

See Also

`derive_param_qtc()`

BDS-Findings Functions for adding Parameters/Records: `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_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 = NULL)
```

Arguments

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

Value

An object of class derivation_slice

See Also

slice_derivation(), params()

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

derive_basetype_records *Derive Basetype Variable*

Description

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

Usage

```
derive_basetype_records(dataset, basetypes)
```

Arguments

dataset	Input dataset The variables specified by the basetypes argument are expected to be in the dataset.
basetypes	A *named* list of expressions created using the rlang::exprs() function The names corresponds to the values of the newly created BASETYPE variables and the expressions are used to subset the input dataset.

Details

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

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

Value

The input dataset with variable BASETYPE added

See Also

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

Examples

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

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

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

```
    "DOUBLE-BLIND" = EPOCH %in% c("DOUBLE-BLIND", "OPEN-LABEL"),
    "OPEN-LABEL" = EPOCH == "OPEN-LABEL"
  )
)


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

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

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

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

print(bds_with_basetype, n = Inf)

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

`derive_expected_records` *Derive Expected Records*

Description

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

Usage

```
derive_expected_records(
  dataset,
  dataset_ref,
  by_vars = NULL,
  set_values_to = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the dataset_ref and by_vars arguments are expected to be in the dataset.
dataset_ref	Expected observations dataset Data frame with the expected observations, e.g., all the expected combinations of PARAMCD, PARAM, AVISIT, AVISITN, ...
by_vars	Grouping variables For each group defined by by_vars those observations from dataset_ref are added to the output dataset which do not have a corresponding observation in the input dataset.
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. A list of variable name-value pairs is expected. • LHS refers to a variable. • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, NA, or expressions, e.g., exprs(PARAMCD = "TDOSE", PARCAT1 = "OVERALL").

Details

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

Value

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

See Also

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

Examples

```
library(tibble)

adqs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVISITN, ~AVISIT, ~AVAL,
  "1",      "a",             1, "WEEK 1",   10,
  "1",      "b",             1, "WEEK 1",   11,
  "2",      "a",             2, "WEEK 2",   12,
  "2",      "b",             2, "WEEK 2",   14
```

```
)

# Example 1. visit variables are parameter independent
parm_visit_ref <- tribble(
  ~AVISITN, ~AVISIT,
  1,        "WEEK 1",
  2,        "WEEK 2"
)

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

# Example 2. visit variables are parameter dependent
parm_visit_ref <- tribble(
  ~PARAMCD, ~AVISITN, ~AVISIT,
  "a",             1, "WEEK 1",
  "a",             2, "WEEK 2",
  "b",             1, "WEEK 1"
)

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

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

Description

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

Usage

```
derive_extreme_event(
  dataset = NULL,
  by_vars,
  events,
  tmp_event_nr_var = NULL,
```

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

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset.
by_vars	Grouping variables
events	Conditions and new values defining events A list of event() or event_joined() objects is expected. Only observations listed in the events are considered for deriving extreme event. If multiple records meet the filter condition, take the first record sorted by order. The data is grouped by by_vars, i.e., summary functions like all() or any() can be used in condition. For event_joined() events the observations are selected by calling filter_joined(). The condition field is passed to the filter_join argument.
tmp_event_nr_var	Temporary event number variable The specified variable is added to all source datasets and is set to the number of the event before selecting the records of the event. It can be used in order to determine which record should be used if records from more than one event are selected. The variable is not included in the output dataset.
order	Sort order If a particular event from events has more than one observation, within the event and by group, the records are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
mode	Selection mode (first or last) If a particular event from events has more than one observation, "first"/"last" is used to select the first/last record of this type of event sorting by order.
source_datasets	Source datasets A named list of datasets is expected. The dataset_name field of event() and event_joined() refers to the dataset provided in the list.
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.

`set_values_to` Variables to be set

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

Set a list of variables to some specified value for the new records

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA.

For example:

```
set_values_to = exprs(
  PARAMCD = "WOBS",
  PARAM = "Worst Observations"
)
```

`keep_source_vars`

Variables to keep from the source dataset

For each event the specified variables are kept from the selected observations. The variables specified for `by_vars` and created by `set_values_to` are always kept. The `keep_source_vars` field of the event will take precedence over the value of the `keep_source_vars` argument.

Details

1. For each event select the observations to consider:
   (a) If the event is of class `event`, the observations of the source dataset are restricted by `condition` and then the first or last (`mode`) observation per by group (`by_vars`) is selected.
   If the event is of class `event_joined`, `filter_joined()` is called to select the observations.
   (b) The variables specified by the `set_values_to` field of the event are added to the selected observations.
   (c) The variable specified for `tmp_event_nr_var` is added and set to the number of the event.
   (d) Only the variables specified for the `keep_source_vars` field of the event, and the by variables (`by_vars`) and the variables created by `set_values_to` are kept. If `keep_source_vars = NULL` is used for an event in `derive_extreme_event()` the value of the `keep_source_vars` argument of `derive_extreme_event()` is used.
2. All selected observations are bound together.
3. 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.
4. The variables specified by the `set_values_to` parameter are added to the selected observations.
5. The observations are added to input dataset.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

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

See Also

event(), event_joined(), derive_vars_extreme_event()

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

---

derive_extreme_records

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

---

Description

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

Usage

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

Arguments

dataset	Input dataset If the argument is not specified (or set to NULL), a new dataset is created. Otherwise, the new records are appended to the specified dataset.

dataset_add
: Additional dataset

  The additional dataset, which determines the by groups returned in the input dataset, based on the groups that exist in this dataset after being subset by filter_add.

  The variables specified in the by_vars and filter_add parameters are expected in this dataset. If mode and order are specified, the first or last observation within each by group, defined by by_vars, is selected.

dataset_ref
: Reference dataset

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

  For records which are added from dataset_ref because there are no records in dataset_add for the by group only those variables are kept which are also in dataset_add (and are included in keep_source_vars).

by_vars
: Grouping variables

  If dataset_ref is specified, this argument must be specified.

order
: Sort order

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

mode
: Selection mode (first or last)

  If "first" is specified, the first observation of each by group is added to the input dataset. If "last" is specified, the last observation of each by group is added to the input dataset.

filter_add
: Filter for additional dataset (dataset_add)

  Only observations in dataset_add fulfilling the specified condition are considered.

check_type
: Check uniqueness?

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

exist_flag
: Existence flag

  The specified variable is added to the output dataset.

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

  For all other by groups exist_flag is set to the value specified by the false_value argument.

true_value
: True value

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

false_value
: False value

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

keep_source_vars
: Variables to be kept in the new records

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

`set_values_to` Variables to be set

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

Set a list of variables to some specified value for the new records

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for `by_vars`. Any expression on the RHS must result in a single value per by group.

For example:

```
set_values_to = exprs(
  AVAL = sum(AVAL),
  DTYPE = "AVERAGE",
)
```

Details

1. The additional dataset (`dataset_add`) is restricted as specified by the `filter_add` argument.
2. For each group (with respect to the variables specified for the `by_vars` argument) the first or last observation (with respect to the order specified for the `order` argument and the mode specified for the `mode` argument) is selected.
3. If `dataset_ref` is specified, observations which are in `dataset_ref` but not in the selected records are added. Variables that are common across `dataset_ref`, `dataset_add` and `keep_source_vars()` are also populated for the new observations.
4. The variables specified by the `set_values_to` argument are added to the selected observations.
5. The variables specified by the `keep_source_vars` argument are selected along with the variables specified in `by_vars` and `set_values_to` arguments.
6. The observations are added to input dataset (`dataset`). If no input dataset is provided, a new dataset is created.

Value

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

See Also

`derive_summary_records()`

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

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

Description

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

Usage

```
derive_locf_records(
  dataset,
  dataset_ref,
  by_vars,
  id_vars_ref = NULL,
  analysis_var = AVAL,
  imputation = "add",
  order,
  keep_vars = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars`, `analysis_var`, `order`, and `keep_vars` arguments are expected to be in the dataset.
`dataset_ref`	Expected observations dataset Data frame with all the combinations of `PARAMCD`, `PARAM`, `AVISIT`, `AVISITN`, ... which are expected in the dataset is expected.
`by_vars`	Grouping variables For each group defined by `by_vars` those observations from `dataset_ref` are added to the output dataset which do not have a corresponding observation in the input dataset or for which `analysis_var` is `NA` for the corresponding observation in the input dataset.
`id_vars_ref`	Grouping variables in expected observations dataset The variables to group by in `dataset_ref` when determining which observations should be added to the input dataset.
`analysis_var`	Analysis variable.
`imputation`	Select the mode of imputation: `add`: Keep all original records and add imputed records for missing timepoints and missing `analysis_var` values from `dataset_ref`. `update`: Update records with missing `analysis_var` and add imputed records for missing timepoints from `dataset_ref`. `update_add`: Keep all original records, update records with missing `analysis_var` and add imputed records for missing timepoints from `dataset_ref`.

order	Sort order The dataset is sorted by order before carrying the last observation forward (e.g. AVAL) within each by_vars. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
keep_vars	Variables that need carrying the last observation forward Keep variables that need carrying the last observation forward other than analysis_var (e.g., PARAMN, VISITNUM). If by default NULL, only variables specified in by_vars and analysis_var will be populated in the newly created records.

Details

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

- which do not have a corresponding observation in the input dataset or
- for which analysis_var is NA for the corresponding observation in the input dataset.
  For the new observations, analysis_var is set to the non-missing analysis_var of the previous observation in the input dataset (when sorted by order) and DTYPE is set to "LOCF".
  The imputation argument decides whether to update the existing observation when analysis_var is NA ("update" and "update_add"), or to add a new observation from dataset_ref instead ("add").

Value

The input dataset with the new "LOCF" observations added for each by_vars, based on the value passed to the imputation argument.

Author(s)

G Gayatri

See Also

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

---

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.

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

Usage

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

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by weight_code and height_code.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.
weight_code	WEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the WEIGHT. It is expected that WEIGHT is measured in kg
height_code	HEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the HEIGHT. It is expected that HEIGHT is measured in cm
get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters.
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.
constant_by_vars	By variables for when HEIGHT is constant When HEIGHT is constant, the HEIGHT parameters (measured only once) are merged to the other parameters using the specified variables. If height is constant (e.g. only measured once at screening or baseline) then use constant_by_vars to select the subject-level variable to merge on (e.g.

USUBJID). This will produce BMI at all visits where weight is measured. Otherwise it will only be calculated at visits with both height and weight collected.

Details

The analysis value of the new parameter is derived as

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

Value

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

See Also

compute_bmi()

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

Examples

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

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

```
# Example 2: Derive BMI where height is measured only once and keep only one record
# where both height and weight are measured.
derive_param_bmi(
  advs,
  by_vars = exprs(USUBJID, AVISIT),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = exprs(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)

# Example 3: Pediatric study where height and weight are measured multiple times
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-101-1001", "HEIGHT", "Height (cm)", 47.1, "BASELINE",
  "01-101-1001", "HEIGHT", "Height (cm)", 59.1, "WEEK 12",
  "01-101-1001", "HEIGHT", "Height (cm)", 64.7, "WEEK 24",
  "01-101-1001", "HEIGHT", "Height (cm)", 68.2, "WEEK 48",
  "01-101-1001", "WEIGHT", "Weight (kg)", 2.6, "BASELINE",
  "01-101-1001", "WEIGHT", "Weight (kg)", 5.3, "WEEK 12",
  "01-101-1001", "WEIGHT", "Weight (kg)", 6.7, "WEEK 24",
  "01-101-1001", "WEIGHT", "Weight (kg)", 7.4, "WEEK 48",
)

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

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

Description

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

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

Usage

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

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by HEIGHT and WEIGHT.
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.
method	Derivation method to use. Note that HEIGHT is expected in cm and WEIGHT is expected in kg: Mosteller: sqrt(height * weight / 3600) DuBois-DuBois: 0.20247 * (height/100) ^ 0.725 * weight ^ 0.425 Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378 Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456 Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight))) Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444 Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.
height_code	HEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the HEIGHT assessments. It is expected that HEIGHT is measured in cm.
weight_code	WEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the WEIGHT assessments. It is expected that WEIGHT is measured in kg.

`get_unit_expr` An expression providing the unit of the parameter

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

`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.

`constant_by_vars`

By variables for when HEIGHT is constant

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

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

Value

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

See Also

`compute_bsa()`

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

Examples

```
library(tibble)

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

```
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
  ),
  get_unit_expr = extract_unit(PARAM),
  constant_by_vars = exprs(USUBJID)
)

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

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

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

# Example 3: Pediatric study where height and weight are measured multiple times
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-101-1001", "HEIGHT", "Height (cm)", 47.1, "BASELINE",
  "01-101-1001", "HEIGHT", "Height (cm)", 59.1, "WEEK 12",
  "01-101-1001", "HEIGHT", "Height (cm)", 64.7, "WEEK 24",
  "01-101-1001", "HEIGHT", "Height (cm)", 68.2, "WEEK 48",
  "01-101-1001", "WEIGHT", "Weight (kg)", 2.6, "BASELINE",
  "01-101-1001", "WEIGHT", "Weight (kg)", 5.3, "WEEK 12",
  "01-101-1001", "WEIGHT", "Weight (kg)", 6.7, "WEEK 24",
  "01-101-1001", "WEIGHT", "Weight (kg)", 7.4, "WEEK 48",
)
derive_param_bsa(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  method = "Mosteller",
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
```

```
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

`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, such as addition/sum, subtraction/difference, multiplication/product, division/ratio, exponentiation/logarithm, or by formula.

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 = NULL,
  dataset_add = NULL,
  by_vars,
  parameters,
  set_values_to,
  filter = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL,
  keep_nas = FALSE
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD` is expected as well. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `parameters`.
`dataset_add`	Additional dataset The variables specified by the `by_vars` parameter are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the additional dataset after restricting it to the parameters specified by `parameters`.

If the argument is specified, the observations of the additional dataset are considered in addition to the observations from the input dataset (`dataset` restricted by `filter`).

`by_vars` Grouping variables

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

`parameters` Required parameter codes

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

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

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

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. The values of variables of the parameters specified by `parameters` can be accessed using `<variable name>.<parameter code>`. For example

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

defines the analysis value and parameter code for the new parameter.

Variable names in the expression must not contain more than one dot.

Note that `dplyr` helper functions such as `dplyr::starts_with()` should be avoided unless the list of variable-value pairs is clearly specified in a statement via the `set_values_to` argument.

`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.

`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)

`constant_parameters`

Required constant parameter codes

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

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

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

keep_nas — Keep observations with NAs

If the argument is set to TRUE, observations are added even if some of the values contributing to the computed value are NA (see Example 1b).

If the argument is set to a list of variables, observations are added even if some of specified variables are NA (see Example 1c).

Details

For each group (with respect to the variables specified for the by_vars parameter) an observation is added to the output dataset if the filtered input dataset (dataset) or the additional dataset (dataset_add) contains exactly one observation for each parameter code specified for parameters and all contributing values like AVAL.SYSBP are not NA. The keep_nas can be used to specify variables for which NAs are acceptable. See also Example 1b and 1c.

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

Value

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

See Also

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

`derive_param_doseint` *Adds a Parameter for Dose Intensity*

Description

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

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

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

Usage

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

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well. 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.
`set_values_to`	Variables to be set The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.
`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.
`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.

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

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.

See Also

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

Examples

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

adex <- tribble(
  ~USUBJID, ~PARAMCD, ~VISIT, ~ANL01FL, ~ASTDT, ~AENDT, ~AVAL,
  "P001", "TNDOSE", "V1", "Y", ymd("2020-01-01"), ymd("2020-01-30"), 59,
  "P001", "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 = exprs(USUBJID, VISIT),
  set_values_to = exprs(PARAMCD = "TNDOSINT"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE"
)

derive_param_doseint(
  adex,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(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 measurable disease at baseline.

Usage

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

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD` is expected as well.
`dataset_ref`	Reference dataset, e.g., ADSL The variables specified in `by_vars` are expected. For each group (as defined by `by_vars`) from the specified dataset (`dataset_ref`), the existence flag is calculated and added as a new observation to the input datasets (`dataset`).
`dataset_add`	Additional dataset The variables specified by the `by_vars` parameter are expected. This dataset is used to check if an event occurred or not. Any observation in the dataset fulfilling the event condition (`condition`) is considered as an event.
`condition`	Event condition The condition is evaluated at the additional dataset (`dataset_add`). For all groups where it evaluates as `TRUE` at least once `AVALC` is set to the true value (`true_value`) for the new observations.

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

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

true_value — True value

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

false_value — False value

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

missing_value — Values used for missing information

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

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.

by_vars — Grouping variables

set_values_to — Variables to set

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

Details

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

Value

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

See Also

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

Examples

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

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

tu <- 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_ref = adsl,
  dataset_add = tu,
  filter_add = TUTESTCD == "TUMIDENT" & VISIT == "SCREENING",
  condition = TUSTRESC == "TARGET",
  false_value = "N",
  missing_value = "N",
  set_values_to = exprs(
    AVAL = yn_to_numeric(AVALC),
    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 = NULL,
  dataset_add,
  by_vars,
  input_code,
  filter_add = NULL,
  set_values_to = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Additional dataset The variables specified for by_vars, analysis_var, PARAMCD, alongside either ASTDTM and AENDTM or ASTDT and AENDT are also expected. Observations from the specified dataset are going to be used to calculate and added as new records to the input dataset (dataset).
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.
input_code	Required parameter code The observations where PARAMCD equals the specified value are considered to compute the summary record.
filter_add	Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved. For example, • filter_add = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars. • filter_add = (dplyr::n() > 2) will filter n count of by_vars greater than 2.

set_values_to Variable-value pairs

Set a list of variables to some specified value for the new observation(s)

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

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 (as defined by 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,

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

If the input datasets contains

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

See Also

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

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(stringr, warn.conflicts = FALSE)
adex <- 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(
    dataset_add = adex,
    by_vars = exprs(USUBJID),
    set_values_to = exprs(
      PARAMCD = "TDOSE",
      PARCAT1 = "OVERALL",
      AVAL = sum(AVAL, na.rm = TRUE)
    ),
    input_code = "DOSE"
  ) %>%
  select(-ASTDTM, -AENDTM)

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

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

derive_param_extreme_record *Adds a Parameter Based on First or Last Record from Multiple Sources*

Description

[Deprecated] The `derive_param_extreme_record()` function has been deprecated in favor of `derive_extreme_event()`.

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

Usage

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

Arguments

`dataset`	Input dataset
`sources`	Sources A list of `records_source()` objects is expected.
`source_datasets`	Source datasets A named list of datasets is expected. The `dataset_name` field of `records_source()` refers to the dataset provided in the list. The variables specified by the `order` and the `by_vars` arguments are expected after applying `new_vars`.
`by_vars`	Grouping variables If the argument is specified, for each by group the observations are selected separately.
`order`	Sort order If the argument is set to a non-null value, for each by group the first or last observation from the source datasets is selected with respect to the specified order. Variables created via `new_vars` e.g., imputed date variables, can be specified as well (see examples below). Please note that `NA` is considered as the last value. I.e., if a order variable is `NA` and `mode = "last"`, this observation is chosen while for `mode = "first"` the observation is chosen only if there are no observations where the variable is not `NA`.

mode — Selection mode (first or last)

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

set_values_to — Variables to be set

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

A list of variable name-value pairs is expected.

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

Details

The following steps are performed to create the output dataset:

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

Value

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

See Also

Other deprecated: call_user_fun(), date_source(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

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

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

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

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

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

Description

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

Usage

```
derive_param_framingham(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "FCVD101"),
  sysbp_code = "SYSBP",
  chol_code = "CHOL",
```

```
  cholhdl_code = "CHOLHDL",
  age = AGE,
  sex = SEX,
  smokefl = SMOKEFL,
  diabetfl = DIABETFL,
  trthypfl = TRTHYPFL,
  get_unit_expr,
  filter = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well. 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 Only variables specified in by_vars will be populated in the newly created records.
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.
sysbp_code	Systolic blood pressure parameter code The observations where PARAMCD equals the specified value are considered as the systolic blood pressure assessments.
chol_code	Total serum cholesterol code The observations where PARAMCD equals the specified value are considered as the total cholesterol assessments. This must be measured in mg/dL.
cholhdl_code	HDL serum cholesterol code The observations where PARAMCD equals the specified value are considered as the HDL cholesterol assessments. This must be measured in mg/dL.
age	Subject age A variable containing the subject's age.
sex	Subject sex A variable containing the subject's sex.
smokefl	Smoking status flag A flag indicating smoking status.
diabetfl	Diabetic flag A flag indicating diabetic status.
trthypfl	Treated with hypertension medication flag A flag indicating if a subject was treated with hypertension medication.

`get_unit_expr` An expression providing the unit of the parameter

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

`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.

Details

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

For Women:

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

See Also

compute_framingham()

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

Examples

```
library(tibble)

adcvrisk <- tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU,
  ~VISIT, ~AGE, ~SEX, ~SMOKEFL, ~DIABETFL, ~TRTHYPFL,
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121,
  "mmHg", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 115,
  "mmHg", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 216.16,
  "mg/dL", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 210.78,
  "mg/dL", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1015", "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 = exprs(USUBJID, VISIT),
    set_values_to = exprs(
      PARAMCD = "FCVD101",
      PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
    ),
    get_unit_expr = AVALU
  )

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

`derive_param_map` *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.

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

Usage

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

Arguments

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

by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.
sysbp_code	Systolic blood pressure parameter code The observations where PARAMCD equals the specified value are considered as the systolic blood pressure assessments.
diabp_code	Diastolic blood pressure parameter code The observations where PARAMCD equals the specified value are considered as the diastolic blood pressure assessments.
hr_code	Heart rate parameter code The observations where PARAMCD equals the specified value are considered as the heart rate assessments.
get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters.
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.

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.

See Also

compute_map()

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

Examples

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

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

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

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

---

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

Description

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

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

Usage

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

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `get_unit_expr` arguments are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well. 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` argument) 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.
`method`	Method used to QT correction See `compute_qtc()` for details.
`set_values_to`	Variables to be set The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.
`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 ms or msec.
`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 ms or msec.
`get_unit_expr`	An expression providing the unit of the parameter The result is used to check the units of the input parameters.
`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.

Value

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

See Also

compute_qtc()

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

Examples

```
library(tibble)

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

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

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

```
  get_unit_expr = extract_unit(PARAM)
)

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

`derive_param_rr` *Adds a Parameter for Derived RR (an ECG measurement)*

Description

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

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

The analysis value of the new parameter is derived as

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

Usage

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

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well. 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` argument) and to the parameters specified by `hr_code`.

by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.
hr_code	HR parameter code The observations where PARAMCD equals the specified value are considered as the heart rate assessments.
get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters.
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.

Value

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

See Also

compute_rr()

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

Examples

```
library(tibble)

adeg <- 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, "ms", "WEEK 2",
  "01-701-1015", "HR", "Heart Rate", 62.66, "beats/min", "WEEK 1",
  "01-701-1015", "RR", "RR Duration", 710, "ms", "WEEK 2",
  "01-701-1028", "HR", "Heart Rate", 85.45, "beats/min", "BASELINE",
  "01-701-1028", "QT", "QT Duration", 480, "ms", "WEEK 2",
  "01-701-1028", "QT", "QT Duration", 350, "ms", "WEEK 3",
  "01-701-1028", "HR", "Heart Rate", 56.54, "beats/min", "WEEK 3",
  "01-701-1028", "RR", "RR Duration", 842, "ms", "WEEK 2"
)
```

```
derive_param_rr(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "RRR",
    PARAM = "RR Duration Rederived (ms)",
    AVALU = "ms"
  ),
  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 = get_admiral_option("subject_keys"),
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset `PARAMCD` is expected.
`dataset_adsl`	ADSL input dataset The variables specified for `start_date`, and `subject_keys` are expected.
`source_datasets`	Source datasets A named list of datasets is expected. The `dataset_name` field of `tte_source()` refers to the dataset provided in the list.

by_vars
: By variables

  If the parameter is specified, separate time to event parameters are derived for each by group.

  The by variables must be in at least one of the source datasets. Each source dataset must contain either all by variables or none of the by variables.

  The by variables are not included in the output dataset.

start_date
: Time to event origin date

  The variable STARTDT is set to the specified date. The value is taken from the ADSL dataset.

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

event_conditions
: Sources and conditions defining events

  A list of event_source() objects is expected.

censor_conditions
: Sources and conditions defining censorings

  A list of censor_source() objects is expected.

create_datetime
: Create datetime variables?

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

set_values_to
: Variables to set

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

subject_keys
: Variables to uniquely identify a subject

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

check_type
: Check uniqueness

  If "warning", "message", or "error" is specified, the specified message is issued if the observations of the source datasets are not unique with respect to the by variables and the date and order specified in the event_source() and censor_source() objects.

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 subject the first observation (with respect to date and order) 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 subject the first observation (with respect to the ADT/ADTM variable) from the single dataset is selected. If there is more than one event with the same date, the first event with respect to the order of events in event_conditions 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 subject the last observation (with respect to date and order) 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 subject the last observation (with respect to the ADT/ADTM variable) from the single dataset is selected. If there is more than one censoring with the same date, the last censoring with respect to the order of censorings in censor_conditions is selected.

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

Finally:

1. The variable specified for start_date is joined from the ADSL dataset. Only subjects in both datasets are kept, i.e., subjects with both an event or censoring and an observation in dataset_adsl.
2. The variables as defined by the set_values_to parameter are added.
3. The ADT/ADTM variable is set to the maximum of ADT/ADTM and STARTDT/STARTDTM (depending on the create_datetime parameter).
4. The new observations are added to the output dataset.

Value

The input dataset with the new parameter added

See Also

event_source(), censor_source()

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 are expected to be in the dataset. PARAMCD, and AVAL are expected as well. 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
set_values_to	Variables to set A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "LYMPH", PARAM = "Lymphocytes Abs (10^9/L)") is expected.
get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters.
wbc_unit	A string containing the required unit of the WBC parameter
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.
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

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

See Also

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

Examples

```
library(tibble)

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

derive_param_wbc_abs(
  dataset = test_lb,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "LYMPH",
    PARAM = "Lymphocytes Abs (10^9/L)",
    DTYPE = "CALCULATION"
```

```
  ),
  get_unit_expr = extract_unit(PARAM),
  wbc_code = "WBC",
  diff_code = "LYMLE",
  diff_type = "fraction"
)
```

`derive_summary_records` *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, if the derivation deviates from the standard derivation of the parameter.

Usage

```
derive_summary_records(
  dataset = NULL,
  dataset_add,
  dataset_ref = NULL,
  by_vars,
  filter_add = NULL,
  constant_values = NULL,
  set_values_to,
  missing_values = NULL
)
```

Arguments

`dataset`	Input dataset If the argument is not specified (or set to `NULL`), a new dataset is created. Otherwise, the new records are appended to the specified dataset.
`dataset_add`	Additional dataset The variables specified for `by_vars` are expected. Observations from the specified dataset are going to be used to calculate and added as new records to the input dataset (`dataset`).
`dataset_ref`	Reference dataset The variables specified for `by_vars` are expected. For each observation of the specified dataset a new observation is added to the input dataset.
`by_vars`	Grouping variables Variables to consider for generation of groupwise summary records. Providing the names of variables in `exprs()` will create a groupwise summary and generate summary records for the specified groups.

filter_add — Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved.

For example,

- filter_add = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars.
- filter_add = (dplyr::n() > 2) will filter n count of by_vars greater than 2.

constant_values

Constant variables to set

The specified variables are set to the specified values for all new summary records, including those with data in dataset_add and those with no data imputed using dataset_ref and missing_values.

Set a list of variables to some specified value for the new records

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be an expression.

set_values_to — Variables to be set

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

Set a list of variables to some specified value for the new records

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for by_vars. Any expression on the RHS must result in a single value per by group.

For example:

```
set_values_to = exprs(
  AVAL = sum(AVAL),
  DTYPE = "AVERAGE",
)
```

missing_values — Values for missing summary values

For observations of the reference dataset (dataset_ref) which do not have a complete mapping defined by the summarization defined in set_values_to. Only variables specified for set_values_to can be specified for missing_values.

Details

For the newly derived records, only variables specified within by_vars or set_values_to will be populated. All other variables will be set to NA.

Value

A data frame with derived records appended to original dataset.

See Also

derive_vars_merged_summary()

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

---

derive_vars_aage *Derive Analysis Age*

Description

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

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

Usage

```
derive_vars_aage(
  dataset,
  start_date = BRTHDT,
  end_date = RANDDT,
  age_unit = "YEARS",
  type = "interval"
)
```

Arguments

dataset	Input dataset The variables specified by the start_date and end_date arguments are expected to be in the dataset.
start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
end_date	The end date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.
age_unit	Age unit The age is derived in the specified unit
type	lubridate duration type See below for details. Default: "interval" Permitted Values: "duration", "interval"

Details

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

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

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

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

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

Value

The input dataset with `AAGE` and `AAGEU` added

See Also

`derive_vars_duration()`

ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_vars_extreme_event()`, `derive_vars_period()`

Examples

```
library(tibble)
library(lubridate)

data <- tribble(
  ~BRTHDT, ~RANDDT,
  ymd("1984-09-06"), 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`.

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

Usage

```
derive_vars_atc(
  dataset,
  dataset_facm,
  by_vars = exprs(!!!get_admiral_option("subject_keys"), CMREFID = FAREFID),
  id_vars = NULL,
  value_var = FASTRESC
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_facm	FACM dataset The variables specified by the by_vars, id_vars, and value_var arguments and FATESTCD are required. The variables by_vars, id_vars, and FATESTCD must be a unique key.
by_vars	Grouping variables Keys used to merge dataset_facm with dataset.
id_vars	ID variables Variables (excluding by_vars) that uniquely identify each observation in dataset_merge.
value_var	The variable of dataset_facm containing the values of the transposed variables

Value

The input dataset with ATC variables added

See Also

derive_vars_transposed()

OCCDS Functions: derive_var_trtemfl(), derive_vars_query()

Examples

```
library(tibble)

cm <- tribble(
  ~STUDYID,  ~USUBJID,       ~CMGRPID, ~CMREFID,  ~CMDECOD,
  "STUDY01", "BP40257-1001", "14",     "1192056", "PARACETAMOL",
  "STUDY01", "BP40257-1001", "18",     "2007001", "SOLUMEDROL",
  "STUDY01", "BP40257-1002", "19",     "2791596", "SPIRONOLACTONE"
)
facm <- tribble(
  ~STUDYID,  ~USUBJID,       ~FAGRPID, ~FAREFID,  ~FATESTCD,  ~FASTRESC,
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC1CD", "N",
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC2CD", "N02",
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC3CD", "N02B",
```

```
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC4CD", "N02BE",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC1CD", "D",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC2CD", "D10",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC3CD", "D10A",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC4CD", "D10AA",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC1CD", "D",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC2CD", "D07",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC3CD", "D07A",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC4CD", "D07AA",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC1CD", "H",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC2CD", "H02",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC3CD", "H02A",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC4CD", "H02AB",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC1CD", "C",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC2CD", "C03",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC3CD", "C03D",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC4CD", "C03DA"
)

derive_vars_atc(cm, facm, id_vars = exprs(FAGRPID))
```

`derive_vars_cat` *Derive Categorization Variables Like* `AVALCATy` *and* `AVALCAyN`

Description

Derive Categorization Variables Like `AVALCATy` and `AVALCAyN`

Usage

```
derive_vars_cat(dataset, definition, by_vars = NULL)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `definition` arguments are expected to be in the dataset.
`definition`	List of expressions created by `exprs()`. Must be in rectangular format and specified using the same syntax as when creating a `tibble` using the `tribble()` function. The `definition` object will be converted to a `tibble` using `tribble()` inside this function. Must contain: • the column `condition` which will be converted to a logical expression and will be used on the `dataset` input. • at least one additional column with the new column name and the category value(s) used by the logical expression. • the column specified in `by_vars` (if `by_vars` is specified)

e.g. if by_vars is not specified:

```
exprs(~condition,   ~AVALCAT1, ~AVALCA1N,
      AVAL >= 140, ">=140 cm",         1,
      AVAL < 140,   "<140 cm",         2)
```

e.g. if by_vars is specified as exprs(VSTEST):

```
exprs(~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
      "Height", AVAL >= 140, ">=140 cm",         1,
      "Height",  AVAL < 140,  "<140 cm",         2)
```

by_vars — list of expressions with one element. NULL by default. Allows for specifying by groups, e.g. exprs(PARAMCD). Variable must be present in both dataset and definition. The conditions in definition are applied only to those records that match by_vars. The categorization variables are set to NA for records not matching any of the by groups in definition.

Details

If conditions are overlapping, the row order of definitions must be carefully considered. The **first** match will determine the category. i.e. if

AVAL = 155

and the definition is:

```
definition <- exprs(
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height",  AVAL > 170,  ">170 cm",         1,
  "Height", AVAL <= 170, "<=170 cm",         2,
  "Height", AVAL <= 160, "<=160 cm",         3
)
```

then AVALCAT1 will be "<=170 cm", as this is the first match for AVAL. If you specify:

```
definition <- exprs(
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height", AVAL <= 160, "<=160 cm",         3,
  "Height", AVAL <= 170, "<=170 cm",         2,
  "Height",  AVAL > 170,  ">170 cm",         1
)
```

Then AVAL <= 160 will lead to AVALCAT1 == "<=160 cm", AVAL in-between 160 and 170 will lead to AVALCAT1 == "<=170 cm", and AVAL <= 170 will lead to AVALCAT1 == ">170 cm".

However, we suggest to be more explicit when defining the condition, to avoid overlap. In this case, the middle condition should be: AVAL <= 170 & AVAL > 160

Value

The input dataset with the new variables defined in definition added

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

---

derive_vars_computed *Adds Variable(s) Computed from the Analysis Value of one or more Parameters*

Description

Adds Variable(s) computed from the analysis value of one or more parameters. It is expected that the value of the new variable is defined by an expression using the analysis values of other parameters, such as addition/sum, subtraction/difference, multiplication/product, division/ratio, exponentiation/logarithm, or by formula.

For example Body Mass Index at Baseline (BMIBL) in ADSL can be derived from of HEIGHT and WEIGHT parameters in ADVS.

Usage

```
derive_vars_computed(
  dataset,
  dataset_add,
  by_vars,
  parameters,
  new_vars,
  filter_add = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL
)
```

Arguments

dataset	The variables specified by the by_vars parameter are expected.
dataset_add	Additional dataset The variables specified by the by_vars parameter are expected. The variable specified by by_vars and PARAMCD must be a unique key of the additional dataset after restricting it by the filter condition (filter_add parameter) and to the parameters specified by parameters.
by_vars	Grouping variables Grouping variables uniquely identifying a set of records for which new_vars are to be calculated.

parameters

Required parameter codes

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

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

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

new_vars

Name of the newly created variables

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

```
exprs(
  BMIBL = (AVAL.WEIGHT / (AVAL.HEIGHT/100)^2)
)
```

defines the value for the new variable.

Variable names in the expression must not contain more than one dot.

filter_add

Filter condition of additional dataset

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

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 the Example)

constant_parameters

Required constant parameter codes

It is expected that all the parameter codes (PARAMCD) which are required to derive the new variable 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 the Example)

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

and dataset_add), sets PARAMCD = "HGHT" for these observations, and adds them to the observations to consider.

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

Details

For each group (with respect to the variables specified for the by_vars argument), the values of the new variables (new_vars) are computed based on the parameters in the additional dataset (dataset_add) and then the new variables are merged to the input dataset (dataset).

Value

The input dataset with the new variables added.

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

```
library(tibble)
library(dplyr)

# Example 1: Derive BMIBL
adsl <- tribble(
  ~STUDYID,   ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01", "01-1302",   61,   "YEARS",
  "PILOT01", "17-1344",   64,   "YEARS"
)

advs <- tribble(
  ~STUDYID,  ~USUBJID,  ~PARAMCD, ~PARAM,        ~VISIT,      ~AVAL, ~AVALU, ~ABLFL,
  "PILOT01", "01-1302", "HEIGHT", "Height (cm)", "SCREENING", 177.8, "cm",   "Y",
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "SCREENING", 81.19, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "BASELINE",   82.1, "kg",   "Y",
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 2",    81.19, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 4",    82.56, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 6",    80.74, "kg",   NA,
  "PILOT01", "17-1344", "HEIGHT", "Height (cm)", "SCREENING", 163.5, "cm",   "Y",
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "SCREENING", 58.06, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "BASELINE",  58.06, "kg",   "Y",
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 2",    58.97, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 4",    57.97, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 6",    58.97, "kg",   NA
```

```
)

derive_vars_computed(
  dataset = adsl,
  dataset_add = advs,
  by_vars = exprs(STUDYID, USUBJID),
  parameters = c("WEIGHT", "HEIGHT"),
  new_vars = exprs(BMIBL = compute_bmi(height = AVAL.HEIGHT, weight = AVAL.WEIGHT)),
  filter_add = ABLFL == "Y"
)
```

---

derive_vars_crit_flag *Derive Criterion Flag Variables* CRITy, CRITyFL, *and* CRITyFN

Description

The function derives ADaM compliant criterion flags, e.g., to facilitate subgroup analyses.

If a criterion flag can't be derived with this function, the derivation is not ADaM compliant. It helps to ensure that:

- the condition of the criterion depends only on variables of the same row,
- the CRITyFL is populated with valid values, i.e, either "Y" and NA or "Y", "N", and NA,
- the CRITy variable is populated correctly, i.e.,
  - set to a constant value within a parameter if CRITyFL is populated with "Y", "N", and NA and
  - set to a constant value within a parameter if the criterion condition is fulfilled and to NA otherwise if CRITyFL is populated with "Y", and NA

Usage

```
derive_vars_crit_flag(
  dataset,
  crit_nr = 1,
  condition,
  description,
  values_yn = FALSE,
  create_numeric_flag = FALSE
)
```

Arguments

dataset	Input dataset
crit_nr	The criterion number, i.e., the y in CRITy
condition	Condition for flagging records See description of the values_yn argument for details on how the CRITyFL variable is populated.

description — The description of the criterion

The CRITy variable is set to the specified value.

An expression can be specified to set the value depending on the parameter. Please note that the value must be constant within a parameter.

values_yn — Should "Y" and "N" be used for CRITyFL?

If set to TRUE, the CRITyFL variable is set to "Y" if the condition (condition) evaluates to TRUE, it is set to "N" if the condition evaluate to FALSE, and to NA if it evaluates to NA.

Otherwise, the CRITyFL variable is set to "Y" if the condition (condition) evaluates to TRUE, and to NA otherwise.

create_numeric_flag

Create a numeric flag?

If set to TRUE, the CRITyFN variable is created. It is set to 1 if CRITyFL == "Y", it set to 0 if CRITyFL == "N", and to NA otherwise.

Value

The input dataset with the variables CRITy, CRITyFL, and optionally CRITyFN added.

See Also

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

---

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

Description

Derive a date (*DT) from a character date (--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 variables specified by the `dtc` argument are expected to be in the dataset.

`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"` (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, `NA_character_` is returned.

If `"Y"` (year, highest level) is specified, `date_imputation` must be `"first"` or `"last"` and `min_dates` or `max_dates` must be specified respectively. Otherwise, an error is thrown.

`date_imputation`

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

A character value is expected.

- If `highest_imputation` is `"M"`, month and day can be specified as `"mm-dd"`: e.g. `"06-15"` for the 15th of June
- When `highest_imputation` is `"M"` or `"D"`, the following keywords are available: `"first"`, `"mid"`, `"last"` to impute to the first/mid/last day/month. If `"mid"` is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as `"06-30"` (middle of the year).
  - If only day is missing, it is imputed as `"15"` (middle of the month).

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).

`flag_imputation`

Whether the date imputation flag must also be derived.

If `"auto"` is specified and `highest_imputation` argument is not `"n"`, then date imputation flag is derived.

If `"date"` is specified, then date imputation flag is derived.

If "none" is specified, then no date imputation flag is derived.

Please note that CDISC requirements dictate the need for a date imputation flag if any imputation is performed, so flag_imputation = "none" should only be used if the imputed variable is not part of the final ADaM dataset.

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").

Details

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

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

Value

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

See Also

vignette("imputation")

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

---

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

Description

Derive a datetime object (*DTM) from a character date (--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 = TRUE
)
```

Arguments

dataset	Input dataset The variables specified by the dtc argument are expected to be in the dataset.
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.

date_imputation

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

A character value is expected.

- If highest_imputation is "M", month and day can be specified as "mm-dd": e.g. "06-15" for the 15th of June
- When highest_imputation is "M" or "D", the following keywords are available: "first", "mid", "last" to impute to the first/mid/last day/month. If "mid" is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).

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).

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".

flag_imputation

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

If "both" or "date" is specified, then date imputation flag is derived. If "auto" is specified and highest_imputation argument is greater than "h", then date imputation flag is derived.

If "both" or "time" is specified, then time imputation flag is derived. If "auto" is specified and highest_imputation argument is not "n", then time imputation flag is derived.

If "none" is specified, then no date or time imputation flag is derived.

Please note that CDISC requirements dictate the need for a date/time imputation flag if any imputation is performed, so flag_imputation = "none" should only be used if the imputed variable is not part of the final ADaM dataset.

`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_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  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 lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

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

`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"`.

By default it is assumed that no seconds are collected and `*TMF` shouldn't be set to `"S"`. A user can set this to `FALSE` if seconds are collected.

The default value of `ignore_seconds_flag` is set to `TRUE` in admiral 1.4.0 and later.

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.

See Also

vignette("imputation")

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

---

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 The variables specified by the source_vars argument are expected to be in the dataset.
source_vars	A list of datetime variables created using exprs() from which dates are to be extracted

Value

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

See Also

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

Examples

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

adcm <- tribble(
  ~USUBJID, ~TRTSDTM,              ~ASTDTM,               ~AENDTM,
  "PAT01",  "2012-02-25 23:00:00", "2012-02-28 19:00:00", "2012-02-25 23:00:00",
  "PAT01",  NA,                    "2012-02-28 19:00:00", NA,
  "PAT01",  "2017-02-25 23:00:00", "2013-02-25 19:00:00", "2014-02-25 19:00:00",
  "PAT01",  "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-03-25 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(exprs(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 The variables specified by the source_vars argument are expected to be in the dataset.
source_vars	A list of datetime variables created using exprs() from which time is to be extracted

Details

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

Value

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

See Also

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

Examples

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

adcm <- tribble(
  ~USUBJID, ~TRTSDTM, ~ASTDTM, ~AENDTM,
  "PAT01", "2012-02-25 23:41:10", "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(exprs(TRTSDTM)) %>%
  select(USUBJID, starts_with("TRT"), everything())

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

---

derive_vars_duration *Derive Duration*

Description

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

Usage

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

Arguments

dataset	Input dataset The variables specified by the start_date and end_date arguments are expected to be in the dataset.
new_var	Name of variable to create
new_var_unit	Name of the unit variable If the parameter is not specified, no variable for the unit is created.
start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Refer to convert_dtc_to_dt() to obtain a vector of imputed dates.
end_date	The end date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Refer to convert_dtc_to_dt() to obtain a vector of imputed dates.
in_unit	Input unit See floor_in and add_one parameter for details. Permitted Values (case-insensitive): For years: "year", "years", "yr", "yrs", "y" For months: "month", "months", "mo", "mos" For days: "day", "days", "d" For hours: "hour", "hours", "hr", "hrs", "h" For minutes: "minute", "minutes", "min", "mins" For seconds: "second", "seconds", "sec", "secs", "s"
out_unit	Output unit The duration is derived in the specified unit

Permitted Values (case-insensitive):
For years: "year", "years", "yr", "yrs", "y"
For months: "month", "months", "mo", "mos"
For weeks: "week", "weeks", "wk", "wks", "w"
For days: "day", "days", "d"
For hours: "hour", "hours", "hr", "hrs", "h"
For minutes: "minute", "minutes", "min", "mins"
For seconds: "second", "seconds", "sec", "secs", "s"

floor_in — Round down input dates?
The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.

add_one — Add one input unit?
If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.

trunc_out — Return integer part
The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.

type — lubridate duration type.
See below for details.

Details

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

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

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

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

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

Value

The input dataset with the duration and unit variable added

See Also

compute_duration()

Date/Time Derivation Functions that returns variable appended to dataset: derive_var_trtdurd(), derive_vars_dt(), derive_vars_dtm(), derive_vars_dtm_to_dt(), derive_vars_dtm_to_tm(), 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 adverse event start time since last dose in hours
data <- tribble(
  ~USUBJID, ~ASTDTM, ~LDOSEDTM,
  "P01", ymd_hms("2019-08-09T04:30:56"), ymd_hms("2019-08-08T10:05:00"),
  "P02", ymd_hms("2019-11-11T23:59:59"), ymd_hms("2019-10-11T11:37:00"),
  "P03", ymd_hms("2019-11-11T00:00:00"), ymd_hms("2019-11-10T23:59:59"),
  "P04", ymd_hms("2019-11-11T12:34:56"), NA,
  "P05", NA, ymd_hms("2019-09-28T12:34:56")
)
derive_vars_duration(
  data,
  new_var = LDRELTM,
  new_var_unit = LDRELTMU,
  start_date = LDOSEDTM,
  end_date = ASTDTM,
  in_unit = "hours",
  out_unit = "hours",
  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 The variables specified by the `reference_date` and `source_vars` arguments are expected to be in the dataset.
`reference_date`	A date or date-time column, e.g., date of first treatment or date-time of last exposure to treatment. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

source_vars A list of datetime or date variables created using exprs() from which dates are to be extracted. This can either be a list of date(time) variables or named *DY variables and corresponding *DT(M) variables e.g. exprs(TRTSDTM, ASTDTM, AENDT) or exprs(TRTSDT, ASTDTM, AENDT, DEATHDY = DTHDT). If the source variable does not end in *DT(M), a name for the resulting *DY variable must be provided.

Details

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

Value

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

See Also

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

Examples

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

datain <- 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 = exprs(TRTSDTM, ASTDTM, AENDT)
)

# specifying name of new variables
datain <- tribble(
  ~TRTSDT, ~DTHDT,
  "2014-01-17", "2014-02-01"
) %>%
```

```
  mutate(
    TRTSDT = ymd(TRTSDT),
    DTHDT = ymd(DTHDT)
  )

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

`derive_vars_extreme_event` *Add the Worst or Best Observation for Each By Group as New Variables*

Description

Add the first available record from `events` for each by group as new variables, all variables of the selected observation are kept. It can be used for selecting the extreme observation from a series of user-defined events.

Usage

```
derive_vars_extreme_event(
  dataset,
  by_vars,
  events,
  tmp_event_nr_var = NULL,
  order,
  mode,
  source_datasets = NULL,
  check_type = "warning",
  new_vars
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset.
`by_vars`	Grouping variables
`events`	Conditions and new values defining events A list of `event()` or `event_joined()` objects is expected. Only observations listed in the `events` are considered for deriving extreme event. If multiple records meet the filter `condition`, take the first record sorted by `order`. The

data is grouped by by_vars, i.e., summary functions like all() or any() can be used in condition.

For event_joined() events the observations are selected by calling filter_joined(). The condition field is passed to the filter_join argument.

tmp_event_nr_var

Temporary event number variable

The specified variable is added to all source datasets and is set to the number of the event before selecting the records of the event.

It can be used in order to determine which record should be used if records from more than one event are selected.

The variable is not included in the output dataset.

order

Sort order

If a particular event from events has more than one observation, within the event and by group, the records are ordered by the specified order.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

mode

Selection mode (first or last)

If a particular event from events has more than one observation, "first"/"last" is used to select the first/last record of this type of event sorting by order.

source_datasets

Source datasets

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

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.

new_vars

Variables to add

The specified variables from the events are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>).

Details

1. For each event select the observations to consider:
   (a) If the event is of class event, the observations of the source dataset are restricted by condition and then the first or last (mode) observation per by group (by_vars) is selected.
   If the event is of class event_joined, filter_joined() is called to select the observations.
   (b) The variables specified by the set_values_to field of the event are added to the selected observations.
   (c) The variable specified for tmp_event_nr_var is added and set to the number of the event.
2. All selected observations are bound together.
3. 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.

4. The variables specified by the new_vars parameter are added to the selected observations.
5. The variables are added to input dataset.

Value

The input dataset with the best or worst observation of each by group added as new variables.

See Also

event(), event_joined(), derive_extreme_event()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_vars_aage(), derive_vars_period()

Examples

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

adsl <- tribble(
  ~STUDYID, ~USUBJID, ~TRTEDT, ~DTHDT,
  "PILOT01", "01-1130", ymd("2014-08-16"), ymd("2014-09-13"),
  "PILOT01", "01-1133", ymd("2013-04-28"), ymd(""),
  "PILOT01", "01-1211", ymd("2013-01-12"), ymd(""),
  "PILOT01", "09-1081", ymd("2014-04-27"), ymd(""),
  "PILOT01", "09-1088", ymd("2014-10-09"), ymd("2014-11-01"),
)

lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,             ~LBDTC,
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-05-01T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
) %>%
  mutate(
    ADT = convert_dtc_to_dt(LBDTC)
  )

derive_vars_extreme_event(
  adsl,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      dataset_name = "adsl",
      condition = !is.na(DTHDT),
```

```
      set_values_to = exprs(LSTALVDT = DTHDT, DTHFL = "Y")
    ),
    event(
      dataset_name = "lb",
      condition = !is.na(ADT),
      order = exprs(ADT),
      mode = "last",
      set_values_to = exprs(LSTALVDT = ADT, DTHFL = "N")
    ),
    event(
      dataset_name = "adsl",
      condition = !is.na(TRTEDT),
      order = exprs(TRTEDT),
      mode = "last",
      set_values_to = exprs(LSTALVDT = TRTEDT, DTHFL = "N")
    )
  ),
  source_datasets = list(adsl = adsl, lb = lb),
  tmp_event_nr_var = event_nr,
  order = exprs(LSTALVDT, event_nr),
  mode = "last",
  new_vars = exprs(LSTALVDT, DTHFL)
)

# Derive DTHCAUS from AE and DS domain data
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",
  "STUDY01", "PAT01", 13, "CARDIAC ARREST", "FATAL", "2021-04-03",
)

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"
)

derive_vars_extreme_event(
  adsl,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      dataset_name = "ae",
      condition = AEOUT == "FATAL",
      set_values_to = exprs(DTHCAUS = AEDECOD, DTHDT = convert_dtc_to_dt(AEDTHDTC)),
```

```
      order = exprs(DTHDT)
    ),
    event(
      dataset_name = "ds",
      condition = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
      set_values_to = exprs(DTHCAUS = DSTERM, DTHDT = convert_dtc_to_dt(DSSTDTC)),
      order = exprs(DTHDT)
    )
  ),
  source_datasets = list(ae = ae, ds = ds),
  tmp_event_nr_var = event_nr,
  order = exprs(DTHDT, event_nr),
  mode = "first",
  new_vars = exprs(DTHCAUS, DTHDT)
)
```

`derive_vars_joined` *Add Variables from an Additional Dataset Based on Conditions from Both Datasets*

Description

The function adds variables from an additional dataset to the input dataset. The selection of the observations from the additional dataset can depend on variables from both datasets. For example, add the lowest value (nadir) before the current observation.

Usage

```
derive_vars_joined(
  dataset,
  dataset_add,
  by_vars = NULL,
  order = NULL,
  new_vars = NULL,
  tmp_obs_nr_var = NULL,
  join_vars = NULL,
  join_type,
  filter_add = NULL,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_join = NULL,
  mode = NULL,
  exist_flag = NULL,
  true_value = "Y",
  false_value = NA_character_,
  missing_values = NULL,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Additional dataset The variables specified by the by_vars, the new_vars, the join_vars, and the order argument are expected.
by_vars	Grouping variables The two datasets are joined by the specified variables. Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r similar to the dplyr joins.
order	Sort order If the argument is set to a non-null value, for each observation of the input dataset the first or last observation from the joined dataset is selected with respect to the specified order. The specified variables are expected in the additional dataset (dataset_add). If a variable is available in both dataset and dataset_add, the one from dataset_add is used for the sorting. If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC), EXSEQ), a corresponding variable (EXSTDT) is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for join_vars and new_vars. The variable is not included in the output dataset. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
new_vars	Variables to add The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>). For example new_vars = exprs(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset. And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from dataset_add and adds them to the input dataset renaming old_var2 to new_var2. Values of the added variables can be modified by specifying an expression. For example, new_vars = LASTRSP = exprs(str_to_upper(AVALC)) adds the variable LASTRSP to the dataset and sets it to the upper case value of AVALC. If the argument is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to by_vars, removes or renames.
tmp_obs_nr_var	Temporary observation number The specified variable is added to the input dataset (dataset) and the additional dataset (dataset_add). It is set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. If there is more than one record for specific values for by_vars and order, all records get the same observation number. By default, a warning (see check_type) is issued in this case. The variable can be used in the conditions (filter_join,

first_cond_upper, first_cond_lower). It can also be used to select consecutive observations or the last observation.

The variable is not included in the output dataset. To include it specify it for new_vars.

join_vars — Variables to use from additional dataset

Any extra variables required from the additional dataset for filter_join should be specified for this argument. Variables specified for new_vars do not need to be repeated for join_vars. If a specified variable exists in both the input dataset and the additional dataset, the suffix ".join" is added to the variable from the additional dataset.

If an expression is named, e.g., exprs(EXTDT = convert_dtc_to_dt(EXSTDTC)), a corresponding variable is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for new_vars. The variable is not included in the output dataset.

The variables are not included in the output dataset.

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. Thus join_type = "after" could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus join_type = "all" could be used.

filter_add — Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by order or new_vars arguments can be used in the condition.

The condition can include summary functions like all() or any(). The additional dataset is grouped by the by variables (by_vars).

first_cond_lower — Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the last observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example, see the "Examples" section below.

first_cond_upper — Condition for selecting range of data (after)

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.

This argument should be specified if `filter_join` contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example, see the "Examples" section below.

`filter_join` Filter for the joined dataset

The specified condition is applied to the joined dataset. Therefore variables from both datasets `dataset` and `dataset_add` can be used.

Variables created by `order` or `new_vars` arguments can be used in the condition.

The condition can include summary functions like `all()` or `any()`. The joined dataset is grouped by the original observations.

`mode` Selection mode

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

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

`exist_flag` Exist flag

If the argument is specified (e.g., `exist_flag = FLAG`), the specified variable (e.g., `FLAG`) is added to the input dataset. This variable will be the value provided in `true_value` for all selected records from `dataset_add` which are merged into the input dataset, and the value provided in `false_value` otherwise.

`true_value` True value

The value for the specified variable `exist_flag`, applicable to the first or last observation (depending on the mode) of each by group.

`false_value` False value

The value for the specified variable `exist_flag`, NOT applicable to the first or last observation (depending on the mode) of each by group.

`missing_values` Values for non-matching observations

For observations of the input dataset (`dataset`) which do not have a matching observation in the additional dataset (`dataset_add`) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

`check_type` Check uniqueness?

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

This argument is ignored if `order` is not specified. In this case an error is issued independent of `check_type` if the restricted joined dataset contains more than one observation for any of the observations of the input dataset.

Details

1. The variables specified by `order` are added to the additional dataset (`dataset_add`).
2. The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
3. The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.
4. The input dataset and the (restricted) additional dataset are left joined by the grouping variables (`by_vars`). If no grouping variables are specified, a full join is performed.

5. If first_cond_lower is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where first_cond_lower is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   If first_cond_upper is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where first_cond_upper 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.

   For an example, see the "Examples" section below.
6. The joined dataset is restricted by the filter_join condition.
7. If order is specified, for each observation of the input dataset the first or last observation (depending on mode) is selected.
8. The variables specified for new_vars are created (if requested) and merged to the input dataset. I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the joined dataset the new variables are set as specified by missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option save_memory to TRUE (see set_admiral_options()). This reduces the memory consumption but increases the run-time.

Value

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

See Also

derive_var_joined_exist_flag(), filter_joined()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_su
derive_vars_transposed()

---

derive_vars_joined_summary

Summarize Variables from an Additional Dataset Based on Conditions from Both Datasets

---

Description

The function summarizes variables from an additional dataset and adds the summarized values as new variables to the input dataset. The selection of the observations from the additional dataset can depend on variables from both datasets. For example, all doses before the current observation can be selected and the sum be added to the input dataset.

Usage

```
derive_vars_joined_summary(
  dataset,
  dataset_add,
  by_vars = NULL,
  order = NULL,
  new_vars,
  tmp_obs_nr_var = NULL,
  join_vars = NULL,
  join_type,
  filter_add = NULL,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_join = NULL,
  missing_values = NULL,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Additional dataset The variables specified by the by_vars, the new_vars, the join_vars, and the order argument are expected.
by_vars	Grouping variables The two datasets are joined by the specified variables. Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r similar to the dplyr joins.
order	Sort order The specified variables are used to determine the order of the records if first_cond_lower or first_cond_upper is specified or if join_type equals "before" or "after". If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC), EXSEQ), a corresponding variable (EXSTDT) is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for join_vars and new_vars. The variable is not included in the output dataset. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
new_vars	Variables to add The new variables can be defined by named expressions, i.e., new_vars = exprs(<new variable> = <val The value must be defined such that it results in a single record per by group, e.g., by using a summary function like mean(), sum(), ...
tmp_obs_nr_var	Temporary observation number The specified variable is added to the input dataset (dataset) and the restricted additional dataset (dataset_add after applying filter_add). It is set to the

observation number with respect to order. For each by group (by_vars) the observation number starts with 1. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It can also be used to select consecutive observations or the last observation.

The variable is not included in the output dataset. To include it specify it for new_vars.

join_vars — Variables to use from additional dataset

Any extra variables required from the additional dataset for filter_join should be specified for this argument. Variables specified for new_vars do not need to be repeated for join_vars. If a specified variable exists in both the input dataset and the additional dataset, the suffix ".join" is added to the variable from the additional dataset.

If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC)), a corresponding variable is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for new_vars.

The 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.

filter_add — Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by order or new_vars arguments can be used in the condition.

The condition can include summary functions like all() or any(). The additional dataset is grouped by the by variables (by_vars).

first_cond_lower

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example see the last example below.

first_cond_upper

Condition for selecting range of data (after)

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.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example see the last example below.

`filter_join` Filter for the joined dataset

The specified condition is applied to the joined dataset. Therefore variables from both datasets `dataset` and `dataset_add` can be used.

Variables created by `order` or `new_vars` arguments can be used in the condition.

The condition can include summary functions like `all()` or `any()`. The joined dataset is grouped by the original observations.

`missing_values` Values for non-matching observations

For observations of the input dataset (`dataset`) which do not have a matching observation in the additional dataset (`dataset_add`) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

`check_type` Check uniqueness?

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

The uniqueness is checked only if `tmp_obs_nr_var`, `first_cond_lower`, or `first_cond_upper` is specified or `join_type` equals `"before"` or `"after"`.

Details

1. The variables specified by `order` are added to the additional dataset (`dataset_add`).
2. The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
3. The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.
4. The input dataset and the (restricted) additional dataset are left joined by the grouping variables (`by_vars`). If no grouping variables are specified, a full join is performed.
5. If `first_cond_lower` is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where `first_cond_lower` is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   If `first_cond_upper` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond_upper` 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.

   For an example see the last example in the "Examples" section.
6. The joined dataset is restricted by the `filter_join` condition.
7. The variables specified for `new_vars` are created and merged to the input dataset. I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the joined dataset the new variables are set as specified by `missing_values` (or to `NA` for variables not in `missing_values`). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

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

See Also

`derive_vars_joined()`, `derive_vars_merged_summary()`, `derive_var_joined_exist_flag()`, `filter_joined()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_cat()`, `derive_vars_computed()`, `derive_vars_joined()`, `derive_vars_merged()`, `derive_vars_merged_lookup()`, `derive_vars_merged_summary()`, `derive_vars_transposed()`

---

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,
  filter_add = NULL,
  mode = NULL,
  exist_flag = NULL,
  true_value = "Y",
  false_value = NA_character_,
  missing_values = NULL,
  check_type = "warning",
  duplicate_msg = NULL,
  relationship = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Additional dataset The variables specified by the by_vars, the new_vars, and the order argument are expected.
by_vars	Grouping variables The input dataset and the selected observations from the additional dataset are merged by the specified variables. Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r similar to the dplyr joins.
order	Sort order If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order. Variables defined by the new_vars argument can be used in the sort order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
new_vars	Variables to add The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>). For example new_vars = exprs(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset. And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from dataset_add and adds them to the input dataset renaming old_var2 to new_var2. Values of the added variables can be modified by specifying an expression. For example, new_vars = LASTRSP = exprs(str_to_upper(AVALC)) adds the variable LASTRSP to the dataset and sets it to the upper case value of AVALC. If the argument is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to by_vars, removes or renames.
filter_add	Filter for additional dataset (dataset_add) Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered. Variables defined by the new_vars argument can be used in the filter condition.
mode	Selection mode Determines if the first or last observation is selected. If the order argument is specified, mode must be non-null. If the order argument is not specified, the mode argument is ignored.
exist_flag	Exist flag If the argument is specified (e.g., exist_flag = FLAG), the specified variable (e.g., FLAG) is added to the input dataset. This variable will be the value provided

in true_value for all selected records from dataset_add which are merged into the input dataset, and the value provided in false_value otherwise.

true_value — True value

The value for the specified variable exist_flag, applicable to the first or last observation (depending on the mode) of each by group.

false_value — False value

The value for the specified variable exist_flag, NOT applicable to the first or last observation (depending on the mode) of each by group.

missing_values — Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

check_type — Check uniqueness?

If "warning", "message", 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.

If the order argument is not specified, the check_type argument is ignored: if the observations of the (restricted) additional dataset are not unique with respect to the by variables, an error is issued.

duplicate_msg — Message of unique check

If the uniqueness check fails, the specified message is displayed.

relationship — Expected merge-relationship between the by_vars variable(s) in dataset (input dataset) and the dataset_add (additional dataset) containing the additional new_vars.

This argument is passed to the dplyr::left_join() function. See https://dplyr.tidyverse.org/reference/mutate-joins.html#arguments for more details.

Details

1. The new variables (new_vars) are added to the additional dataset (dataset_add).
2. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The variables specified for new_vars are merged to the input dataset using left_join(). I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the additional dataset the new variables are set as specified by missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

---

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 The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Lookup table The variables specified by the by_vars argument are expected.
by_vars	Grouping variables The input dataset and the selected observations from the additional dataset are merged by the specified variables. Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r similar to the dplyr joins.

`order` Sort order

If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.

Variables defined by the `new_vars` argument can be used in the sort order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

`new_vars` Variables to add

The specified variables from the additional dataset are added to the output dataset.

Variables can be renamed by naming the element, i.e., `new_vars = exprs(<new name> = <old name>)`.

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

And `new_vars = exprs(var1, new_var2 = old_var2)` takes `var1` and `old_var2` from `dataset_add` and adds them to the input dataset renaming `old_var2` to `new_var2`.

Values of the added variables can be modified by specifying an expression. For example, `new_vars = LASTRSP = exprs(str_to_upper(AVALC))` adds the variable `LASTRSP` to the dataset and sets it to the upper case value of `AVALC`.

If the argument is not specified or set to `NULL`, all variables from the additional dataset (`dataset_add`) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to `by_vars`, removes or renames.

`mode` Selection mode

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

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

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

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

Variables defined by the `new_vars` argument can be used in the filter condition.

`check_type` Check uniqueness?

If `"warning"`, `"message"`, 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.

If the `order` argument is not specified, the `check_type` argument is ignored: if the observations of the (restricted) additional dataset are not unique with respect to the by variables, an error is issued.

`duplicate_msg` Message of unique check

If the uniqueness check fails, the specified message is displayed.

`print_not_mapped`

Print a list of unique `by_vars` values that do not have corresponding records from the lookup table?

Value

The output dataset contains all observations and variables of the input dataset, and add the variables specified in `new_vars` from the lookup table specified in `dataset_add`. Optionally prints a list of

unique by_vars values that do not have corresponding records from the lookup table (by specifying print_not_mapped = TRUE).

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_summary(), derive_vars_transposed()

Examples

```
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,        ~VISIT, ~VSTESTCD,       ~VSTEST,
  "PILOT01",    "VS", "01-1028",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "01-1028",   "SCREENING",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "01-1028",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "01-1028",      "WEEK 4",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "04-1325",   "SCREENING",    "TEMP", "Temperature",
  "PILOT01",    "VS", "04-1325",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "04-1325",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "04-1325", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",      "WEEK 4",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "10-1027",   "SCREENING",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",      "WEEK 4",  "WEIGHT",      "Weight"
)

param_lookup <- tribble(
  ~VSTESTCD,                 ~VSTEST, ~PARAMCD,                      ~PARAM,
  "SYSBP", "Systolic Blood Pressure",  "SYSBP", "Syst Blood Pressure (mmHg)",
  "WEIGHT",                "Weight", "WEIGHT",               "Weight (kg)",
  "HEIGHT",                "Height", "HEIGHT",               "Height (cm)",
  "TEMP",              "Temperature",   "TEMP",           "Temperature (C)",
  "MAP",    "Mean Arterial Pressure",    "MAP",   "Mean Art Pressure (mmHg)",
  "BMI",           "Body Mass Index",    "BMI",    "Body Mass Index(kg/m^2)",
  "BSA",         "Body Surface Area",    "BSA",     "Body Surface Area(m^2)"
)

derive_vars_merged_lookup(
  dataset = vs,
```

```
  dataset_add = param_lookup,
  by_vars = exprs(VSTESTCD),
  new_vars = exprs(PARAMCD, PARAM),
  print_not_mapped = TRUE
)
```

`derive_vars_merged_summary` *Merge Summary Variables*

Description

Merge a summary variable from a dataset to the input dataset.

Usage

```
derive_vars_merged_summary(
  dataset,
  dataset_add,
  by_vars,
  new_vars = NULL,
  filter_add = NULL,
  missing_values = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.
`dataset_add`	Additional dataset The variables specified by the `by_vars` and the variables used on the left hand sides of the `new_vars` arguments are expected.
`by_vars`	Grouping variables The expressions on the left hand sides of `new_vars` are evaluated by the specified *variables*. Then the resulting values are merged to the input dataset (`dataset`) by the specified *variables*.
`new_vars`	New variables to add The specified variables are added to the input dataset. A named list of expressions is expected: • LHS refer to a variable. • RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for `by_vars`. For example:

```
  new_vars = exprs(
    DOSESUM = sum(AVAL),
    DOSEMEAN = mean(AVAL)
  )
```

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

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

`missing_values` Values for non-matching observations

For observations of the input dataset (`dataset`) which do not have a matching observation in the additional dataset (`dataset_add`) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

Details

1. The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.
2. The new variables (`new_vars`) are created for each by group (`by_vars`) in the additional dataset (`dataset_add`) by calling `summarize()`. I.e., all observations of a by group are summarized to a single observation.
3. The new variables are merged to 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`.

See Also

`derive_summary_records()`, `get_summary_records()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_cat()`, `derive_vars_computed()`, `derive_vars_joined()`, `derive_vars_joined_summary()`, `derive_vars_merged()`, `derive_vars_merged_lookup()`, `derive_vars_transposed()`

Examples

```
library(tibble)

# Add a variable for the mean of AVAL within each visit
adbds <- tribble(
  ~USUBJID,  ~AVISIT,  ~ASEQ, ~AVAL,
  "1",      "WEEK 1",      1,    10,
  "1",      "WEEK 1",      2,    NA,
  "1",      "WEEK 2",      3,    NA,
```

```
  "1",      "WEEK 3",      4,    42,
  "1",      "WEEK 4",      5,    12,
  "1",      "WEEK 4",      6,    12,
  "1",      "WEEK 4",      7,    15,
  "2",      "WEEK 1",      1,    21,
  "2",      "WEEK 4",      2,    22
)

derive_vars_merged_summary(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(USUBJID, AVISIT),
  new_vars = exprs(
    MEANVIS = mean(AVAL, na.rm = TRUE),
    MAXVIS = max(AVAL, na.rm = TRUE)
  )
)

# Add a variable listing the lesion ids at baseline
adsl <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3"
)

adtr <- tribble(
  ~USUBJID,     ~AVISIT, ~LESIONID,
  "1",       "BASELINE",  "INV-T1",
  "1",       "BASELINE",  "INV-T2",
  "1",       "BASELINE",  "INV-T3",
  "1",       "BASELINE",  "INV-T4",
  "1",         "WEEK 1",  "INV-T1",
  "1",         "WEEK 1",  "INV-T2",
  "1",         "WEEK 1",  "INV-T4",
  "2",       "BASELINE",  "INV-T1",
  "2",       "BASELINE",  "INV-T2",
  "2",       "BASELINE",  "INV-T3",
  "2",         "WEEK 1",  "INV-T1",
  "2",         "WEEK 1",  "INV-N1"
)

derive_vars_merged_summary(
  adsl,
  dataset_add = adtr,
  by_vars = exprs(USUBJID),
  filter_add = AVISIT == "BASELINE",
  new_vars = exprs(LESIONSBL = paste(LESIONID, collapse = ", "))
)
```

derive_vars_period *Add Subperiod, Period, or Phase Variables to ADSL*

Description

The function adds subperiod, period, or phase variables like P01S1SDT, P01S2SDT, AP01SDTM, AP02SDTM, TRT01A, TRT02A, PH1SDT, PH2SDT, ... to the input dataset. The values of the variables are defined by a period reference dataset which has one observations per patient and subperiod, period, or phase.

Usage

```
derive_vars_period(
  dataset,
  dataset_ref,
  new_vars,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

dataset	Input dataset The variables specified by the subject_keys argument are expected to be in the dataset.
dataset_ref	Period reference dataset The variables specified by new_vars and subject_keys are expected. If subperiod variables are requested, APERIOD and ASPER are expected. If period variables are requested. APERIOD is expected. If phase variables are requested, APHASEN is expected.
new_vars	New variables A named list of variables like exprs(PHwSDT = PHSDT, PHwEDT = PHEDT, APHASEw = APHASE) is expected. The left hand side of the elements defines a set of variables (in CDISC notation) to be added to the output dataset. The right hand side defines the source variable from the period reference dataset. If the lower case letter "w" is used it refers to a phase variable, if the lower case letters "xx" are used it refers to a period variable, and if both "xx" and "w" are used it refers to a subperiod variable. Only one type must be used, e.g., all left hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables. If period *and* subperiod variables are required, separate calls must be used.
subject_keys	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by exprs() is expected.

Details

For each subperiod/period/phase in the period reference dataset and each element in `new_vars` a variable (LHS value of `new_vars`) is added to the output dataset and set to the value of the source variable (RHS value of `new_vars`.

Value

The input dataset with subperiod/period/phase variables added (see "Details" section)

See Also

`create_period_dataset()`

ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_vars_aage()`, `derive_vars_extreme_event()`

Examples

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

adsl <- tibble(STUDYID = "xyz", USUBJID = c("1", "2"))

# Add period variables to ADSL
period_ref <- tribble(
  ~USUBJID, ~APERIOD, ~APERSDT,     ~APEREDT,
  "1",             1, "2021-01-04", "2021-02-06",
  "1",             2, "2021-02-07", "2021-03-07",
  "2",             1, "2021-02-02", "2021-03-02",
  "2",             2, "2021-03-03", "2021-04-01"
) %>%
  mutate(
    STUDYID = "xyz",
    APERIOD = as.integer(APERIOD),
    across(matches("APER[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = period_ref,
  new_vars = exprs(APxxSDT = APERSDT, APxxEDT = APEREDT)
) %>%
  select(STUDYID, USUBJID, AP01SDT, AP01EDT, AP02SDT, AP02EDT)

# Add phase variables to ADSL
phase_ref <- tribble(
  ~USUBJID, ~APHASEN, ~PHSDT,       ~PHEDT,       ~APHASE,
  "1",             1, "2021-01-04", "2021-02-06", "TREATMENT",
  "1",             2, "2021-02-07", "2021-03-07", "FUP",
  "2",             1, "2021-02-02", "2021-03-02", "TREATMENT"
) %>%
```

```
  mutate(
    STUDYID = "xyz",
    APHASEN = as.integer(APHASEN),
    across(matches("PH[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = phase_ref,
  new_vars = exprs(PHwSDT = PHSDT, PHwEDT = PHEDT, APHASEw = APHASE)
) %>%
  select(STUDYID, USUBJID, PH1SDT, PH1EDT, PH2SDT, PH2EDT, APHASE1, APHASE2)

# Add subperiod variables to ADSL
subperiod_ref <- tribble(
  ~USUBJID, ~APERIOD, ~ASPER, ~ASPRSDT,     ~ASPREDT,
  "1",              1,      1, "2021-01-04", "2021-01-19",
  "1",              1,      2, "2021-01-20", "2021-02-06",
  "1",              2,      1, "2021-02-07", "2021-03-07",
  "2",              1,      1, "2021-02-02", "2021-03-02",
  "2",              2,      1, "2021-03-03", "2021-04-01"
) %>%
  mutate(
    STUDYID = "xyz",
    APERIOD = as.integer(APERIOD),
    ASPER = as.integer(ASPER),
    across(matches("ASPR[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = subperiod_ref,
  new_vars = exprs(PxxSwSDT = ASPRSDT, PxxSwEDT = ASPREDT)
) %>%
  select(STUDYID, USUBJID, P01S1SDT, P01S1EDT, P01S2SDT, P01S2EDT, P02S1SDT, P02S1EDT)
```

`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 PREFIX, GRPNAME, SRCVAR, TERMCHAR and/or TERMNUM, and optional columns GRPID, SCOPE, SCOPEN. 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 vignette("occds").

A query dataset is expected as an input to this function. See the vignette("queries_dataset") for descriptions, or call data("queries") for an example of a query dataset.

For each unique element in PREFIX, the corresponding "NAM" variable will be created. For each unique PREFIX, if GRPID is not "" or NA, then the corresponding "CD" variable is created; similarly, if SCOPE is not "" or NA, then the corresponding "SC" variable will be created; if SCOPEN is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in dataset, the "NAM" variable takes the value of GRPNAME if the value of TERMCHAR or TERMNUM in dataset_queries matches the value of the respective SRCVAR in dataset. Note that TERMCHAR in dataset_queries dataset may be NA only when TERMNUM is non-NA and vice versa. The matching is case insensitive. The "CD", "SC", and "SCN" variables are derived accordingly based on GRPID, SCOPE, and SCOPEN respectively, whenever not missing.

Value

The input dataset with query variables derived.

See Also

create_query_data()

OCCDS Functions: derive_var_trtemfl(), derive_vars_atc()

Examples

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

`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,
  id_vars = NULL,
  key_var,
  value_var,
  filter = NULL,
  relationship = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.
`dataset_merge`	Dataset to transpose and merge The variables specified by the `by_vars`, `id_vars`, `key_var` and `value_var` arguments are expected. The variables `by_vars`, `id_vars`, `key_var` have to be a unique key.
`by_vars`	Grouping variables Keys used to merge `dataset_merge` with `dataset`.
`id_vars`	ID variables Variables (excluding `by_vars` and `key_var`) that uniquely identify each observation in `dataset_merge`.
`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
`relationship`	Expected merge-relationship between the `by_vars` variable(s) in `dataset` and `dataset_merge` (after transposition) This argument is passed to the `dplyr::left_join()` function. See https://dplyr.tidyverse.org/reference/mutate-joins.html#arguments for more details.

Details

1. The records from the dataset to transpose and merge (`dataset_merge`) are restricted to those matching the `filter` condition, if provided.
2. The records from `dataset_merge` are checked to ensure they are uniquely identified using `by_vars`, `id_vars` and `key_var`.
3. `dataset_merge` is transposed (from "tall" to "wide"), with new variables added whose names come from `key_var` and values come from `value_var`.
4. The transposed dataset is merged with the input `dataset` using `by_vars` as keys. If a `relationship` has been provided, this merge must satisfy the relationship, otherwise an error is thrown.

Note that unlike other `derive_vars_*()` functions, the final step may cause new records to be added to the input dataset. The `relationship` argument can be specified to ensure this does not happen inadvertently.

Value

The input dataset with transposed variables from `dataset_merge` added

See Also

`derive_vars_atc()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_cat()`, `derive_vars_computed()`, `derive_vars_joined()`, `derive_vars_joined_summary()`, `derive_vars_merged()`, `derive_vars_merged_lookup()`, `derive_vars_merged_summary()`

Examples

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

# Adding ATC classes to CM using FACM
cm <- tribble(
  ~USUBJID,       ~CMGRPID, ~CMREFID,  ~CMDECOD,
  "BP40257-1001", "14",     "1192056", "PARACETAMOL",
  "BP40257-1001", "18",     "2007001", "SOLUMEDROL",
  "BP40257-1002", "19",     "2791596", "SPIRONOLACTONE"
)
facm <- tribble(
  ~USUBJID,       ~FAGRPID, ~FAREFID,  ~FATESTCD,  ~FASTRESC,
  "BP40257-1001", "1",      "1192056", "CMATC1CD", "N",
  "BP40257-1001", "1",      "1192056", "CMATC2CD", "N02",
  "BP40257-1001", "1",      "1192056", "CMATC3CD", "N02B",
  "BP40257-1001", "1",      "1192056", "CMATC4CD", "N02BE",
  "BP40257-1001", "1",      "2007001", "CMATC1CD", "D",
  "BP40257-1001", "1",      "2007001", "CMATC2CD", "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(
    dataset_merge = facm,
    by_vars = exprs(USUBJID, CMREFID = FAREFID),
    id_vars = exprs(FAGRPID),
    key_var = FATESTCD,
    value_var = FASTRESC
  ) %>%
  select(USUBJID, CMDECOD, starts_with("CMATC"))

# Note: the `id_vars` argument here is needed to uniquely identify
# rows of dataset_merge and avoid duplicates-related errors.
# Compare the above call to when `id_vars = NULL`:

try(
  cm %>%
    derive_vars_transposed(
      dataset_merge = facm,
      by_vars = exprs(USUBJID, CMREFID = FAREFID),
      id_vars = NULL,
      key_var = FATESTCD,
      value_var = FASTRESC
    )
)
```

`derive_var_age_years` *Derive Age in Years*

Description

Converts the given age variable (`age_var`) to the unit 'years' from the current units given in the `age_var+U` variable or `age_unit` argument and stores in a new variable (`new_var`).

Usage

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

Arguments

dataset	Input dataset The variables specified by the age_var argument are expected to be in the dataset.
age_var	Age variable. A numeric object is expected.
age_unit	Age unit. The age_unit argument is only expected when there is NOT a variable age_var+U in dataset. This gives the unit of the age_var variable and is used to convert AGE to 'years' so that grouping can occur.
new_var	New age variable to be created in years. The returned values are doubles and NOT integers. '

Details

This function is used to convert an age variable into the unit 'years' which can then be used to create age groups. The resulting column contains the equivalent years as a double. Note, underlying computations assume an equal number of days in each year (365.25).

Value

The input dataset (dataset) with new_var variable added in years.

See Also

derive_vars_duration()

ADSL Functions that returns variable appended to dataset: derive_vars_aage(), derive_vars_extreme_event(), derive_vars_period()

Examples

```
library(tibble)

# Derive age with age units specified
data <- tribble(
  ~AGE, ~AGEU,
  27, "days",
  24, "months",
  3, "years",
  4, "weeks",
  1, "years"
)

derive_var_age_years(data, AGE, new_var = AAGE)

# Derive age without age units variable specified
data <- tribble(
  ~AGE,
  12,
```

```
  24,
  36,
  48
)
derive_var_age_years(data, AGE, age_unit = "months", new_var = AAGE)
```

`derive_var_analysis_ratio` *Derive Ratio Variable*

Description

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

Usage

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

Arguments

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

Details

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

Note that `R2AyHI` and `R2AyLO` can also be derived using this function.

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

Value

The input dataset with a ratio variable appended

See Also

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

Examples

```
library(tibble)

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

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

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

---

derive_var_anrind *Derive Reference Range Indicator*

Description

Derive Reference Range Indicator

Usage

```
derive_var_anrind(
  dataset,
  signif_dig = get_admiral_option("signif_digits"),
  use_a1hia1lo = FALSE
)
```

Arguments

dataset	Input dataset ANRLO, ANRHI, and AVAL are expected and if use_a1hia1lo is set to TRUE, A1LO and A1H1 are expected as well.
signif_dig	Number of significant digits to use when comparing values. Significant digits used to avoid floating point discrepancies when comparing numeric values. See blog: How admiral handles floating points
use_a1hia1lo	A logical value indicating whether to use A1H1 and A1LO in the derivation of ANRIND.

Details

In the case that A1H1 and A1LO are to be used, ANRIND is set to:

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

In the case that A1H1 and A1LO are not to be used, ANRIND is set to:

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

Value

The input dataset with additional column ANRIND

See Also

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

Examples

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

vs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ANRLO, ~ANRHI, ~A1LO, ~A1HI,
  "P01",       "PUL",    70,     60,    100,    40,   110,
  "P01",       "PUL",    57,     60,    100,    40,   110,
  "P01",       "PUL",    60,     60,    100,    40,   110,
  "P01",     "DIABP",   102,     60,     80,    40,    90,
  "P02",       "PUL",   109,     60,    100,    40,   110,
  "P02",       "PUL",   100,     60,    100,    40,   110,
  "P02",     "DIABP",    80,     60,     80,    40,    90,
  "P03",       "PUL",    39,     60,    100,    40,   110,
  "P03",       "PUL",    40,     60,    100,    40,   110
)

vs %>% derive_var_anrind(use_a1hia1lo = TRUE)
vs %>% derive_var_anrind(use_a1hia1lo = FALSE)
```

`derive_var_atoxgr` *Derive Lab High toxicity Grade 0 - 4 and Low Toxicity Grades 0 - (-4)*

Description

Derives character lab grade based on high and low severity/toxicity grade(s).

Usage

```
derive_var_atoxgr(
  dataset,
  lotox_description_var = ATOXDSCL,
  hitox_description_var = ATOXDSCH
)
```

Arguments

`dataset`	Input dataset The variables specified by the `lotox_description_var` and `hitox_description_var` arguments are expected to be in the dataset. `ATOXGRL`, and `ATOXGRH` are expected as well.
`lotox_description_var`	Variable containing the toxicity grade description for low values, eg. "Anemia"
`hitox_description_var`	Variable containing the toxicity grade description for high values, eg. "Hemoglobin Increased".

Details

Created variable ATOXGR will contain values "-4", "-3", "-2", "-1" for low values and "1", "2", "3", "4" for high values, and will contain "0" if value is gradable and does not satisfy any of the criteria for high or low values. ATOXGR is set to missing if information not available to give a grade.

Function applies the following rules:

- High and low missing - overall missing
- Low grade not missing and > 0 - overall holds low grade
- High grade not missing and > 0 - overall holds high grade
- (Only high direction OR low direction is NORMAL) and high grade normal - overall NORMAL
- (Only low direction OR high direction is NORMAL) and low grade normal - overall NORMAL
- otherwise set to missing

Value

The input data set with the character variable added

See Also

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

Examples

```
library(tibble)

adlb <- tribble(
  ~ATOXDSCL,          ~ATOXDSCH,        ~ATOXGRL,      ~ATOXGRH,
  "Hypoglycemia",     "Hyperglycemia",  NA_character_, "0",
  "Hypoglycemia",     "Hyperglycemia",  "0",           "1",
  "Hypoglycemia",     "Hyperglycemia",  "0",           "0",
  NA_character_,      "INR Increased",  NA_character_, "0",
  "Hypophosphatemia", NA_character_,    "1",           NA_character_
)

derive_var_atoxgr(adlb)
```

`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,
  criteria_direction,
  abnormal_indicator = NULL,
  high_indicator = NULL,
  low_indicator = NULL,
  get_unit_expr,
  signif_dig = get_admiral_option("signif_digits")
)
```

Arguments

`dataset`	Input dataset The variables specified by the `tox_description_var` argument are expected to be in the dataset.
`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)
`criteria_direction`	Direction (L= Low, H = High) of toxicity grade.
`abnormal_indicator`	**[Deprecated]** Please use `low_indicator` and `high_indicator` instead.
`high_indicator`	Value in `BNRIND` derivation to indicate an abnormal high value. Usually "HIGH" for `criteria_direction` = "H". This is only required when `meta_criteria` = `atoxgr_criteria_ctcv5` or `meta_criteria` = `atoxgr_criteria_ctcv6` and `BNRIND` is a required variable. Currently, for terms `"Alanine aminotransferase increased"`, `"Aspartate aminotransferase increased"`, `"Blood bilirubin increased"` and `"GGT increased"` for both sets of criteria. Also, term `"Alkaline phosphatase increased"` for `meta_criteria` = `atoxgr_criteria_ctcv5`.

low_indicator — Value in BNRIND derivation to indicate an abnormal low value. Usually "LOW" for criteria_direction = "L".

This is only required when meta_criteria = atoxgr_criteria_ctcv6 and BNRIND is a required variable. Currently, only for term "Creatinine increased".

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 UNIT_CHECK in metadata (see meta_criteria parameter).

signif_dig — Number of significant digits to use when comparing a lab value against another value.

Significant digits used to avoid floating point discrepancies when comparing numeric values. See blog: How admiral handles floating points

Details

new_var is derived with values NA, "0", "1", "2", "3", "4", where "4" is the most severe grade

- "4" is where the lab value satisfies the criteria for grade 4.
- "3" is where the lab value satisfies the criteria for grade 3.
- "2" is where the lab value satisfies the criteria for grade 2.
- "1" is where the lab value satisfies the criteria for grade 1.
- "0" is where a grade can be derived and is not grade "1", "2", "3" or "4".
- NA is where a grade cannot be derived.

Value

The input dataset with the character variable added

See Also

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

Examples

```
library(tibble)

data <- tribble(
  ~ATOXDSCL,                   ~AVAL, ~ANRLO, ~ANRHI, ~PARAM,
  "Hypoglycemia",              119,   4,      7,      "Glucose (mmol/L)",
  "Lymphocyte count decreased", 0.7,   1,      4,      "Lymphocytes Abs (10^9/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_ctcv5,
  criteria_direction = "L",
  get_unit_expr = extract_unit(PARAM)
)

data <- tribble(
  ~ATOXDSCH,                    ~AVAL,  ~ANRLO,   ~ANRHI, ~PARAM,
  "CPK increased",              129,    0,        30,     "Creatine Kinase (U/L)",
  "Lymphocyte count increased", 4,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "Lymphocyte count increased", 2,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "CPK increased",              140,    120,      180,    "Creatine Kinase (U/L)"
)

derive_var_atoxgr_dir(data,
  new_var = ATOXGRH,
  tox_description_var = ATOXDSCH,
  meta_criteria = atoxgr_criteria_ctcv5,
  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.

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

Usage

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

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `source_var` arguments are expected to be in the dataset.

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

Details

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

Value

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

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)

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

## Derive `BASE` variable from `AVAL`
derive_var_base(
  dataset,
```

```
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = AVAL,
  new_var = BASE
)

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

## Derive `BNRIND` variable from `ANRIND`
derive_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = ANRIND,
  new_var = BNRIND
)
```

`derive_var_chg` *Derive Change from Baseline*

Description

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

Usage

```
derive_var_chg(dataset)
```

Arguments

`dataset`	Input dataset `AVAL` and `BASE` are expected.

Details

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

Value

The input dataset with an additional column named `CHG`

See Also

BDS-Findings Functions that returns variable appended to dataset: `derive_basetype_records()`, `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_atoxgr_dir()`, `derive_var_base()`, `derive_var_nfrlt()`, `derive_var_ontrtfl()`, `derive_var_pchg()`, `derive_var_shift()`, `derive_vars_crit_flag()`

Examples

```
library(tibble)

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

---

derive_var_dthcaus *Derive Death Cause*

Description

[Deprecated] The `derive_var_dthcaus()` function has been deprecated in favor of `derive_vars_extreme_event()`.

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

Usage

```
derive_var_dthcaus(
  dataset,
  ...,
  source_datasets,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

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

Details

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

Value

The input dataset with DTHCAUS variable added.

See Also

dthcaus_source()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

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

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

ds <- tribble(
  ~STUDYID, ~USUBJID, ~DSSEQ, ~DSDECOD, ~DSTERM, ~DSSTDTC,
 "STUDY01", "PAT02", 1, "INFORMED CONSENT OBTAINED", "INFORMED CONSENT OBTAINED", "2021-04-03",
  "STUDY01", "PAT02", 2, "RANDOMIZATION", "RANDOMIZATION", "2021-04-11",
  "STUDY01", "PAT02", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
  "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
)

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

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)

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

```
# Derive `DTHCAUS` and add traceability variables - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD,
  set_values_to = exprs(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM,
  set_values_to = exprs(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

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

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

ds <- mutate(
  ds,
  DSSTDT = convert_dtc_to_dt(DSSTDTC)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDT,
  mode = "first",
  dthcaus = DSTERM,
  set_values_to = exprs(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",
  set_values_to = exprs(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

[Deprecated] The derive_var_extreme_dt() function has been deprecated in favor of derive_vars_extreme_event().

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

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

Usage

```
derive_var_extreme_dt(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

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

Details

The following steps are performed to create the output dataset:

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

Value

The input dataset with the new variable added.

See Also

`date_source()`, `derive_var_extreme_dtm()`, `derive_vars_merged()`

Other deprecated: `call_user_fun()`, `date_source()`, `derive_param_extreme_record()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_var_merged_summary()`, `dthcaus_source()`, `get_summary_records()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AESEQ,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "01-1130",      5, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      6, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      4, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      8, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      7, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      2, "2014-03-09", "2014-03-09",
  "PILOT01",    "AE", "01-1130",      1, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1130",      3, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1133",      1, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      3, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      2, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      4, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1211",      5, "2012-11-29",           NA,
  "PILOT01",    "AE", "01-1211",      1, "2012-11-16",           NA,
  "PILOT01",    "AE", "01-1211",      7, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      8, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      4, "2012-11-22",           NA,
  "PILOT01",    "AE", "01-1211",      2, "2012-11-21", "2012-11-21",
  "PILOT01",    "AE", "01-1211",      3, "2012-11-21",           NA,
  "PILOT01",    "AE", "01-1211",      6, "2012-12-09",           NA,
  "PILOT01",    "AE", "01-1211",      9, "2013-01-14", "2013-01-14",
```

```
  "PILOT01",    "AE", "09-1081",      2, "2014-05-01",          NA,
  "PILOT01",    "AE", "09-1081",      1, "2014-04-07",          NA,
  "PILOT01",    "AE", "09-1088",      1, "2014-05-08",          NA,
  "PILOT01",    "AE", "09-1088",      2, "2014-08-02",          NA
)

adsl <- tribble(
  ~STUDYID,   ~USUBJID,              ~TRTEDTM,      ~TRTEDT,
  "PILOT01", "01-1130", "2014-08-16 23:59:59", "2014-08-16",
  "PILOT01", "01-1133", "2013-04-28 23:59:59", "2013-04-28",
  "PILOT01", "01-1211", "2013-01-12 23:59:59", "2013-01-12",
  "PILOT01", "09-1081", "2014-04-27 23:59:59", "2014-04-27",
  "PILOT01", "09-1088", "2014-10-09 23:59:59", "2014-10-09"
) %>%
  mutate(
    across(TRTEDTM:TRTEDT, as.Date)
  )


lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,             ~LBDTC,
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-04-29T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
)

dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1130",   84, "YEARS",
  "PILOT01",    "DM", "01-1133",   81, "YEARS",
  "PILOT01",    "DM", "01-1211",   76, "YEARS",
  "PILOT01",    "DM", "09-1081",   86, "YEARS",
  "PILOT01",    "DM", "09-1088",   69, "YEARS"
)

ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M")
)
ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M")
)

ae_ext <- ae %>%
  derive_vars_dt(
```

```
    dtc = AESTDTC,
    new_vars_prefix = "AEST",
    highest_imputation = "M"
  ) %>%
  derive_vars_dt(
    dtc = AEENDTC,
    new_vars_prefix = "AEEN",
    highest_imputation = "M"
  )

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC)
)

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

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

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

# derive last alive date and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
```

```
    LALVVAR = "AEENDTC"
  )
)

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC),
  set_values_to = exprs(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

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

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

`derive_var_extreme_dtm` *Derive First or Last Datetime from Multiple Sources*

Description

[Deprecated]

The `derive_var_extreme_dtm()` function has been deprecated in favor of `derive_vars_extreme_event()`.

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 = get_admiral_option("subject_keys")
)
```

Arguments

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

Details

The following steps are performed to create the output dataset:

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

Value

The input dataset with the new variable added.

See Also

date_source(), derive_var_extreme_dt(), derive_vars_merged()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate)
dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1130",   84, "YEARS",
  "PILOT01",    "DM", "01-1133",   81, "YEARS",
  "PILOT01",    "DM", "01-1211",   76, "YEARS",
  "PILOT01",    "DM", "09-1081",   86, "YEARS",
  "PILOT01",    "DM", "09-1088",   69, "YEARS"
)
ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AESEQ,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "01-1130",      5, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      6, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      4, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      8, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      7, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      2, "2014-03-09", "2014-03-09",
  "PILOT01",    "AE", "01-1130",      1, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1130",      3, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1133",      1, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      3, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      2, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      4, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1211",      5, "2012-11-29",           NA,
  "PILOT01",    "AE", "01-1211",      1, "2012-11-16",           NA,
  "PILOT01",    "AE", "01-1211",      7, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      8, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      4, "2012-11-22",           NA,
  "PILOT01",    "AE", "01-1211",      2, "2012-11-21", "2012-11-21",
  "PILOT01",    "AE", "01-1211",      3, "2012-11-21",           NA,
  "PILOT01",    "AE", "01-1211",      6, "2012-12-09",           NA,
  "PILOT01",    "AE", "01-1211",      9, "2013-01-14", "2013-01-14",
  "PILOT01",    "AE", "09-1081",      2, "2014-05-01",           NA,
  "PILOT01",    "AE", "09-1081",      1, "2014-04-07",           NA,
  "PILOT01",    "AE", "09-1088",      1, "2014-05-08",           NA,
  "PILOT01",    "AE", "09-1088",      2, "2014-08-02",           NA
)
lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,            ~LBDTC,
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-04-29T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
```

```
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
)
adsl <- tribble(
  ~STUDYID,   ~USUBJID,               ~TRTEDTM,
  "PILOT01", "01-1130", "2014-08-16 23:59:59",
  "PILOT01", "01-1133", "2013-04-28 23:59:59",
  "PILOT01", "01-1211", "2013-01-12 23:59:59",
  "PILOT01", "09-1081", "2014-04-27 23:59:59",
  "PILOT01", "09-1088", "2014-10-09 23:59:59"
) %>%
  mutate(
    TRTEDTM = as_datetime(TRTEDTM)
  )

# derive last known alive datetime (LSTALVDTM)
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
)
ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
)

ae_ext <- ae %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AEST",
    highest_imputation = "M"
  ) %>%
  derive_vars_dtm(
    dtc = AEENDTC,
    new_vars_prefix = "AEEN",
    highest_imputation = "M"
  )

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
)

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

adsl_date <- date_source(
  dataset_name = "adsl",
```

```
  date = TRTEDTM
)

dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM)

# derive last alive datetime and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)
lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
  set_values_to = exprs(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

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

```
  )
)

dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM, 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,
  true_value = "Y",
  false_value = NA_character_,
  flag_all = FALSE,
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.
`by_vars`	Grouping variables
`order`	Sort order The first or last observation is determined with respect to the specified order. For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

`new_var` Variable to add

The specified variable is added to the output dataset. It is set to the value set in `true_value` for the first or last observation (depending on the mode) of each by group.

`mode` Flag mode

Determines of the first or last observation is flagged.

`true_value` True value

The value for the specified variable `new_var`, applicable to the first or last observation (depending on the mode) of each by group.

`false_value` False value

The value for the specified variable `new_var`, NOT applicable to the first or last observation (depending on the mode) of each by group.

`flag_all` Flag setting

A logical value where if set to `TRUE`, all records are flagged and no error or warning is issued if the first or last record is not unique.

`check_type` Check uniqueness?

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

Details

For each group (with respect to the variables specified for the `by_vars` parameter), `new_var` is set to `"Y"` for the first or last observation (with respect to the order specified for the `order` parameter and the flag mode specified for the `mode` parameter). In the case where the user wants to flag multiple records of a grouping, for example records that all happen on the same visit and time, the argument `flag_all` can be set to `TRUE`. Otherwise, `new_var` is set to `NA`. Thus, the direction of "worst" is considered fixed for all parameters in the dataset depending on the `order` and the `mode`, i.e. for every parameter the first or last record will be flagged across the whole dataset.

Value

The input dataset with the new flag variable added

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_cat()`, `derive_vars_computed()`, `derive_vars_joined()`, `derive_vars_joined_summary()`, `derive_vars_merged()`, `derive_vars_merged_lookup()`, `derive_vars_merged_sum`
`derive_vars_transposed()`

derive_var_joined_exist_flag
Derives a Flag Based on an Existing Flag

Description

Derive a flag which depends on other observations of the dataset. For example, flagging events which need to be confirmed by a second event.

Usage

```
derive_var_joined_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  tmp_obs_nr_var = NULL,
  join_vars,
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_add = NULL,
  filter_join,
  true_value = "Y",
  false_value = NA_character_,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and join_vars arguments are expected to be in the dataset.
dataset_add	Additional dataset The variables specified for by_vars, join_vars, and order are expected.
by_vars	Grouping variables The specified variables are used for joining the input dataset (dataset) with the additional dataset (dataset_add).
order	Order The observations are ordered by the specified order if join_type = "after", join_type = "before", first_cond_lower, first_cond_upper, or tmp_obs_nr_var are specified. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

new_var — New variable

The specified variable is added to the input dataset.

tmp_obs_nr_var — Temporary observation number

The specified variable is added to the input dataset (dataset) and the additional dataset (dataset_add). It is set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. If there is more than one record for specific values for by_vars and order, all records get the same observation number. By default, a warning (see check_type) is issued in this case. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It is not included in the output dataset. It can also be used to flag consecutive observations or the last observation (see last example below).

join_vars — Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to flag all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit join_vars = exprs(AVALC, AVISITN) and filter_join = 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. Thus join_type = "after" could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus join_type = "all" could be used.

first_cond_lower

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. 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_join contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example see the last example below.

first_cond_upper

Condition for selecting range of data (after)

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_join` 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_add` Filter for additional dataset (`dataset_add`)

Only observations from `dataset_add` fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by `order` or `new_vars` arguments can be used in the condition.

The condition can include summary functions like `all()` or `any()`. The additional dataset is grouped by the by variables (`by_vars`).

`filter_join` Condition for selecting observations

The filter is applied to the joined dataset for flagging the confirmed observations. The condition can include summary functions like `all()` or `any()`. 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_join = 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

`false_value` Value of `new_var` for observations not flagged

`check_type` Check uniqueness?

If `"message"`, `"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.

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 variables specified by `order` are added to the additional dataset (`dataset_add`).
- The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
- The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.

The input dataset (`dataset`) is joined with the restricted additional dataset by the variables specified for `by_vars`. From the additional dataset only the variables specified for `join_vars` are kept. The suffix ".join" is added to those variables which also exist in the input dataset.

For example, for `by_vars = USUBJID`, `join_vars = exprs(AVISITN, AVALC)` and input dataset and additional 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 = exprs(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_lower` is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where `first_cond_lower` is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

If `first_cond_upper` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond_upper` 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.

For examples see the "Examples" section.

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_join`.

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`.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

The input dataset with the variable specified by new_var added.

See Also

filter_joined(), derive_vars_joined()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

---

derive_var_merged_ef_msrc

Merge an Existence Flag From Multiple Sources

Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in one of the source datasets fulfilling a certain condition. For example, if a dose adjustment flag should be added to ADEX but the dose adjustment information is collected in different datasets, e.g., EX, EC, and FA.

Usage

```
derive_var_merged_ef_msrc(
  dataset,
  by_vars,
  flag_events,
  source_datasets,
  new_var,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
by_vars	Grouping variables
flag_events	Flag events A list of flag_event() objects is expected. For each event the condition (condition field) is evaluated in the source dataset referenced by the dataset_name field. If it evaluates to TRUE at least once, the new variable is set to true_value.

`source_datasets`
: Source datasets

  A named list of datasets is expected. The `dataset_name` field of `flag_event()` refers to the dataset provided in the list.

`new_var`
: New variable

  The specified variable is added to the input dataset.

`true_value`
: True value

  The new variable (`new_var`) is set to the specified value for all by groups for which at least one of the source object (`sources`) has the condition evaluate to `TRUE`.

  The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

`false_value`
: False value

  The new variable (`new_var`) is set to the specified value for all by groups which occur in at least one source (`sources`) but the condition never evaluates to `TRUE`.

  The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

`missing_value`
: Values used for missing information

  The new variable is set to the specified value for all by groups without observations in any of the sources (`sources`).

  The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

Details

1. For each `flag_event()` object specified for `flag_events`: The condition (`condition`) is evaluated in the dataset referenced by `dataset_name`. If the `by_vars` field is specified the dataset is grouped by the specified variables for evaluating the condition. If named elements are used in `by_vars` like `by_vars = exprs(USUBJID, EXLNKID = ECLNKID)`, the variables are renamed after the evaluation. If the `by_vars` element is not specified, the observations are grouped by the variables specified for the `by_vars` argument.
2. The new variable (`new_var`) is added to the input dataset and set to the true value (`true_value`) if for the by group at least one condition evaluates to `TRUE` in one of the sources. 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`.

See Also

`flag_event()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`,

derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

Examples

```
library(dplyr)

# Derive a flag indicating anti-cancer treatment based on CM and PR
adsl <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3",
  "4"
)

cm <- tribble(
  ~USUBJID, ~CMCAT,         ~CMSEQ,
  "1",      "ANTI-CANCER",      1,
  "1",      "GENERAL",          2,
  "2",      "GENERAL",          1,
  "3",      "ANTI-CANCER",      1
)

# Assuming all records in PR indicate cancer treatment
pr <- tibble::tribble(
  ~USUBJID, ~PRSEQ,
  "2",      1,
  "3",      1
)

derive_var_merged_ef_msrc(
  adsl,
  by_vars = exprs(USUBJID),
  flag_events = list(
    flag_event(
      dataset_name = "cm",
      condition = CMCAT == "ANTI-CANCER"
    ),
    flag_event(
      dataset_name = "pr"
    )
  ),
  source_datasets = list(cm = cm, pr = pr),
  new_var = CANCTRFL
)

# Using different by variables depending on the source
# Add a dose adjustment flag to ADEX based on ADEX, EC, and FA
adex <- tribble(
  ~USUBJID, ~EXLNKID, ~EXADJ,
  "1",      "1",      "AE",
```

```
  "1",      "2",      NA_character_,
  "1",      "3",      NA_character_,
  "2",      "1",      NA_character_,
  "3",      "1",      NA_character_
)

ec <- tribble(
  ~USUBJID, ~ECLNKID, ~ECADJ,
  "1",      "3",      "AE",
  "3",      "1",      NA_character_
)

fa <- tribble(
  ~USUBJID, ~FALNKID, ~FATESTCD, ~FAOBJ,            ~FASTRESC,
  "3",      "1",      "OCCUR",   "DOSE ADJUSTMENT", "Y"
)

derive_var_merged_ef_msrc(
  adex,
  by_vars = exprs(USUBJID, EXLNKID),
  flag_events = list(
    flag_event(
      dataset_name = "ex",
      condition = !is.na(EXADJ)
    ),
    flag_event(
      dataset_name = "ec",
      condition = !is.na(ECADJ),
      by_vars = exprs(USUBJID, EXLNKID = ECLNKID)
    ),
    flag_event(
      dataset_name = "fa",
      condition = FATESTCD == "OCCUR" & FAOBJ == "DOSE ADJUSTMENT" & FASTRESC == "Y",
      by_vars = exprs(USUBJID, EXLNKID = FALNKID)
    )
  ),
  source_datasets = list(ex = adex, ec = ec, fa = fa),
  new_var = DOSADJFL
)
```

`derive_var_merged_exist_flag`

Merge an Existence Flag

Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in another dataset fulfilling a certain condition.

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

Usage

```
derive_var_merged_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  new_var,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
dataset_add	Additional dataset The variables specified by the by_vars argument are expected.
by_vars	Grouping variables
new_var	New variable The specified variable is added to the input dataset.
condition	Condition The condition is evaluated at the additional dataset (dataset_add). For all by groups where it evaluates as TRUE at least once the new variable is set to the true value (true_value). For all by groups where it evaluates as FALSE or NA for all observations the new variable is set to the false value (false_value). The new variable is set to the missing value (missing_value) for by groups not present in the additional dataset.
true_value	True value
false_value	False value
missing_value	Value used for missing information The new variable is set to the specified value for all by groups without observations in the additional dataset.
filter_add	Filter for additional data Only observations fulfilling the specified condition are taken into account for flagging. If the argument is not specified, all observations are considered.

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The new variable is added to the input dataset and set to the true value (true_value) if for the by group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE. It is set to the false value (false_value) if for the by group at

least one observation exists and for all observations the condition evaluates to FALSE or NA. Otherwise, it is set to the missing value (missing_value).

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

Examples

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

dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1028",   71, "YEARS",
  "PILOT01",    "DM", "04-1127",   84, "YEARS",
  "PILOT01",    "DM", "06-1049",   60, "YEARS"
)

ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,    ~AETERM,     ~AEREL,
  "PILOT01",    "AE", "01-1028", "ERYTHEMA", "POSSIBLE",
  "PILOT01",    "AE", "01-1028", "PRURITUS", "PROBABLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "POSSIBLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "PROBABLE"
)


derive_var_merged_exist_flag(
  dm,
  dataset_add = ae,
  by_vars = exprs(STUDYID, USUBJID),
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, AERELFL)

vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,      ~VISIT, ~VSTESTCD, ~VSSTRESN, ~VSBLFL,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "HEIGHT",     177.8,      NA,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "01-1028",  "BASELINE",  "WEIGHT",     99.34,     "Y",
  "PILOT01",    "VS", "01-1028",    "WEEK 4",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "HEIGHT",     165.1,      NA,
```

```
  "PILOT01",    "VS", "04-1127", "SCREENING",  "WEIGHT",     42.87,      NA,
  "PILOT01",    "VS", "04-1127",  "BASELINE",  "WEIGHT",     41.05,     "Y",
  "PILOT01",    "VS", "04-1127",    "WEEK 4",  "WEIGHT",     41.73,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "HEIGHT",    167.64,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "WEIGHT",     57.61,      NA,
  "PILOT01",    "VS", "06-1049",  "BASELINE",  "WEIGHT",     57.83,     "Y",
  "PILOT01",    "VS", "06-1049",    "WEEK 4",  "WEIGHT",     58.97,      NA
)
derive_var_merged_exist_flag(
  dm,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  false_value = "N",
  missing_value = "M"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WTBLHIFL)
```

`derive_var_merged_summary`
Merge Summary Variables

Description

[Deprecated] The `derive_var_merged_summary()` function has been deprecated in favor of `derive_vars_merged_summar`

Usage

```
derive_var_merged_summary(
  dataset,
  dataset_add,
  by_vars,
  new_vars = NULL,
  filter_add = NULL,
  missing_values = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.
`dataset_add`	Additional dataset The variables specified by the `by_vars` and the variables used on the left hand sides of the `new_vars` arguments are expected.

by_vars
: Grouping variables

  The expressions on the left hand sides of new_vars are evaluated by the specified *variables*. Then the resulting values are merged to the input dataset (dataset) by the specified *variables*.

new_vars
: New variables to add

  The specified variables are added to the input dataset.

  A named list of expressions is expected:

  - LHS refer to a variable.
  - RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for by_vars. Any expression on the RHS must result in a single value per by group.

  For example:

```
new_vars = exprs(
  DOSESUM = sum(AVAL),
  DOSEMEAN = mean(AVAL)
)
```

filter_add
: Filter for additional dataset (dataset_add)

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

missing_values
: Values for non-matching observations

  For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

Details

1. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
2. The new variables (new_vars) are created for each by group (by_vars) in the additional dataset (dataset_add) by calling summarize(). I.e., all observations of a by group are summarized to a single observation.
3. The new variables are merged to 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.

See Also

derive_summary_records(), get_summary_records()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), dthcaus_source(), get_summary_records()

---

derive_var_nfrlt *Derive Nominal Relative Time from First Dose (NFRLT)*

Description

[Experimental]

Derives nominal/planned time from first dose in hours by combining visit day information with timepoint descriptions. The function converts timepoint strings to hours using convert_xxtpt_to_hours() and adds them to the day-based offset. Optionally creates a corresponding unit variable.

Usage

```
derive_var_nfrlt(
  dataset,
  new_var = NFRLT,
  new_var_unit = NULL,
  out_unit = "HOURS",
  tpt_var = NULL,
  visit_day,
  first_dose_day = 1,
  treatment_duration = 0,
  range_method = "midpoint",
  set_values_to_na = NULL
)
```

Arguments

dataset	Input dataset containing visit day variable and optionally timepoint variable.
new_var	Name of the new variable to create (unquoted). Default is NFRLT.
new_var_unit	Name of the unit variable to create (unquoted). If specified, a character variable will be created containing the unit of time exactly as provided in out_unit. Common CDISC variables are FRLTU (First Dose Relative Time Unit) or RRLTU (Reference Relative Time Unit). If not specified, no unit variable is created.
out_unit	Unit of time for the output variable. Options are: • Days: "day", "days", "d" • Hours: "hour", "hours", "hr", "hrs", "h" (default: "hours") • Minutes: "minute", "minutes", "min", "mins" • Weeks: "week", "weeks", "wk", "wks", "w"

Case-insensitive. The internal calculation is performed in hours, then converted to the specified unit. If `new_var_unit` is specified, it will contain the value exactly as provided by the user.

`tpt_var` Timepoint variable containing descriptions like "Pre-dose", "1H Post-dose", etc. (unquoted). If not provided or if the variable doesn't exist in the dataset, only the visit day offset is calculated (timepoint contribution is 0).

`visit_day` Visit day variable (unquoted). This should be the planned/ nominal visit day (e.g., `VISITDY`). Records with `NA` in this variable will have NFRLT set to `NA`.

`first_dose_day` The day number considered as the first dose day. Default is 1. For multiple-dose studies, this is typically Day 1.

`treatment_duration`

Duration of treatment in hours. Can be either:

- A numeric scalar (used for all records), or
- An unquoted variable name from the dataset (e.g., `EXDUR`) where each record can have a different treatment duration

Passed to `convert_xxtpt_to_hours()`. Must be non-negative. Default is 0 hours (for instantaneous treatments like oral medications).

`range_method` Method for converting time ranges to single values. Options are "midpoint" (default), "start", or "end". Passed to `convert_xxtpt_to_hours()`. For example, "0-6h" with midpoint returns 3, with start returns 0, with end returns 6.

`set_values_to_na`

An optional condition that marks derived NFRLT values as `NA`. For example, `set_values_to_na = VISIT == "UNSCHEDULED"` will set NFRLT to `NA` for all unscheduled visits. Can use any variables in the dataset. When `new_var_unit` is specified, the unit variable will also be set to `NA` for these records.

Details

The nominal relative time is calculated as:

`NFRLT = (day_offset * 24 + timepoint_hours) * conversion_factor`

Where:

- `day_offset` is calculated from `visit_day` and `first_dose_day`, accounting for the absence of Day 0 in clinical trial convention
- `timepoint_hours` is derived from the timepoint description using `convert_xxtpt_to_hours()`, or 0 if `tpt_var` is not provided
- `conversion_factor` is:
  - 1 for "hours" (default)
  - 1/24 for "days"
  - 1/168 for "weeks" (1/24/7)
  - 60 for "minutes"

If `new_var_unit` is specified, a character variable is created containing the value of `out_unit` exactly as provided by the user. For example:

- `out_unit = "hours"` creates unit variable with value "hours"

- `out_unit = "HOURS"` creates unit variable with value "HOURS"
- `out_unit = "Days"` creates unit variable with value "Days"
- `NA` when the corresponding time value is `NA`

This matches the behavior of `derive_vars_duration()` and allows consistency when deriving multiple time variables.

Handling "No Day 0":

In clinical trials, day numbering typically follows the convention: ..., Day -2, Day -1, Day 1, Day 2, ... (no Day 0). This function accounts for this by adjusting the day offset when `visit_day` is negative and `first_dose_day` is positive.

For example, with `first_dose_day = 1` and different output units:

- Day -1, `out_unit = "hours"` -> -24 hours
- Day -1, `out_unit = "days"` -> -1 day
- Day -1, `out_unit = "weeks"` -> -0.1429 weeks
- Day -1, `out_unit = "minutes"` -> -1440 minutes
- Day -7 -> -168 hours, -7 days, -1 week, or -10080 minutes
- Day 1 -> 0 (in any unit, first dose day)
- Day 8 -> 168 hours, 7 days, 1 week, or 10080 minutes

With `first_dose_day = 7`:

- Day -1 -> -168 hours, -7 days, -1 week, or -10080 minutes
- Day 1 -> -144 hours, -6 days, -0.857 weeks, or -8640 minutes
- Day 6 -> -24 hours, -1 day, -0.143 weeks, or -1440 minutes
- Day 7 -> 0 (in any unit, first dose day)

Common Use Cases:

- **Single dose study**: Day 1 only, with samples at various timepoints (e.g., Pre-dose, 1H, 2H, 4H, 8H, 24H)
- **Multiple dose study**: Dosing on multiple days (e.g., Day 1, Day 8, Day 15) with samples around each dose
- **Screening visits**: Negative visit days (e.g., Day -14, Day -7) before first dose
- **Steady state study**: Multiple daily doses with sampling on specific days
- **Oral medications**: Use default `treatment_duration = 0` for instantaneous absorption
- **IV infusions**: Specify `treatment_duration` as infusion duration in hours (scalar) or as a variable name containing duration per record
- **Exposure records (EX)**: Can be called without `tpt_var` to derive NFRLT based only on visit day
- **Unscheduled visits**: Use `set_values_to_na` to set NFRLT to `NA` for unscheduled or early discontinuation visits
- **Variable treatment durations**: Use a variable name (e.g., `EXDUR`) when different subjects or visits have different treatment durations

- **Hours output**: Use `out_unit = "hours"` (default) for variables like `NFRLT` with `FRLTU`
- **Days output**: Use `out_unit = "days"` for variables like `NFRLTDY` with `FRLTU`
- **Weeks output**: Use `out_unit = "weeks"` for long-term studies with weekly dosing
- **Minutes output**: Use `out_unit = "minutes"` for very short-term PK studies or when minute precision is needed
- **CDISC compliance**: Use `new_var_unit = FRLTU` for first dose relative time or `new_var_unit = RRLTU` for reference relative time
- **Consistency with duration**: Use the same case for `out_unit` across `derive_vars_duration()` and `derive_var_nfrlt()` to ensure unit variables match

Important Notes:

- The function assumes `visit_day` represents the nominal/planned day, not the actual study day
- Day numbering follows clinical trial convention with no Day 0
- For timepoints that span multiple days (e.g., "24H Post-dose"), ensure `visit_day` is set to the day when the sample was taken. For example, if dosing occurs on Day 3, a "24H Post-dose" sample taken on Day 4 should have `visit_day = 4`.
- For crossover studies, consider deriving NFRLT separately per period
- `NA` values in `visit_day` will automatically result in `NA` for NFRLT (no need to use `set_values_to_na` for this case)
- `NA` values in `tpt_var` will result in `NA` for NFRLT
- `NA` values in the `treatment_duration` variable (if using a variable) will result in `NA` for NFRLT for those records
- Use `set_values_to_na` when you need to set NFRLT to `NA` based on other variables (e.g., `VISIT == "UNSCHEDULED"`), especially when `visit_day` is populated but should not be used for the NFRLT calculation
- If `tpt_var` is not provided or doesn't exist in the dataset, timepoint contribution is assumed to be 0 hours
- When using non-hour units, timepoint contributions are still calculated in hours first (e.g., "2H Post-dose" = 2 hours), then the entire result is converted to the specified unit
- The unit variable (if created) will contain the exact value provided in `out_unit`, preserving case and format

Setting Special Values:

If you need to set NFRLT to a specific value (e.g., 99999) for certain visits instead of `NA`, use `set_values_to_na` first to set them to `NA`, then use a subsequent `mutate()` call to replace those `NA` values:

```
dataset %>%
  derive_var_nfrlt(
    ...,
    set_values_to_na = VISIT == "UNSCHEDULED"
  ) %>%
  mutate(NFRLT = if_else(is.na(NFRLT) & VISIT == "UNSCHEDULED", 99999, NFRLT))
```

Value

The input dataset with the new nominal relative time variable added, and optionally the unit variable if new_var_unit is specified.

See Also

convert_xxtpt_to_hours(), derive_vars_duration()

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

---

derive_var_obs_number *Adds a Variable Numbering the Observations Within Each By Group*

Description

Adds a variable numbering the observations within each by group

Usage

```
derive_var_obs_number(
  dataset,
  by_vars = NULL,
  order = NULL,
  new_var = ASEQ,
  check_type = "none"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset.
by_vars	Grouping variables
order	Sort order Within each by group the observations are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
new_var	Name of variable to create The new variable is set to the observation number for each by group. The numbering starts with 1.
check_type	Check uniqueness? If "message", "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.

Details

For each group (with respect to the variables specified for the by_vars parameter) the first or last observation (with respect to the order specified for the order parameter and the mode specified for the mode parameter) is included in the output dataset.

Value

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

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

```
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID,  ~DOMAIN,      ~USUBJID, ~VSTESTCD, ~VISITNUM, ~VSTPTNUM,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         4,       816,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         4,       816,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         4,       816
)
vs %>%
  derive_var_obs_number(
    by_vars = exprs(USUBJID, VSTESTCD),
```

```
    order = exprs(VISITNUM, desc(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,
  ignore_time_for_ref_end_date = TRUE,
  filter_pre_timepoint = NULL,
  span_period = FALSE
)
```

Arguments

`dataset`	Input dataset Required columns are `start_date`, `end_date`, `ref_start_date` and `ref_end_date`.
`new_var`	On-treatment flag variable name to be created.
`start_date`	The start date (e.g. `AESDT`) or assessment date (e.g. `ADT`) Required; A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
`end_date`	The end date of assessment/event (e.g. `AENDT`) A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object. Optional; Default is null. If the used and date value is missing on an observation, it is assumed the medication is ongoing and `ONTRTFL` is set to `"Y"`.
`ref_start_date`	The lower bound of the on-treatment period Required; A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

ref_end_date — The upper bound of the on-treatment period A date or date-time object column is expected.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

If set to NULL, everything after ref_start_date will be considered on-treatment.

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)

ignore_time_for_ref_end_date

If the argument is set to TRUE, the time part is ignored for checking if the event occurred more than ref_end_window days after reference end date.

filter_pre_timepoint

An expression to filter observations as not on-treatment when date = ref_start_date. For example, if observations where VSTPT = PRE should not be considered on-treatment when date = ref_start_date, filter_pre_timepoint should be used to denote when the on-treatment flag should be set to null. Optional; default is NULL.

span_period — A logical scalar. If TRUE, events that started prior to the ref_start_dateand are ongoing or end after the ref_start_date are flagged as "Y". Optional; default is FALSE.

Details

On-Treatment is calculated by determining whether the assessment date or start/stop dates fall between 2 dates. The following logic is used to assign on-treatment = "Y":

1. start_date is missing and ref_start_dateis non-missing
2. No timepoint filter is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date
3. A timepoint is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date and the filter provided in filter_pre_timepoint is not true.
4. ref_end_date is not provided and ref_start_date < start_date
5. ref_end_date is provided and ref_start_date < start_date <= ref_end_date + ref_end_window.

If the end_date is provided and the end_date < ref_start_date then the ONTRTFL is set to NULL.This would be applicable to cases where the start_date is missing and ONTRTFL has been assigned as "Y" above.

If the span_period is TRUE, this allows the user to assign ONTRTFL as "Y" to cases where the record started prior to the ref_start_date and was ongoing or ended after the ref_start_date.

Any date imputations needed should be done prior to calling this function.

Value

The input dataset with an additional column named ONTRTFL with a value of "Y" or NA

See Also

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

Examples

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

advs <- tribble(
  ~USUBJID, ~ADT,              ~TRTSDT,           ~TRTEDT,
  "P01",    ymd("2020-02-24"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P02",    ymd("2020-01-01"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P03",    ymd("2019-12-31"), ymd("2020-01-01"), ymd("2020-03-01")
)
derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT
)

advs <- tribble(
  ~USUBJID, ~ADT,              ~TRTSDT,           ~TRTEDT,
  "P01",    ymd("2020-07-01"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P02",    ymd("2020-04-30"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P03",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01")
)
derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  ref_end_window = 60
)

advs <- tribble(
  ~USUBJID, ~ADTM,                      ~TRTSDTM,                   ~TRTEDTM,
 "P01",   ymd_hm("2020-01-02T12:00"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
 "P02",   ymd("2020-01-01"),       ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
 "P03",   ymd("2019-12-31"),       ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
) %>%
  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 = TRUE
)

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 = TRUE
)
```

`derive_var_pchg` *Derive Percent Change from Baseline*

Description

Derive percent change from baseline (PCHG) in a BDS dataset

Usage

```
derive_var_pchg(dataset)
```

Arguments

dataset	Input dataset AVAL and BASE are expected.

Details

Percent change from baseline is calculated by dividing change from baseline by the absolute value of the baseline value and multiplying the result by `100`.

Value

The input dataset with an additional column named `PCHG`

See Also

`derive_var_chg()`

BDS-Findings Functions that returns variable appended to dataset: `derive_basetype_records()`, `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_atoxgr_dir()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_nfrlt()`, `derive_var_ontrtfl()`, `derive_var_shift()`, `derive_vars_crit_flag()`

Examples

```
library(tibble)

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

---

derive_var_relative_flag

Flag Observations Before or After a Condition is Fulfilled

Description

Flag all observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to flag for each subject all observations before the first disease progression or to flag all AEs after a specific AE.

Usage

```
derive_var_relative_flag(
  dataset,
  by_vars,
  order,
  new_var,
```

```
  condition,
  mode,
  selection,
  inclusive,
  flag_no_ref_groups = TRUE,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset.
by_vars	Grouping variables
order	Sort order Within each by group the observations are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
new_var	New variable The variable is added to the input dataset and set to "Y" for all observations before or after the condition is fulfilled. For all other observations it is set to NA.
condition	Condition for Reference Observation The specified condition determines the reference observation. In the output dataset all observations before or after (selection argument) the reference observation are flagged.
mode	Selection mode (first or last) If "first" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the *first* time is flagged in the output dataset. If "last" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the *last* time is flagged in the output dataset.
selection	Flag observations before or after the reference observation?
inclusive	Flag the reference observation?
flag_no_ref_groups	Should by groups without reference observation be flagged?
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.

Details

For each by group (by_vars argument) the observations before or after (selection argument) the observations where the condition (condition argument) is fulfilled the first or last time (order argument and mode argument) is flagged in the output dataset.

Value

The input dataset with the new variable (new_var) added

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

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

# Flag all AEs after the first COVID AE
adae <- tribble(
  ~USUBJID, ~ASTDY, ~ACOVFL, ~AESEQ,
  "1",           2, NA,           1,
  "1",           5, "Y",          2,
  "1",           5, NA,           3,
  "1",          17, NA,           4,
  "1",          27, "Y",          5,
  "1",          32, NA,           6,
  "2",           8, NA,           1,
  "2",          11, NA,           2,
)

derive_var_relative_flag(
  adae,
  by_vars = exprs(USUBJID),
  order = exprs(ASTDY, AESEQ),
  new_var = PSTCOVFL,
  condition = ACOVFL == "Y",
  mode = "first",
  selection = "after",
  inclusive = FALSE,
  flag_no_ref_groups = FALSE
)

response <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      0,        "PR",
  "1",      1,        "CR",
  "1",      2,        "CR",
  "1",      3,        "SD",
  "1",      4,        "NE",
  "2",      0,        "SD",
  "2",      1,        "PD",
  "2",      2,        "PD",
  "3",      0,        "SD",
```

```
  "4",      0,        "SD",
  "4",      1,        "PR",
  "4",      2,        "PD",
  "4",      3,        "SD",
  "4",      4,        "PR"
)

# Flag observations up to first PD for each patient
response %>%
  derive_var_relative_flag(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    new_var = ANL02FL,
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Flag observations up to first PD excluding baseline (AVISITN = 0) for each patient
response %>%
  restrict_derivation(
    derivation = derive_var_relative_flag,
    args = params(
      by_vars = exprs(USUBJID),
      order = exprs(AVISITN),
      new_var = ANL02FL,
      condition = AVALC == "PD",
      mode = "first",
      selection = "before",
      inclusive = TRUE
    ),
    filter = AVISITN > 0
  ) %>%
  arrange(USUBJID, AVISITN)
```

`derive_var_shift` *Derive Shift*

Description

Derives a character shift variable containing concatenated shift in values based on user-defined pairing, e.g., shift from baseline to analysis value, shift from baseline grade to analysis grade, ...

Usage

```
derive_var_shift(
  dataset,
  new_var,
  from_var,
```

```
  to_var,
  missing_value = "NULL",
  sep_val = " to "
)
```

Arguments

dataset	Input dataset The variables specified by the from_var and to_var arguments are expected to be in the dataset.
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.
missing_value	Character string to replace missing values in from_var or to_var.
sep_val	Character string to concatenate values of from_var and to_var.

Details

new_var is derived by concatenating the values of from_var to values of to_var (e.g. "NORMAL to HIGH"). When from_var or to_var has missing value, the missing value is replaced by missing_value (e.g. "NORMAL to NULL").

Value

The input dataset with the character shift variable added

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_basetype_records(), derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_vars_crit_flag()

Examples

```
library(tibble)

data <- 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`).

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

Usage

```
derive_var_trtdurd(dataset, start_date = TRTSDT, end_date = TRTEDT)
```

Arguments

`dataset`	Input dataset The variables specified by the `start_date` and `end_date` arguments are expected to be in the dataset.
`start_date`	The start date A date or date-time object is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.
`end_date`	The end date A date or date-time object is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Details

The total treatment duration is derived as the number of days from start to end date plus one.

Value

The input dataset with TRTDURD added

See Also

derive_vars_duration()

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

Examples

```
library(tibble)
library(lubridate)

data <- tribble(
  ~TRTSDT, ~TRTEDT,
  ymd("2020-01-01"), ymd("2020-02-24")
)

derive_var_trtdurd(data)
```

---

derive_var_trtemfl *Derive Treatment-emergent Flag*

Description

Derive treatment emergent analysis flag (e.g., TRTEMFL).

Usage

```
derive_var_trtemfl(
  dataset,
  new_var = TRTEMFL,
  start_date = ASTDTM,
  end_date = AENDTM,
  trt_start_date = TRTSDTM,
  trt_end_date = NULL,
  end_window = NULL,
  ignore_time_for_trt_end = TRUE,
  initial_intensity = NULL,
  intensity = NULL,
  group_var = NULL,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

dataset	Input dataset The variables specified by start_date, end_date, trt_start_date, trt_end_date, initial_intensity, and intensity are expected.
new_var	New variable
start_date	Event start date
end_date	Event end date
trt_start_date	Treatment start date
trt_end_date	Treatment end date
end_window	If the argument is specified (in 'days'), events starting more than the specified number of days after end of treatment, are not flagged.
ignore_time_for_trt_end	If the argument is set to TRUE, the time part is ignored for checking if the event occurred more than end_window days after end of treatment.
initial_intensity	Initial severity/intensity or toxicity initial_intensity is ignored when group_var is specified. If this argument is specified and group_var is NULL, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged. The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.
intensity	Severity/intensity or toxicity If the argument is specified, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged. The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.
group_var	Grouping variable If the argument is specified, it assumes that AEs are recorded as one episode of AE with multiple lines using a grouping variable. Events starting during treatment or before treatment and worsening afterward are flagged. Once an AE record in a group is flagged, all subsequent records in the treatment window are flagged regardless of severity.
subject_keys	Variables to uniquely identify a subject. This argument is only used when group_var is specified.

Details

For the derivation of the new variable the following cases are considered in this order. The first case which applies, defines the value of the variable.

- *not treated*: If trt_start_date is NA, it is set to NA_character_.
- *event before treatment*: If end_date is before trt_start_date (and end_date is not NA), it is set to NA_character_.
- *no event date*: If start_date is NA, it is set to "Y" as in such cases it is usually considered more conservative to assume the event was treatment-emergent.
- *event started during treatment*:
  - if end_window is not specified: if start_date is on or after trt_start_date, it is set to "Y",
  - if end_window is specified: if start_date is on or after trt_start_date and start_date is on or before trt_end_date + end_window days, it is set to "Y",
- *event started before treatment and (possibly) worsened on treatment*:
  - if initial_intensity, intensity is specified and group_var is not specified: if initial_intensity < intensity and start_date is before trt_start_date and end_date is on or after trt_start_date or end_date is NA, it is set to "Y";
  - if group_var is specified: if intensity at treatment start < intensity and start_date is after trt_start_date and end_date is on or after trt_start_date or end_date is NA, it is set to "Y";
- Otherwise it is set to NA_character_.

The behavior of derive_var_trtemfl() is aligned with the proposed treatment-emergent AE assignment in the following PHUSE White Paper. See the final example in the examples section below.

Value

The input dataset with the variable specified by new_var added

See Also

OCCDS Functions: derive_vars_atc(), derive_vars_query()

---

desc *dplyr desc*

Description

See dplyr::desc for details.

dose_freq_lookup *Pre-Defined Dose Frequencies*

Description

These pre-defined dose frequencies are sourced from CDISC. The number of rows to generate using `create_single_dose_dataset()` arguments `start_date` and `end_date` is derived from `DOSE_COUNT`, `DOSE_WINDOW`, and `CONVERSION_FACTOR` with appropriate functions from `lubridate`.

Usage

```
dose_freq_lookup
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 86 rows and 5 columns.

Details

`NCI_CODE` and `CDISC_VALUE` are included from the CDISC source for traceability.

`DOSE_COUNT` represents the number of doses received in one single unit of `DOSE_WINDOW`. For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `DOSE_WINDOW=="MONTH"` and `DOSE_COUNT==10`. Similarly, for `CDISC_VALUE=="EVERY 2 WEEKS"`, `DOSE_WINDOW=="WEEK"` and `DOSE_COUNT==0.5` (to yield one dose every two weeks).

`CONVERSION_FACTOR` is used to convert `DOSE_WINDOW` units `"WEEK"`, `"MONTH"`, and `"YEAR"` to the unit `"DAY"`.

For example, for `CDISC_VALUE=="10 DAYS PER MONTH"`, `CONVERSION_FACTOR` is `0.0329`. One day of a month is assumed to be `1 / 30.4375` of a month (one day is assumed to be `1/365.25` of a year). Given only `start_date` and `end_date` in the aggregate dataset, `CONVERSION_FACTOR` is used to calculate specific dates for`start_date` and `end_date` in the resulting single dose dataset for the doses that occur. In such cases, doses are assumed to occur at evenly spaced increments over the interval.

To see the entire table in the console, run `print(dose_freq_lookup)`.

See Also

`create_single_dose_dataset()`

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `country_code_lookup`

dthcaus_source *Create a* dthcaus_source *Object*

Description

[Deprecated] The dthcaus_source() function and dthcaus_source() have been deprecated in favor of event().

Usage

```
dthcaus_source(
  dataset_name,
  filter,
  date,
  order = NULL,
  mode = "first",
  dthcaus,
  set_values_to = NULL
)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the death cause.
filter	An expression used for filtering dataset.
date	A date or datetime variable or an expression to be used for sorting dataset.
order	Sort order Additional variables/expressions to be used for sorting the dataset. The dataset is ordered by date and order. Can be used to avoid duplicate record warning.
mode	One of "first" or "last". Either the "first" or "last" observation is preserved from the dataset which is ordered by date.
dthcaus	A variable name, an expression, or a string literal If a variable name is specified, e.g., AEDECOD, it is the variable in the source dataset to be used to assign values to DTHCAUS; if an expression, e.g., str_to_upper(AEDECOD), it is evaluated in the source dataset and the results is assigned to DTHCAUS; if a string literal, e.g. "Adverse Event", it is the fixed value to be assigned to DTHCAUS.
set_values_to	Variables to be set to trace the source dataset

Value

An object of class "dthcaus_source".

See Also

derive_var_dthcaus()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), get_summary_records()

Examples

```
# Deaths sourced from AE
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDT,
  mode = "first",
  dthcaus = AEDECOD
)

# Deaths sourced from DS
src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH",
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)
```

---

event *Create a* event *Object*

Description

The event object is used to define events as input for the derive_extreme_event() and derive_vars_extreme_event() functions.

Usage

```
event(
  dataset_name = NULL,
  condition = NULL,
  mode = NULL,
  order = NULL,
  set_values_to = NULL,
  keep_source_vars = NULL,
  description = NULL
)
```

Arguments

dataset_name	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for source_datasets in derive_extreme_event(). If the argument is not specified, the input dataset (dataset) of derive_extreme_event() is used.
condition	An unquoted condition for selecting the observations, which will contribute to the extreme event. If the condition contains summary functions like all(), they are evaluated for each by group separately.

mode — If specified, the first or last observation with respect to order is selected for each by group.

order — The specified variables or expressions are used to select the first or last observation if mode is specified.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

set_values_to — A named list returned by exprs() defining the variables to be set for the event, e.g. exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems"). The values can be a symbol, a character string, a numeric value, NA or an expression.

keep_source_vars

Variables to keep from the source dataset

The specified variables are kept for the selected observations. The variables specified for by_vars (of derive_extreme_event()) and created by set_values_to are always kept.

description — Description of the event

The description does not affect the derivations where the event is used. It is intended for documentation only.

Value

An object of class event

See Also

derive_extreme_event(), derive_vars_extreme_event(), event_joined()

Source Objects: basket_select(), censor_source(), death_event, event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

---

event_joined *Create a* event_joined *Object*

Description

The event_joined object is used to define events as input for the derive_extreme_event() and derive_vars_extreme_event() functions. This object should be used if the event does not depend on a single observation of the source dataset but on multiple observations. For example, if the event needs to be confirmed by a second observation of the source dataset.

The events are selected by calling filter_joined(). See its documentation for more details.

Usage

```
event_joined(
  dataset_name = NULL,
  condition,
  order = NULL,
  join_vars,
```

```
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  set_values_to = NULL,
  keep_source_vars = NULL,
  description = NULL
)
```

Arguments

dataset_name	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for source_datasets in derive_extreme_event(). If the argument is not specified, the input dataset (dataset) of derive_extreme_event() is used.
condition	An unquoted condition for selecting the observations, which will contribute to the extreme event. The condition is applied to the joined dataset for selecting the confirmed observations. The condition can include summary functions like all() or any(). 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, condition = 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".
order	If specified, the specified variables or expressions are used to select the first observation. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
join_vars	Variables to keep from joined dataset The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to select all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit join_vars = exprs(AVALC, AVISITN) and condition = 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.
first_cond_lower	Condition for selecting range of data (before) If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. 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 condition contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation.

first_cond_upper

Condition for selecting range of data (after)

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 condition contains summary functions which should not apply to all observations but only up to the confirmation assessment.

set_values_to A named list returned by exprs() defining the variables to be set for the event, e.g. exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems"). The values can be a symbol, a character string, a numeric value, NA or an expression.

keep_source_vars

Variables to keep from the source dataset

The specified variables are kept for the selected observations. The variables specified for by_vars (of derive_extreme_event()) and created by set_values_to are always kept.

description Description of the event

The description does not affect the derivations where the event is used. It is intended for documentation only.

Value

An object of class event_joined

See Also

derive_extreme_event(), derive_vars_extreme_event(), event()

Source Objects: basket_select(), censor_source(), death_event, event(), event_source(), flag_event(), query(), records_source(), tte_source()

Examples

```
library(tibble)
library(dplyr)
library(lubridate)
# Derive confirmed best overall response (using event_joined())
# CR - complete response, PR - partial response, SD - stable disease
# NE - not evaluable, PD - progressive disease
adsl <- tribble(
  ~USUBJID, ~TRTSDTC,
  "1",      "2020-01-01",
  "2",      "2019-12-12",
  "3",      "2019-11-11",
  "4",      "2019-12-30",
```

```
  "5",      "2020-01-01",
  "6",      "2020-02-02",
  "7",      "2020-02-02",
  "8",      "2020-02-01"
) %>%
  mutate(TRTSDT = ymd(TRTSDTC))

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

derive_extreme_event(
  adrs,
  by_vars = exprs(USUBJID),
  order = exprs(ADT),
  mode = "first",
  source_datasets = list(adsl = adsl),
  events = list(
    event_joined(
      description = paste(
```

```
    "CR needs to be confirmed by a second CR at least 28 days later",
    "at most one NE is acceptable between the two assessments"
  ),
  join_vars = exprs(AVALC, ADT),
  join_type = "after",
  first_cond_upper = AVALC.join == "CR" &
    ADT.join >= ADT + 28,
  condition = AVALC == "CR" &
    all(AVALC.join %in% c("CR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1,
  set_values_to = exprs(
    AVALC = "CR"
  )
),
event_joined(
  description = paste(
    "PR needs to be confirmed by a second CR or PR at least 28 days later,",
    "at most one NE is acceptable between the two assessments"
  ),
  join_vars = exprs(AVALC, ADT),
  join_type = "after",
  first_cond_upper = AVALC.join %in% c("CR", "PR") &
    ADT.join >= ADT + 28,
  condition = AVALC == "PR" &
    all(AVALC.join %in% c("CR", "PR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1,
  set_values_to = exprs(
    AVALC = "PR"
  )
),
event(
  description = paste(
    "CR, PR, or SD are considered as SD if occurring at least 28",
    "after treatment start"
  ),
  condition = AVALC %in% c("CR", "PR", "SD") & ADT >= TRTSDT + 28,
  set_values_to = exprs(
    AVALC = "SD"
  )
),
event(
  condition = AVALC == "PD",
  set_values_to = exprs(
    AVALC = "PD"
  )
),
event(
  condition = AVALC %in% c("CR", "PR", "SD", "NE"),
  set_values_to = exprs(
    AVALC = "NE"
  )
),
event(
```

```
      description = "set response to MISSING for patients without records in ADRS",
      dataset_name = "adsl",
      condition = TRUE,
      set_values_to = exprs(
        AVALC = "MISSING"
      ),
      keep_source_vars = exprs(TRTSDT)
    )
  ),
  set_values_to = exprs(
    PARAMCD = "CBOR",
    PARAM = "Best Confirmed Overall Response by Investigator"
  )
) %>%
  filter(PARAMCD == "CBOR")
```

`event_source` *Create an* `event_source` *Object*

Description

`event_source` objects are used to define events as input for the `derive_param_tte()` function.

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

Usage

```
event_source(
  dataset_name,
  filter = NULL,
  date,
  set_values_to = NULL,
  order = NULL
)
```

Arguments

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

set_values_to	A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.
order	Sort order An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date.

Value

An object of class event_source, inheriting from class tte_source

See Also

derive_param_tte(), censor_source()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), flag_event(), query(), records_source(), tte_source()

Examples

```
# Death event

event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = exprs(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)
```

---

example_qs *Example* QS *Dataset*

Description

An example QS dataset based on the examples from the CDISC ADaM Supplements Generalized Anxiety Disorder 7-Item Version 2 (GAD-7) and Geriatric Depression Scale Short Form (GDS-SF).

Usage

```
example_qs
```

Format

An object of class tbl_df (inherits from tbl, data.frame) with 161 rows and 11 columns.

Source

Created by (https://github.com/pharmaverse/admiral/blob/main/data-raw/create_example_qs.R)

See Also

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

---

exprs *rlang exprs*

Description

See rlang::exprs for details.

---

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: get_flagged_records(), get_not_mapped(), get_vars_query()

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 16 columns.

Source

Derived from the ex dataset using `{admiral}` and `{dplyr}`

See Also

Other datasets: `admiral_adlb`, `admiral_adsl`, `example_qs`, `queries`, `queries_mh`

filter_exist *Returns records that fit into existing by groups in a filtered source dataset*

Description

Returns all records in the input dataset that belong to by groups that are present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return ADSL records for subjects that experienced a certain adverse event during the course of the study (as per records in ADAE).

Usage

```
filter_exist(dataset, dataset_add, by_vars, filter_add = NULL)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.

dataset_add	Source dataset The source dataset, which determines the by groups returned in the input dataset, based on the groups that exist in this dataset after being subset by filter_add. The variables specified in the by_vars and filter_add parameters are expected in this dataset.
by_vars	Grouping variables
filter_add	Filter for the source dataset The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed.

Details

Returns the records in dataset which match an existing by group in dataset_add, after being filtered according to filter_add. If there are no by groups that exist in both datasets, an empty dataset will be returned.

Value

The records in the input dataset which are contained within an existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: count_vals(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

Examples

```
# Get demographic information about subjects who have suffered from moderate or
# severe fatigue

library(tibble)

adsl <- tribble(
  ~USUBJID,      ~AGE, ~SEX,
  "01-701-1015", 63,   "F",
  "01-701-1034", 77,   "F",
  "01-701-1115", 84,   "M",
  "01-701-1146", 75,   "F",
  "01-701-1444", 63,   "M"
)

adae <- tribble(
  ~USUBJID,      ~AEDECOD,                    ~AESEV,     ~AESTDTC,
  "01-701-1015", "DIARRHOEA",                 "MODERATE", "2014-01-09",
  "01-701-1034", "FATIGUE",                   "SEVERE",   "2014-11-02",
  "01-701-1034", "APPLICATION SITE PRURITUS", "MODERATE", "2014-08-27",
  "01-701-1115", "FATIGUE",                   "MILD",     "2013-01-14",
  "01-701-1146", "FATIGUE",                   "MODERATE", "2013-06-03"
```

```
)

filter_exist(
  dataset = adsl,
  dataset_add = adae,
  by_vars = exprs(USUBJID),
  filter_add = AEDECOD == "FATIGUE" & AESEV %in% c("MODERATE", "SEVERE")
)
```

`filter_extreme` *Filter the First or Last Observation for Each By Group*

Description

Filters the first or last observation for each by group.

Usage

```
filter_extreme(dataset, by_vars = NULL, order, mode, check_type = "warning")
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset.
`by_vars`	Grouping variables
`order`	Sort order Within each by group the observations are ordered by the specified order.
`mode`	Selection mode (first or last) If `"first"` is specified, the first observation of each by group is included in the output dataset. If `"last"` is specified, the last observation of each by group is included in the output dataset.
`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.

Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is included in the output dataset.

Value

A dataset containing the first or last observation of each by group

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_joined(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

Examples

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

ex <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~EXSEQ, ~EXDOSE,    ~EXTRT,
  "PILOT01",    "EX", "01-1442",      1,      54,    "XANO",
  "PILOT01",    "EX", "01-1442",      2,      54,    "XANO",
  "PILOT01",    "EX", "01-1442",      3,      54,    "XANO",
  "PILOT01",    "EX", "01-1444",      1,      54,    "XANO",
  "PILOT01",    "EX", "01-1444",      2,      81,    "XANO",
  "PILOT01",    "EX", "05-1382",      1,      54,    "XANO",
  "PILOT01",    "EX", "08-1213",      1,      54,    "XANO",
  "PILOT01",    "EX", "10-1053",      1,      54,    "XANO",
  "PILOT01",    "EX", "10-1053",      2,      54,    "XANO",
  "PILOT01",    "EX", "10-1183",      1,       0, "PLACEBO",
  "PILOT01",    "EX", "10-1183",      2,       0, "PLACEBO",
  "PILOT01",    "EX", "10-1183",      3,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      1,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      2,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      3,       0, "PLACEBO",
  "PILOT01",    "EX", "14-1425",      1,      54,    "XANO",
  "PILOT01",    "EX", "15-1319",      1,      54,    "XANO",
  "PILOT01",    "EX", "15-1319",      2,      81,    "XANO",
  "PILOT01",    "EX", "16-1151",      1,      54,    "XANO",
  "PILOT01",    "EX", "16-1151",      2,      54,    "XANO"
)


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

# Select highest dose for each patient on the active drug
ex %>%
  filter(EXTRT != "PLACEBO") %>%
  filter_extreme(
    by_vars = exprs(USUBJID),
    order = exprs(EXDOSE),
    mode = "last",
    check_type = "none"
  ) %>%
  select(USUBJID, EXTRT, EXDOSE)
```

filter_joined *Filter Observations Taking Other Observations into Account*

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_joined(
  dataset,
  dataset_add,
  by_vars,
  join_vars,
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  order = NULL,
  tmp_obs_nr_var = NULL,
  filter_add = NULL,
  filter_join,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset.
dataset_add	Additional dataset The variables specified for by_vars, join_vars, and order are expected.
by_vars	By variables The specified variables are used as by variables for joining the input dataset with itself.

`join_vars` Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to select all observations with `AVALC == "Y"` and `AVALC == "Y"` for at least one subsequent visit `join_vars = exprs(AVALC, AVISITN)` and `filter_join = 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. Thus `join_type = "after"` could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus `join_type = "all"` could be used.

`first_cond_lower`

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. 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_join` contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For examples see the "Examples" section below.

`first_cond_upper`

Condition for selecting range of data (after)

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_join` contains summary functions which should not apply to all observations but only up to the confirmation assessment. For examples see the "Examples" section below.

`order` Order

The observations are ordered by the specified order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

`tmp_obs_nr_var` Temporary observation number

The specified variable is added to the input dataset (`dataset`) and the additional dataset (`dataset_add`). It is set to the observation number with respect to `order`. For each by group (`by_vars`) the observation number starts with `1`.

If there is more than one record for specific values for by_vars and order, all records get the same observation number. By default, a warning (see check_type) is issued in this case. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It is not included in the output dataset. It can also be used to select consecutive observations or the last observation (see example below).

filter_add — Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by the order argument can be used in the condition.

The condition can include summary functions. The additional dataset is grouped by the by variables (by_vars).

filter_join — Condition for selecting observations

The filter is applied to the joined dataset for selecting the confirmed observations. The condition can include summary functions like all() or any(). 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_join = 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 "message", "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.

Details

The following steps are performed to produce the output dataset.

Step 1:

- The variables specified by order are added to the additional dataset (dataset_add).
- The variables specified by join_vars are added to the additional dataset (dataset_add).
- The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.

Then the input dataset (dataset) is joined with the restricted additional dataset by the variables specified for by_vars. From the additional dataset only the variables specified for join_vars are kept. The suffix ".join" is added to those variables which are also present in the input dataset.

For example, for by_vars = USUBJID, join_vars = exprs(AVISITN, AVALC) and input dataset and additional 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 = exprs(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_lower` is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where `first_cond_lower` is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

If `first_cond_upper` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond_upper` 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.

For an example see the last example in the "Examples" section.

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_join`.

Step 5:

The first observation of each group is selected and the `*.join` variables are dropped.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

A subset of the observations of the input dataset. All variables of the input dataset are included in the output dataset.

See Also

count_vals(), min_cond(), max_cond()

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

`filter_not_exist` *Returns records that don't fit into existing by groups in a filtered source dataset*

Description

Returns all records in the input dataset that belong to by groups that are not present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return ADSL records for subjects that didn't take certain concomitant medications during the course of the study (as per records in ADCM).

Usage

```
filter_not_exist(dataset, dataset_add, by_vars, filter_add = NULL)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.
`dataset_add`	Source dataset The source dataset, which determines the by groups returned in the input dataset, based on the groups that don't exist in this dataset after being subset by `filter_add`. The variables specified in the `by_vars` and `filter_add` parameters are expected in this dataset.
`by_vars`	Grouping variables
`filter_add`	Filter for the source dataset The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed.

Details

Returns the records in `dataset` which don't match any existing by groups in `dataset_add`, after being filtered according to `filter_add`. If all by groups that exist in `dataset` don't exist in `dataset_add`, an empty dataset will be returned.

Value

The records in the input dataset which are not contained within any existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_relative(), max_cond(), min_cond()

Examples

```
# Get demographic information about subjects who didn't take vitamin supplements
# during the study

library(tibble)

adsl <- tribble(
  ~USUBJID,      ~AGE, ~SEX,
  "01-701-1015", 63,   "F",
  "01-701-1023", 64,   "M",
  "01-701-1034", 77,   "F",
  "01-701-1118", 52,   "M"
)

adcm <- tribble(
  ~USUBJID,      ~CMTRT,          ~CMSTDTC,
  "01-701-1015", "ASPIRIN",       "2013-05-14",
  "01-701-1023", "MYLANTA",       "2014-01-04",
  "01-701-1023", "CALCIUM",       "2014-02-25",
  "01-701-1034", "VITAMIN C",     "2013-12-12",
  "01-701-1034", "CALCIUM",       "2013-03-27",
  "01-701-1118", "MULTIVITAMIN", "2013-02-21"
)

filter_not_exist(
  dataset = adsl,
  dataset_add = adcm,
  by_vars = exprs(USUBJID),
  filter_add = str_detect(CMTRT, "VITAMIN")
)
```

---

filter_relative *Filter the Observations Before or After a Condition is Fulfilled*

Description

Filters the observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to select for each subject all observations before the first disease progression.

Usage

```
filter_relative(
  dataset,
  by_vars,
  order,
  condition,
  mode,
  selection,
  inclusive,
  keep_no_ref_groups = TRUE,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset.
by_vars	Grouping variables
order	Sort order Within each by group the observations are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").
condition	Condition for Reference Observation The specified condition determines the reference observation. The output dataset contains all observations before or after (selection parameter) the reference observation.
mode	Selection mode (first or last) If "first" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the *first* time is included in the output dataset. If "last" is specified, for each by group the observations before or after (selection parameter) the observation where the condition (condition parameter) is fulfilled the *last* time is included in the output dataset.
selection	Select observations before or after the reference observation?
inclusive	Include the reference observation?
keep_no_ref_groups	Should by groups without reference observation be kept?
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.

Details

For each by group (by_vars parameter) the observations before or after (selection parameter) the observations where the condition (condition parameter) is fulfilled the first or last time (order parameter and mode parameter) is included in the output dataset.

Value

A dataset containing for each by group the observations before or after the observation where the condition was fulfilled the first or last time

See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_extreme()`, `filter_joined()`, `filter_not_exist()`, `max_cond()`, `min_cond()`

Examples

```
library(tibble)

response <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "CR",
  "1",      4,        "SD",
  "1",      5,        "NE",
  "2",      1,        "SD",
  "2",      2,        "PD",
  "2",      3,        "PD",
  "3",      1,        "SD",
  "4",      1,        "SD",
  "4",      2,        "PR",
  "4",      3,        "PD",
  "4",      4,        "SD",
  "4",      5,        "PR"
)

# Select observations up to first PD for each patient
response %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Select observations after last CR, PR, or SD for each patient
response %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC %in% c("CR", "PR", "SD"),
    mode = "last",
    selection = "after",
    inclusive = FALSE
  )
```

```
# Select observations from first response to first PD
response %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC %in% c("CR", "PR"),
    mode = "first",
    selection = "after",
    inclusive = TRUE,
    keep_no_ref_groups = FALSE
  ) %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )
```

`flag_event` *Create a* `flag_event` *Object*

Description

The `flag_event` object is used to define events as input for the `derive_var_merged_ef_msrc()` function.

Usage

```
flag_event(dataset_name, condition = NULL, by_vars = NULL)
```

Arguments

`dataset_name`	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for `source_datasets` in `derive_var_merged_ef_msrc()`.
`condition`	Condition The condition is evaluated at the dataset referenced by `dataset_name`. For all by groups where it evaluates as `TRUE` at least once the new variable is set to the true value (`true_value`).
`by_vars`	Grouping variables If specified, the dataset is grouped by the specified variables before the condition is evaluated. If named elements are used in `by_vars` like `by_vars = exprs(USUBJID, EXLNKID = ECLNKID)`, the variables are renamed after the evaluation. If the `by_vars` element is not specified, the observations are grouped by the variables specified for the `by_vars` argument of `derive_var_merged_ef_msrc()`.

See Also

derive_var_merged_ef_msrc()

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

---

get_admiral_option *Get the Value of an Admiral Option*

Description

Get the Value of an Admiral Option Which Can Be Modified for Advanced Users.

Usage

```
get_admiral_option(option)
```

Arguments

option	A character scalar of commonly used admiral function inputs. As of now, support only available for "subject_keys", "signif_digits", and "save_memory". See set_admiral_options() for a description of the options.

Details

This function allows flexibility for function inputs that may need to be repeated multiple times in a script, such as subject_keys.

Value

The value of the specified option.

See Also

set_admiral_options(), derive_param_exist_flag(), derive_param_tte() derive_var_dthcaus(), derive_var_extreme_dtm(), derive_vars_period(), create_period_dataset()

Other admiral_options: set_admiral_options()

Examples

```
library(dplyr, warn.conflicts = FALSE)
dm <- tribble(
  ~STUDYID, ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",   "DM", "01-1302",   61, "YEARS",
  "PILOT01",   "DM", "17-1344",   64, "YEARS"
)

vs <- tribble(
```

```
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~VSTESTCD,     ~VISIT,     ~VSTPT, ~VSSTRESN,
  "PILOT01",    "VS", "01-1302",   "DIABP", "BASELINE",    "LYING",        76,
  "PILOT01",    "VS", "01-1302",   "DIABP", "BASELINE", "STANDING",        87,
  "PILOT01",    "VS", "01-1302",   "DIABP",   "WEEK 2",    "LYING",        71,
  "PILOT01",    "VS", "01-1302",   "DIABP",   "WEEK 2", "STANDING",        79,
  "PILOT01",    "VS", "17-1344",   "DIABP", "BASELINE",    "LYING",        88,
  "PILOT01",    "VS", "17-1344",   "DIABP", "BASELINE", "STANDING",        86,
  "PILOT01",    "VS", "17-1344",   "DIABP",   "WEEK 2",    "LYING",        84,
  "PILOT01",    "VS", "17-1344",   "DIABP",   "WEEK 2", "STANDING",        82
)

# Merging all dm variables to vs
derive_vars_merged(
  vs,
  dataset_add = select(dm, -DOMAIN),
  by_vars = get_admiral_option("subject_keys")
)
```

`get_duplicates_dataset`

Get Duplicate Records that Led to a Prior Error

Description

Get Duplicate Records that Led to a Prior Error

Usage

```
get_duplicates_dataset()
```

Details

Many `{admiral}` function check that the input dataset contains only one record per `by_vars` group and throw an error otherwise. The `get_duplicates_dataset()` function allows one to retrieve the duplicate records that lead to an error.

Note that the function always returns the dataset of duplicates from the last error that has been thrown in the current R session. Thus, after restarting the R 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`

See Also

Utilities for Dataset Checking: `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, exprs(USUBJID), cnd_type = "warning")

get_duplicates_dataset()
```

---

get_flagged_records *Create an Existence Flag*

Description

Create a flag variable for the input dataset which indicates if there exists at least one observation in the input dataset fulfilling a certain condition.

Note: This is a helper function for `derive_vars_merged_exist_flag()` which inputs this result into `derive_vars_merged()`.

Usage

```
get_flagged_records(dataset, new_var, condition, filter = NULL)
```

Arguments

`dataset`	Input dataset
`new_var`	New variable The specified variable is added to the input dataset.
`condition`	Condition The condition is evaluated at the dataset (`dataset`). For all rows where it evaluates as `TRUE` the new variable is set to `1` in the new column. Otherwise, it is set to `0`.
`filter`	Filter for additional data Only observations fulfilling the specified condition are taken into account for flagging. If the argument is not specified, all observations are considered.

Value

The output dataset is the input dataset filtered by the `filter` condition and with the variable specified for `new_var` representing a flag for the condition.

See Also

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

Examples

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


ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,    ~AETERM,     ~AEREL,
  "PILOT01",    "AE", "01-1028", "ERYTHEMA", "POSSIBLE",
  "PILOT01",    "AE", "01-1028", "PRURITUS", "PROBABLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "POSSIBLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "PROBABLE"
)


get_flagged_records(
  dataset = ae,
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AERELFL)

vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,      ~VISIT, ~VSTESTCD, ~VSSTRESN, ~VSBLFL,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "HEIGHT",     177.8,      NA,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "01-1028",  "BASELINE",  "WEIGHT",     99.34,     "Y",
  "PILOT01",    "VS", "01-1028",    "WEEK 4",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "HEIGHT",     165.1,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "WEIGHT",     42.87,      NA,
  "PILOT01",    "VS", "04-1127",  "BASELINE",  "WEIGHT",     41.05,     "Y",
  "PILOT01",    "VS", "04-1127",    "WEEK 4",  "WEIGHT",     41.73,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "HEIGHT",    167.64,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "WEIGHT",     57.61,      NA,
  "PILOT01",    "VS", "06-1049",  "BASELINE",  "WEIGHT",     57.83,     "Y",
  "PILOT01",    "VS", "06-1049",    "WEEK 4",  "WEIGHT",     58.97,      NA
)
get_flagged_records(
  dataset = vs,
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  filter = VSTESTCD == "WEIGHT" & VSBLFL == "Y"
) %>%
  select(STUDYID, USUBJID, WTBLHIFL)
```

---

get_many_to_one_dataset

Get Many to One Values that Led to a Prior Error

Description

Get Many to One Values that Led to a Prior Error

Usage

```
get_many_to_one_dataset()
```

Details

If `assert_one_to_one()` detects an issue, the many to one values are stored in a dataset. This dataset can be retrieved by `get_many_to_one_dataset()`.

Note that the function always returns the many to one values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_many_to_one_dataset()` will return `NULL` and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A `data.frame` or `NULL`

See Also

Utilities for Dataset Checking: `get_duplicates_dataset()`, `get_one_to_many_dataset()`

Examples

```
library(admiraldev, warn.conflicts = FALSE)
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, exprs(SITEID), exprs(STUDYID))
)

get_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: `extract_unit()`, `get_flagged_records()`, `get_vars_query()`

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`

See Also

Utilities for Dataset Checking: `get_duplicates_dataset()`, `get_many_to_one_dataset()`

Examples

```
library(admiraldev, warn.conflicts = FALSE)
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, exprs(STUDYID), exprs(SITEID))
)

get_one_to_many_dataset()
```

get_summary_records *Create Summary Records*

Description

[Deprecated] The get_summary_records() has been deprecated in favor of derive_summary_records() (call it with the dataset_add argument and without the dataset argument).

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, set_values_to = NULL)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset.
by_vars	Grouping variables Variables to consider for generation of groupwise summary records.
filter	Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved. For example, • filter_rows = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars. • filter_rows = (dplyr::n() > 2) will filter n count of by_vars greater than 2.

set_values_to	Variables to be set
	The specified variables are set to the specified values for the new observations.
	Set a list of variables to some specified value for the new records
	• LHS refer to a variable.
	• RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for by_vars.
	For example:
	`set_values_to = exprs(`
	`  AVAL = sum(AVAL),`
	`  PARAMCD = "TDOSE",`
	`  PARCAT1 = "OVERALL"`
	`)`

Details

This function only creates derived observations and does not append them to the original dataset observations. If you would like to this instead, see the `derive_summary_records()` function.

Value

A data frame of derived records.

See Also

`derive_summary_records()`, `derive_vars_merged_summary()`

Other deprecated: `call_user_fun()`, `date_source()`, `derive_param_extreme_record()`, `derive_var_dthcaus()`, `derive_var_extreme_dt()`, `derive_var_extreme_dtm()`, `derive_var_merged_summary()`, `dthcaus_source()`

Examples

```
library(tibble)

adeg <- tribble(
  ~USUBJID,   ~EGSEQ, ~PARAM,             ~AVISIT,    ~EGDTC,             ~AVAL, ~TRTA,
 "XYZ-1001", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385,  NA_character_,
 "XYZ-1001", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399,  NA_character_,
 "XYZ-1001", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396,  NA_character_,
 "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,  NA_character_,
 "XYZ-1002", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410,  NA_character_,
 "XYZ-1002", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392,  NA_character_,
 "XYZ-1002", 4,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:50", 401,  "Active 20mg",
 "XYZ-1002", 5,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:53", 407,  "Active 20mg",
 "XYZ-1002", 6,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:56", 400,  "Active 20mg",
 "XYZ-1002", 7,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:50", 412,  "Active 20mg",
 "XYZ-1002", 8,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:53", 414,  "Active 20mg",
 "XYZ-1002", 9,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:56", 402,  "Active 20mg"
)

# Summarize the average of the triplicate ECG interval values (AVAL)
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
)

# Derive more than one summary variable
```

```
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL),
    ASTDTM = min(convert_dtc_to_dtm(EGDTC)),
    AENDTM = max(convert_dtc_to_dtm(EGDTC)),
    DTYPE = "AVERAGE"
  )
)

# Sample ADEG dataset with triplicate record for only AVISIT = 'Baseline'
adeg <- tribble(
  ~USUBJID,   ~EGSEQ, ~PARAM,             ~AVISIT,    ~EGDTC,             ~AVAL, ~TRTA,
 "XYZ-1001", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385,  NA_character_,
 "XYZ-1001", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399,  NA_character_,
 "XYZ-1001", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396,  NA_character_,
 "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,  NA_character_,
 "XYZ-1002", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410,  NA_character_,
 "XYZ-1002", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392,  NA_character_,
 "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 = exprs(USUBJID, PARAM, AVISIT),
  filter = n() > 2,
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
)
```

`get_vars_query` *Get Query Variables*

Description

Create a table for the input dataset which binds the necessary rows for a `derive_vars_query()` call with the relevant `SRCVAR`, `TERM_NAME_ID` and a temporary index if it is necessary

Note: This function is the first step performed in `derive_vars_query()` requested by some users to be present independently from it.

Usage

```
get_vars_query(dataset, dataset_queries)
```

Arguments

dataset	Input dataset
dataset_queries	A dataset containing required columns PREFIX, GRPNAME, SRCVAR, TERMCHAR and/or TERMNUM, and optional columns GRPID, SCOPE, SCOPEN. 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 vignette("occds").

A query dataset is expected as an input to this function. See the vignette("queries_dataset") for descriptions, or call data("queries") for an example of a query dataset.

For each unique element in PREFIX, the corresponding "NAM" variable will be created. For each unique PREFIX, if GRPID is not "" or NA, then the corresponding "CD" variable is created; similarly, if SCOPE is not "" or NA, then the corresponding "SC" variable will be created; if SCOPEN is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in dataset, the "NAM" variable takes the value of GRPNAME if the value of TERMCHAR or TERMNUM in dataset_queries matches the value of the respective SRCVAR in dataset. Note that TERMCHAR in dataset_queries dataset may be NA only when TERMNUM is non-NA and vice versa. The matching is case insensitive. The "CD", "SC", and "SCN" variables are derived accordingly based on GRPID, SCOPE, and SCOPEN respectively, whenever not missing.

Value

The processed query dataset with SRCVAR and TERM_NAME_ID so that that can be merged to the input dataset to execute the derivations outlined by dataset_queries.

See Also

create_query_data()

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

Examples

```
library(tibble)
data("queries")
adae <- tribble(
  ~USUBJID, ~ASTDTM, ~AETERM, ~AESEQ, ~AEDECOD, ~AELLT, ~AELLTCD,
  "01", "2020-06-02 23:59:59", "ALANINE AMINOTRANSFERASE ABNORMAL",
  3, "Alanine aminotransferase abnormal", NA_character_, NA_integer_,
  "02", "2020-06-05 23:59:59", "BASEDOW'S DISEASE",
  5, "Basedow's disease", NA_character_, 1L,
```

```
  "03", "2020-06-07 23:59:59", "SOME TERM",
  2, "Some query", "Some term", NA_integer_,
  "05", "2020-06-09 23:59:59", "ALVEOLAR PROTEINOSIS",
  7, "Alveolar proteinosis", NA_character_, NA_integer_
)
get_vars_query(adae, queries)
```

`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"` (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, `NA_character_` is returned. If `"Y"` (year, highest level) is specified, `date_imputation` must be `"first"` or `"last"` and `min_dates` or `max_dates` must be specified respectively. Otherwise, an error is thrown.
`date_imputation`	The value to impute the day/month when a datepart is missing. A character value is expected.

- If highest_imputation is "M", month and day can be specified as "mm-dd": e.g. "06-15" for the 15th of June
- When highest_imputation is "M" or "D", the following keywords are available: "first", "mid", "last" to impute to the first/mid/last day/month. If "mid" is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).

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).

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").

Details

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

Value

A character vector

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(), 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_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "01-01"
)

# Impute to last day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "last",
)

# Impute to mid day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "mid"
)

# Impute a date and ensure that the imputed date is not before a list of
# minimum dates
```

```
impute_dtc_dt(
  "2020-12",
  min_dates = list(
    ymd("2020-12-06"),
    ymd("2020-11-11")
  ),
  highest_imputation = "M"
)

# Impute completely missing dates (only possible if min_dates or max_dates is specified)
impute_dtc_dt(
  c("2020-12", NA_character_),
  min_dates = list(
    ymd("2020-12-06", "2020-01-01"),
    ymd("2020-11-11", NA)
  ),
  highest_imputation = "Y"
)
```

`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.

date_imputation

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

A character value is expected.

- If highest_imputation is "M", month and day can be specified as "mm-dd": e.g. "06-15" for the 15th of June
- When highest_imputation is "M" or "D", the following keywords are available: "first", "mid", "last" to impute to the first/mid/last day/month. If "mid" is specified, missing components are imputed as the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).

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).

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".

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_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  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 lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

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

Details

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

Value

A character vector

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(), 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_hm("2020-12-06T12:12"),
```

```
    ymd_hm("2020-11-11T11:11")
  ),
  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_hm("2020-12-06T12:12", "2020-01-01T01:01"),
    ymd_hm("2020-11-11T11:11", NA)
  ),
  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

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

See Also

Other Advanced Functions: params()

Examples

```
list_tte_source_objects()
```

max_cond *Maximum Value on a Subset*

Description

The function derives the maximum value of a vector/column on a subset of entries/observations.

Usage

```
max_cond(var, cond)
```

Arguments

var	A vector
cond	A condition

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), min_cond()

Examples

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

# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR
group_by(data, USUBJID) %>% mutate(
  first_cr_vis = min_cond(var = AVISITN, cond = AVALC == "CR"),
  last_pr_vis = max_cond(var = AVISITN, cond = AVALC == "PR"),
  pr_after_cr = last_pr_vis > first_cr_vis
)
```

---

min_cond *Minimum Value on a Subset*

Description

The function derives the minimum value of a vector/column on a subset of entries/observations.

Usage

```
min_cond(var, cond)
```

Arguments

var	A vector
cond	A condition

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond()

Examples

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

# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR
group_by(data, USUBJID) %>% mutate(
  first_cr_vis = min_cond(var = AVISITN, cond = AVALC == "CR"),
  last_pr_vis = max_cond(var = AVISITN, cond = AVALC == "PR"),
  pr_after_cr = last_pr_vis > first_cr_vis
)
```

---

negate_vars *Negate List of Variables*

Description

The function adds a minus sign as prefix to each variable.

Usage

```
negate_vars(vars = NULL)
```

Arguments

vars	List of variables created by `exprs()`

Details

This is useful if a list of variables should be removed from a dataset, e.g., `select(!!!negate_vars(by_vars))` removes all by variables.

Value

A list of expressions

See Also

Utilities for working with quosures/list of expressions: chr2vars()

Examples

```
negate_vars(exprs(USUBJID, STUDYID))
```

---

params *Create a Set of Parameters*

Description

Create a set of variable parameters/function arguments to be used in call_derivation().

Usage

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

Arguments

...	One or more named arguments

Value

An object of class params

See Also

call_derivation()

Other Advanced Functions: list_tte_source_objects()

Examples

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

adsl <- tribble(
  ~STUDYID,   ~USUBJID,      ~TRTSDT,      ~TRTEDT,
  "PILOT01", "01-1307",           NA,           NA,
  "PILOT01", "05-1377", "2014-01-04", "2014-01-25",
  "PILOT01", "06-1384", "2012-09-15", "2012-09-24",
  "PILOT01", "15-1085", "2013-02-16", "2013-08-18",
  "PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
  mutate(
    across(TRTSDT:TRTEDT, as.Date)
  )

ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,     ~AESTDTC,     ~AEENDTC,
```

```
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06"
)

adae <- ae %>%
  select(USUBJID, AESTDTC, AEENDTC) %>%
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = exprs(TRTSDT, TRTEDT),
    by_vars = exprs(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 = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  )


## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go.
## 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 = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the call using `derive_vars_dt()`
## to derive variables `ASTDT` and `AENDT` separately at the beginning.
```

`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 Standardized MedDRA Query variables in ADAE

See Also

Other datasets: `admiral_adlb`, `admiral_adsl`, `ex_single`, `example_qs`, `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 Standardized MedDRA Query variables in ADMH

See Also

Other datasets: `admiral_adlb`, `admiral_adsl`, `ex_single`, `example_qs`, `queries`

`query` *Create an* `query` *object*

Description

A `query` object defines a query, e.g., a Standard MedDRA Query (SMQ), a Standardized Drug Grouping (SDG), or a customized query (CQ). It is used as input to `create_query_data()`.

Usage

```
query(prefix, name = auto, id = NULL, add_scope_num = FALSE, definition = NULL)
```

Arguments

`prefix`	The value is used to populate `PREFIX` in the output dataset of `create_query_data()`, e.g., `"SMQ03"`
`name`	The value is used to populate `GRPNAME` in the output dataset of `create_query_data()`. If the `auto` keyword is specified, the variable is set to the name of the query in the SMQ/SDG database.
`id`	The value is used to populate `GRPID` in the output dataset of `create_query_data()`. If the `auto` keyword is specified, the variable is set to the id of the query in the SMQ/SDG database.
`add_scope_num`	Determines if `SCOPEN` in the output dataset of `create_query_data()` is populated If the parameter is set to `TRUE`, the definition must be an `basket_select()` object. *Default*: `FALSE`
`definition`	Definition of terms belonging to the query There are three different ways to define the terms: • An `basket_select()` object is specified to select a query from the SMQ database.

- A data frame with columns SRCVAR and TERMCHAR or TERMNUM can be specified to define the terms of a customized query. The SRCVAR should be set to the name of the variable which should be used to select the terms, e.g., "AEDECOD" or "AELLTCD". SRCVAR does not need to be constant within a query. For example a query can be based on AEDECOD and AELLT.
  If SRCVAR refers to a character variable, TERMCHAR should be set to the value the variable. If it refers to a numeric variable, TERMNUM should be set to the value of the variable. If only character variables or only numeric variables are used, TERMNUM or TERMCHAR respectively can be omitted.
- A list of data frames and basket_select() objects can be specified to define a customized query based on custom terms and SMQs. The data frames must have the same structure as described for the previous item.

Value

An object of class query.

See Also

create_query_data(), basket_select(), vignette("queries_dataset")

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), records_source(), tte_source()

Examples

```
# create a query for an SMQ
library(tibble)
library(dplyr, warn.conflicts = FALSE)

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

# create a query for an SDG
query(
  prefix = "SDG01",
  id = auto,
  definition = basket_select(
    name = "5-aminosalicylates for ulcerative colitis",
    scope = NA_character_,
    type = "sdg"
  )
)
```

```
# creating a query for a customized query
cqterms <- tribble(
  ~TERMCHAR, ~TERMNUM,
  "APPLICATION SITE ERYTHEMA", 10003041L,
  "APPLICATION SITE PRURITUS", 10003053L
) %>%
  mutate(SRCVAR = "AEDECOD")

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

# creating a customized query based on SMQs and additional terms
query(
  prefix = "CQ03",
  name = "Special issues of interest",
  definition = list(
    cqterms,
    basket_select(
      name = "Pregnancy and neonatal topics (SMQ)",
      scope = "NARROW",
      type = "smq"
    ),
    basket_select(
      id = 8050L,
      scope = "BROAD",
      type = "smq"
    )
  )
)
```

`records_source` *Create a* `records_source` *Object*

Description

The `records_source` object is used to find extreme records of interest.

Usage

```
records_source(dataset_name, filter = NULL, new_vars)
```

Arguments

`dataset_name`	The name of the source dataset The name refers to the dataset provided by the `source_datasets` argument of `derive_param_extreme_record()`.

filter	An unquoted condition for selecting the observations from dataset.
new_vars	Variables to add The specified variables from the source datasets are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>). For example new_vars = exprs(var1, var2) adds variables var1 and var2 from to the input dataset. And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from the source dataset and adds them to the input dataset renaming old_var2 to new_var2. Expressions can be used to create new variables (see for example new_vars argument in derive_vars_merged()).

Value

An object of class records_source

See Also

derive_param_extreme_record()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), query(), tte_source()

---

restrict_derivation *Execute a Derivation on a Subset of the Input Dataset*

Description

Execute a derivation on a subset of the input dataset.

Usage

```
restrict_derivation(dataset, derivation, args = NULL, filter)
```

Arguments

dataset	Input dataset
derivation	Derivation A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the params() objects passed to the args argument.
args	Arguments of the derivation A params() object is expected.
filter	Filter condition

Details

It is also possible to pass functions from outside the {admiral} package to restrict_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

See Also

params() slice_derivation() call_derivation()

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

Examples

```
library(tibble)

adlb <- 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 = exprs(USUBJID),
    dataset_add = adlb,
    filter_add = ABLFL == "Y",
    new_vars = exprs(BASE = AVAL)
  ),
  filter = AVISITN > 0
)

# Derive BASE for baseline and post-baseline records only
restrict_derivation(
  adlb,
  derivation = derive_var_base,
  args = params(
    by_vars = exprs(USUBJID)
  ),
  filter = AVISITN >= 0
) %>%
  # Derive CHG for post-baseline records only
  restrict_derivation(
    derivation = derive_var_chg,
```

```
  filter = AVISITN > 0
)
```

`set_admiral_options` *Set the Value of admiral Options*

Description

Set the values of admiral options that can be modified for advanced users.

Usage

```
set_admiral_options(subject_keys, signif_digits, save_memory)
```

Arguments

`subject_keys`	Variables to uniquely identify a subject, defaults to `exprs(STUDYID, USUBJID)`. This option is used as default value for the `subject_keys` argument in all admiral functions.
`signif_digits`	Holds number of significant digits when comparing to numeric variables, defaults to 15. This option is used as default value for the `signif_dig` argument in admiral functions `derive_var_atoxgr_dir()` and `derive_var_anrind()`.
`save_memory`	If set to `TRUE`, an alternative algorithm is used in the functions `derive_vars_joined()`, `derive_var_joined_exist_flag()`, `derive_extreme_event()`, and `filter_joined()` which requires less memory but more run-time.

Details

Modify an admiral option, e.g `subject_keys`, such that it automatically affects downstream function inputs where `get_admiral_option()` is called such as `derive_param_exist_flag()`.

Value

No return value, called for side effects.

See Also

`get_admiral_option()`, `derive_param_exist_flag()`,`derive_param_tte()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_vars_period()`, `create_period_dataset()`, `derive_var_atoxgr_dir()`, `derive_var_anrind()`

Other admiral_options: `get_admiral_option()`

Examples

```
library(lubridate)
library(dplyr, warn.conflicts = FALSE)
library(tibble)
set_admiral_options(subject_keys = exprs(STUDYID, USUBJID2))

# Derive a new parameter for measurable disease at baseline
adsl <- tribble(
  ~USUBJID2,
  "1",
  "2",
  "3"
) %>%
  mutate(STUDYID = "XX1234")

tu <- tribble(
  ~USUBJID2,      ~VISIT,    ~TUSTRESC,
  "1",       "SCREENING",     "TARGET",
  "1",          "WEEK 1",     "TARGET",
  "1",          "WEEK 5",     "TARGET",
  "1",          "WEEK 9", "NON-TARGET",
  "2",       "SCREENING", "NON-TARGET",
  "2",       "SCREENING", "NON-TARGET"
) %>%
  mutate(
    STUDYID = "XX1234",
    TUTESTCD = "TUMIDENT"
  )

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

set_admiral_options(signif_digits = 14)

# Derive ANRIND for ADVS
advs <- tribble(
  ~PARAMCD, ~AVAL, ~ANRLO, ~ANRHI,
  "DIABP",     59,     60,     80,
  "SYSBP",    120,     90,    130,
  "RESP",      21,      8,     20,
)
```

```
derive_var_anrind(advs)
```

`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 A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the `params()` object passed to the `args` argument or be provided in *every* `derivation_slice()`.
`...`	A `derivation_slice()` object is expected Each slice defines a subset of the input dataset and some of the parameters for the derivation. The derivation is called on the subset with the parameters specified by the `args` parameter and the `args` field of the `derivation_slice()` object. If a parameter is specified for both, the value in `derivation_slice()` overwrites the one in `args`.
`args`	Arguments of the derivation A `params()` object is expected.

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.
- The derivation is called for slices with no observations.
- Observations with no match to any of the slices are included in the output dataset but the derivation is not called for them.

It is also possible to pass functions from outside the {admiral} package to slice_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

Value

The input dataset with the variables derived by the derivation added

See Also

params() restrict_derivation() call_derivation()

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

Examples

```
library(tibble)
library(stringr)
advs <- tribble(
  ~USUBJID, ~VSDTC,       ~VSTPT,
  "1",      "2020-04-16", NA_character_,
  "1",      "2020-04-16", "BEFORE TREATMENT"
)

# For the second slice filter is set to TRUE. Thus derive_vars_dtm is called
# with time_imputation = "last" for all observations which do not match for the
# first slice.
slice_derivation(
  advs,
  derivation = derive_vars_dtm,
  args = params(
    dtc = VSDTC,
    new_vars_prefix = "A"
  ),
  derivation_slice(
    filter = str_detect(VSTPT, "PRE|BEFORE"),
    args = params(time_imputation = "first")
  ),
  derivation_slice(
    filter = TRUE,
    args = params(time_imputation = "last")
  )
)
```

---

transform_range *Transform Range*

Description

Transforms results from the source range to the target range. For example, for transforming source values 1, 2, 3, 4, 5 to 0, 25, 50, 75, 100.

Usage

```
transform_range(
  source,
  source_range,
  target_range,
  flip_direction = FALSE,
  outside_range = "NA"
)
```

Arguments

source	A vector of values to be transformed A numeric vector is expected.
source_range	The permitted source range A numeric vector containing two elements is expected, representing the lower and upper bounds of the permitted source range.
target_range	The target range A numeric vector containing two elements is expected, representing the lower and upper bounds of the target range.
flip_direction	Flip direction of the range? The transformed values will be reversed within the target range, e.g. within the range 0 to 100, 25 would be reversed to 75.
outside_range	Handling of values outside the source range Values outside the source range (source_range) are transformed to NA. If "warning" or "error" is specified, a warning or error is issued if source includes any values outside the source range.

Details

Returns the values of source linearly transformed from the source range (source_range) to the target range (target_range). Values outside the source range are set to NA.

Value

The source linearly transformed to the target range

See Also

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

Examples

```
transform_range(
  source = c(1, 4, 3, 6, 5),
  source_range = c(1, 5),
  target_range = c(0, 100)
)

transform_range(
  source = c(1, 4, 3, 6, 5),
  source_range = c(1, 5),
  target_range = c(0, 100),
  flip_direction = TRUE
)
```

---

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,
  order = order
)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte().
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points.
date	A variable or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object.
censor	Censoring value CDISC strongly recommends using 0 for events and positive integers for censoring.

set_values_to	A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.
order	Sort order An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date.

Value

An object of class tte_source

See Also

derive_param_tte(), censor_source(), event_source()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), query(), records_source()

---

use_ad_template *Open an ADaM Template Script*

Description

Open an ADaM Template Script

Usage

```
use_ad_template(
  adam_name = "adsl",
  save_path = paste0("./", adam_name, ".R"),
  package = "admiral",
  overwrite = FALSE,
  open = interactive()
)
```

Arguments

adam_name	An ADaM dataset name. You can use any of the available dataset names "ADAB", "ADAE", "ADCM", "ADEG", "ADEX", "ADLB", "ADLBHY", "ADMH", "ADPC", "ADPP", "ADPPK", "ADSL", "ADVS". The dataset name is case-insensitive. The default dataset name is "ADSL".
save_path	Path to save the script.
package	The R package in which to look for templates. By default "admiral".
overwrite	Whether to overwrite an existing file named save_path.
open	Whether to open the script right away.

Details

Running without any arguments such as `use_ad_template()` auto-generates `adsl.R` in the current path. Use `list_all_templates()` to discover which templates are available.

Value

No return values, called for side effects

See Also

Utilities used for examples and template scripts: `list_all_templates()`

Examples

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

---

yn_to_numeric *Map "Y" and "N" to Numeric Values*

Description

Map `"Y"` and `"N"` to numeric values.

Usage

```
yn_to_numeric(arg)
```

Arguments

`arg`	Character vector

Value

`1` if `arg` equals `"Y"`, `0` if `arg` equals `"N"`, `NA_real_` otherwise

See Also

Utilities for Formatting Observations: `convert_blanks_to_na()`, `convert_na_to_blanks()`

Examples

```
yn_to_numeric(c("Y", "N", NA_character_))
```

`%>%` *Pipe operator*

Description

See `magrittr::%>%` for more details.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	A value or the magrittr placeholder.
`rhs`	A function call using the magrittr semantics.

Index